Version 1.3.0:2

Introduction

ActiveServer is a **3D Secure 2 Server** solution provided by **GPayments Pty Ltd**.

GPayments also provides a **3D Secure 2 mobile SDK** solution, **ActiveSDK**. Contact us for more information regarding our SDK solution.

Using this document

Version and language

By default, the latest documentation version of the current **ActiveServer** release will be shown. The **Doc Version** drop down menu in the menu bar can be used to view previous documentation releases if required. The latest documentation release for each software version will be shown. To go back to the latest release, choose **Latest**.

The **Language** drop down menu can be used to switch between different language versions, with English and Japanese currently supported.

Navigation

There is a **Chapter menu** on the left hand side of every page. You can jump to any chapter by selecting from the menu. The **Chapter menu** may not appear on the left side if your window is too narrow, but you will still be able to access it via the hamburger button in the top left corner.

Each page also contains a **Table of Contents** on the right hand side, which lists the sub-sections in that page. You can skip to any sub-section by selecting from the list. The **Table of Contents** may not appear if your window width is too narrow or simply too small, in which case you may need to resize your window for it to appear.

A permanent link to a section can be created by clicking the ¶ icon next to any heading and then copying the link in the address bar.

You can search in the documentation by typing any phrase into the **Search** box in the top right hand corner of the screen. Search results are shown for all pages containing that phrase and link to the relevant part of the documentation.

Download as PDF

The documentation is available to be exported as a PDF by selecting the **Export ActiveServer Documentation** icon in the top right hand corner of any page. This will export the entire documentation in a PDF format if required.

Important

Note that all changes are made to the live documentation site. If you use the export PDF feature, we strongly recommend checking back routinely to see if a new documentation release has been added for your software version.

Overview of this documentation

This documentation provides an introduction to ActiveServer, guides you through the integration process, and provides troubleshooting procedures.

This documentation is organized into the following chapters:

- **Introduction** - an introduction to ActiveServer, and an overview of this document.
- **Quickstart** - installation instructions, tips and essential information to get ActiveServer up and running.
- **Features overview** - descriptions of the main features of the system.
- **Guides** - extensive guides on using system functionality and other walkthroughs for 3DS2 tasks.
- **API References** - a overview on how APIs can be used, links to API reference documents and an error code listing.

- **Glossary** - 3DS2 and ActiveServer specific terms, abbreviations and definitions used throughout the document.
- **Document control** - change log for the document.
- **Release notes** - ActiveServer software release notes by version.
- **Legal notices** - confidentiality, copyright, disclaimer and liability statements.

Product introduction

ActiveServer is a 3D Secure 2 **3DS Server** solution for merchants and Payment Service Providers. ActiveServer allows merchants to implement 3DS2 for their payment flows and have guaranteed protection against Card Not Present (CNP) fraud via Liability Shift. It supports all the major card brands, is fully PCI-DSS 3.2 ready and is simple to integrate with its easy to use APIs.

ActiveServer offers the flexibility of **In-house** deployment or can be utilised from GPayments' **Hosted Service**.

Core features

ActiveServer comes with the following core features:

- **Intelligent Reporting** - key business information available from reporting functionality provided through the administration web application.
- **Application Server and OS Agnostic** - ability to utilise any popular web container to launch ActiveServer via a WAR file or deploy as a standalone application utilising Spring. This extends to all popular operating systems including Windows and Linux based systems.
- **HSM Agnostic** - compatibility with most major general purpose Hardware Security Modules for encryption, including Thales, Gemalto, AWS KMS, or any PKCS11 compatible HSM's.
- **Easy Product Activation** - simple management of all ActiveServer instances deployed via a token-based activation procedure linked to the organisation's account with GPayments.
- **Multiple 3DS Requestors and Merchants** - ability to add multiple 3DS Requestors and merchants to the same ActiveServer instance.
- **Ease of Migration** - for existing customers, GPayments is available to develop a plan and identify the tools needed for migrating to ActiveServer

ActiveServer APIs

ActiveServer offers easy to use RESTful APIs, based on industry standards, for merchants to integrate with their existing systems. Requests can be sent and received in JSON format. The APIs come with detailed documentation and sample code to offer a seamless experience.

Authentication API

ActiveServer exposes its authentication components to allow merchants to embed API code within their existing checkout process for browser and mobile. This code calls ActiveServer to perform the authentication and return the authentication response. This is a flexible model, which offers merchants the ability to utilise ActiveServer remotely, over the Internet, from the merchant's own network.

Admin API

ActiveServer exposes an API to its administration services, enabling system administrators and developers to integrate merchant and acquirer management tasks with existing infrastructure. The Admin API is particularly useful for merchant aggregators and payment gateways, who already maintain and manage some of the merchant information required for setting up merchant profiles. It allows ActiveServer's merchant management tasks to be integrated within a merchant's own system and significantly reduces administration overhead.

About GPayments

GPayments

GPayments is an Australian company, with clients worldwide, that specialises in delivering payment authentication products for online transactions. We provide a range of solutions for card schemes, financial institutions (both issuers and acquirers), online service providers, merchants, and cardholders. Our 3DS2 application suite includes ActiveAccess (ACS), ActiveServer (3DS Server) and ActiveSDK (3DS mobile SDK). GPayments also provides clients with access to its 3DS2 TestLabs, for end-to-end system integration testing. TestLabs has a live and fully developed Directory Server, Mobile SDK and EMVCo-compliant ACS.

Further information about GPayments is available on our website at https://www.gpayments.com/ or by contacting sales@gpayments.com.

If you find any errors in this documentation or would like to contact us for additional support, please email GPayments Tech Support at techsupport@gpayments.com.

Quickstart

Quickstart is intended to guide you smoothly through downloading, setting up, and running ActiveServer. For specific user guides on how to configure and manage ActiveServer, refer to the **Guides** section, two menu items below this one.

Prerequisites

Specifications:

Specifications	Details
Operating System	Linux, Windows Server
Memory	2 GB RAM (recommended)
Disk Space	No minimum requirements, but ensure that sufficient disk space is available for the database, as all data is stored in the database.
Java Development Kit	Java SE Development Kit 8 (Open JDK v1.8)
Java Container	The `.jar` file can run in any container that supports Servlet 2.4/JSP 2.0 or later. *Default container is* `UnderTow`.
Web Browser	The Administration UI can be accessed using Google Chrome, Mozilla Firefox, or Microsoft Edge.

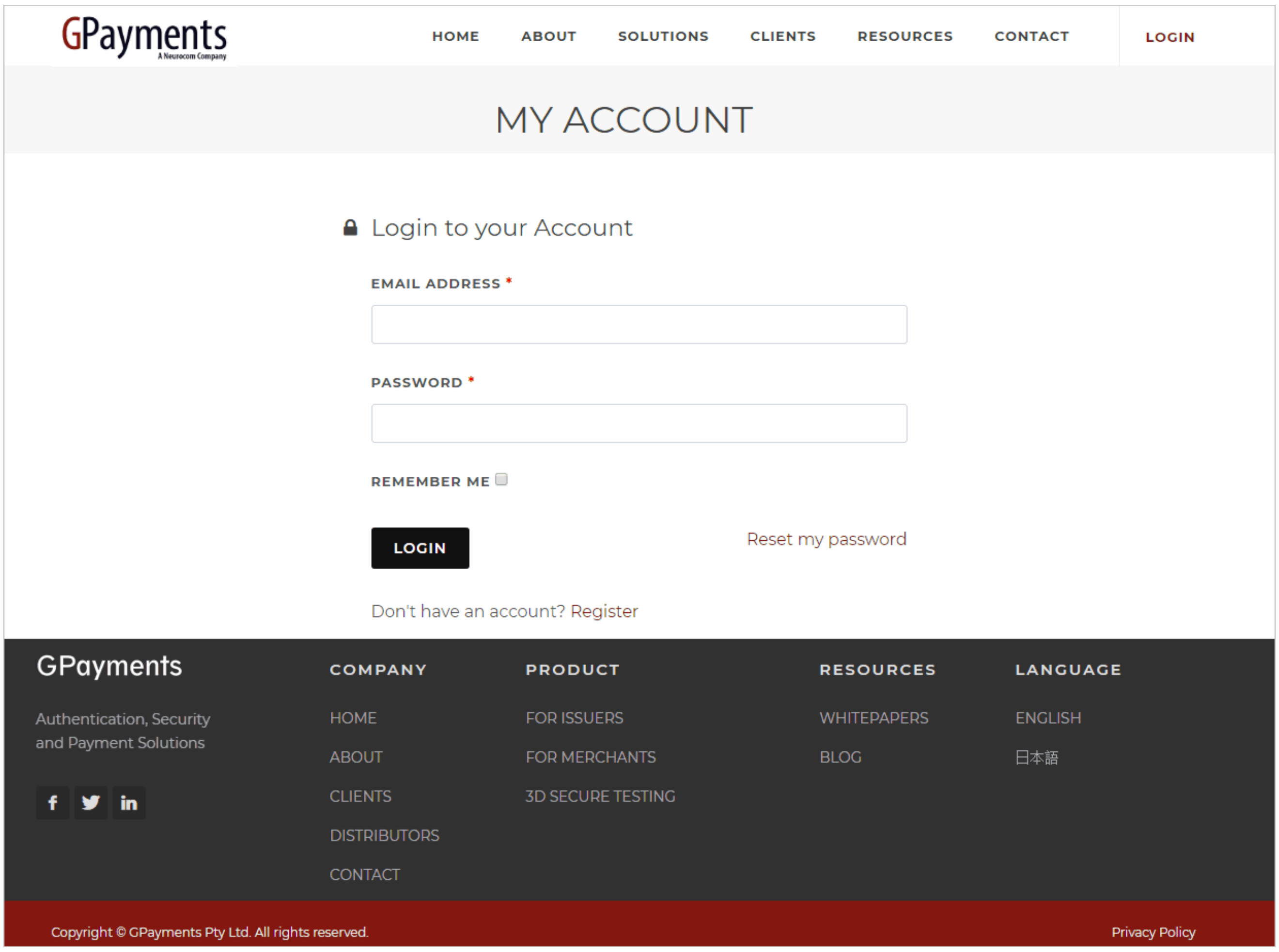
Database:

Database	Compatible Versions
MySQL / Amazon Aurora MySQL	5.7
Oracle	11g, 12c
Microsoft SQL Server	2008 R2, 2012, 2014, 2016, 2017

Database	Compatible Versions
PostgreSQL	8.4 and later
IBM Db2	11.1 and later

Download ActiveServer

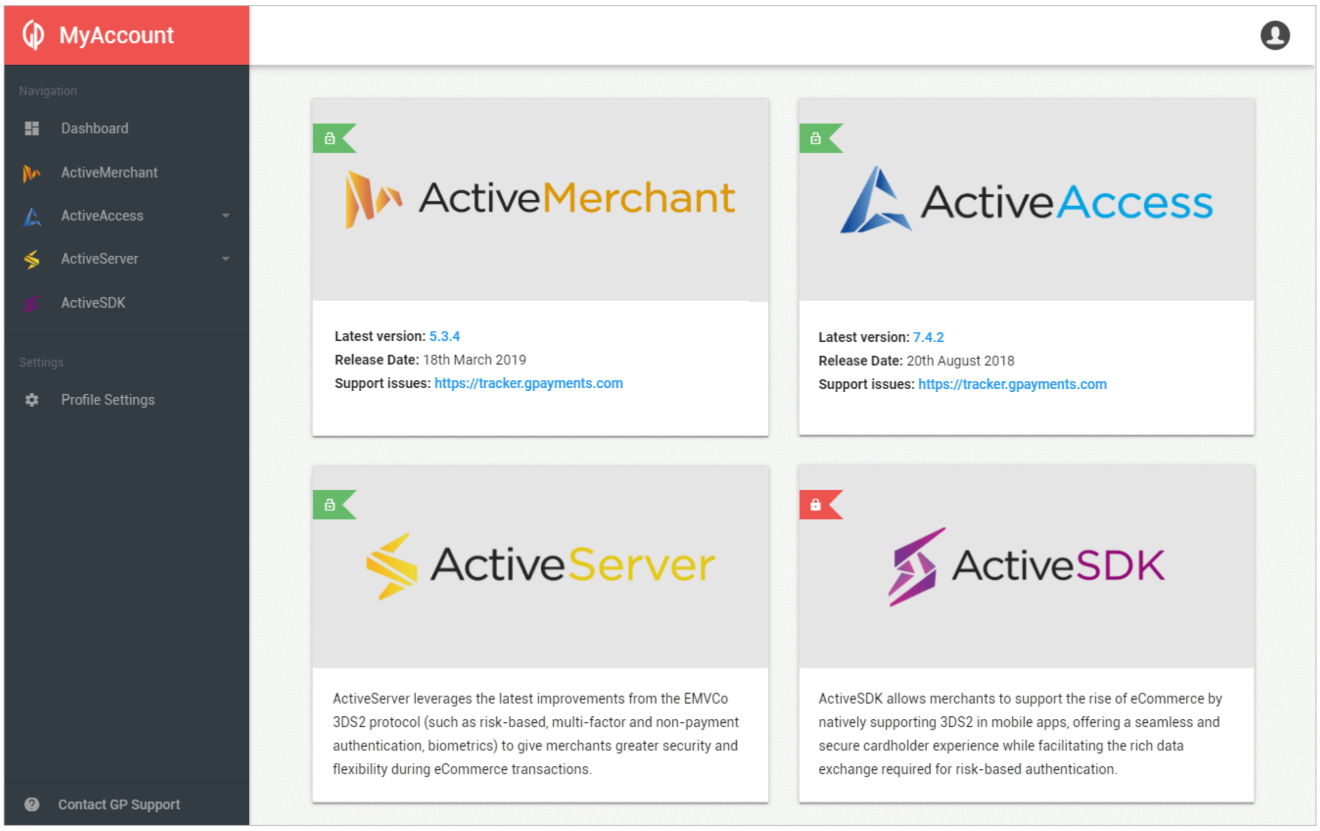
1. Login to GPayments **MyAccount** at https://login.gpayments.com/login.



If you do not have an account...

If you do not have a GPayments MyAccount, register at: https://login.gpayments.com/register

2. Once logged in, you will see the **MyAccount Dashboard**.



Latest version: 5.3.4
Release Date: 18th March 2019
Support issues: https://tracker.gpayments.com

Latest version: 7.4.2
Release Date: 20th August 2018
Support issues: https://tracker.gpayments.com

ActiveServer leverages the latest improvements from the EMVCo 3DS2 protocol (such as risk-based, multi-factor and non-payment authentication, biometrics) to give merchants greater security and flexibility during eCommerce transactions.

ActiveSDK allows merchants to support the rise of eCommerce by natively supporting 3DS2 in mobile apps, offering a seamless and secure cardholder experience while facilitating the rich data exchange required for risk-based authentication.

3. From the Navigation Menu on the left, select **ActiveServer > Download**.
4. Select the release package to start the download.

MyAccount organisations

MyAccount utilises an organisation structure for clients to share product privileges between company users, such as downloading the software, and managing instance activation and licensing.

When downloading the software, if you are not part of an organisation yet, you will be prompted to either **Register** an organisation, or ask your point of contact with GPayments to invite you to your existing organisation. Inviting a user to an organisation can be done from the **MyAccount > Profile Settings > My Organisation** section.

Important

To access your purchased software and manage existing instances for your organisation, confirm your company does not already have an organisation setup before creating a new one.

Once part of an organisation that has already purchased **ActiveServer**, you should be able to download the package. If you are unable to download the software, but have confirmed you are in the correct organisation that has already purchased the product, please contact GPayments Tech Support at techsupport@gpayments.com for assistance. Or, if you would like to talk to our friendly sales team about purchasing **ActiveServer**, you can contact GPayments Sales at sales@gpayments.com.

Installation

Extract the downloaded `.zip` file, and you should see the files below.

```
ActiveServer_vX.XX/
├── application-prod.properties
├── as.jar
├── README.txt
├── release.txt
├── startup.bat
└── startup.sh
```

The files are:

- `application-prod.properties` - Configuration file to initialize ActiveServer.
- `as.jar` - The main ActiveServer Java package.
- `README.txt` - General information about ActiveServer, as well as its documentation, licensing, and support.
- `release.txt` - Release notes for all versions of ActiveServer.
- `startup.bat` - The startup script for Windows.
- `startup.sh` - The startup script for Linux.

Configuration

Configure the system properties for ActiveServer by editing the `application-prod.properties` file.

You can startup ActiveServer without configuring system properties, but...

The `application-prod.properties` file in the package you have downloaded includes default application properties. If you start up an instance of ActiveServer using these default application properties, ActiveServer will:

- Use a default database that will only be temporarily stored in the RAM, and will be wiped once the ActiveServer instance is shut down.
- Create a keystore file with SunJCE that is stored locally in `${AS_HOME}/conf/security/` .
- Skip email server configurations, therefore disabling the system from sending email notifications to users.

Using the default properties may be useful for trying out the software and its interface as it allows you to quickly startup an instance of ActiveServer without having to change any settings. However, you must configure the system properties before setting up a production instance.

To change the default application properties:

Open the file `application-prod.properties` and change the corresponding values associated with each relevant parameter.

There are 4 categories of system properties:

- Database settings
- Web server settings
- Keystore settings
- Email server settings

Database settings

ActiveServer supports the following databases:

- MySQL / Amazon Aurora MySQL
- Oracle
- Microsoft SQL Server
- PostgreSQL
- IBM Db2

Each database has the following set of configurable properties:

- `as.db.vendor=`
  Database vendor/type. Default value is empty, where an in-memory test database will be used. You cannot enter production with the in-memory test database. Possible values are `mysql`, `oracle` or `sqlserver`.
- `as.db.url=`
  The database connection URL used to connect to your database. The URL must be in JDBC format.
- `as.db.username=`
  Database username. Enter the username set by your database administrator.
- `as.db.password=`
  Password. Enter the password set by your database administrator.
- `as.db.password-plain=`
  Whether to encrypt the password above or not. ActiveServer can encrypt the password when storing it in the `application-prod.properties` file. To enable password encryption, enter `false`. To leave the password as plain text, enter `true`.

MySQL / Amazon Aurora MySQL

To use a MySQL or Amazon Aurora MySQL database, you will need the following properties:

MySQL Database Properties (Example)

```
as.db.vendor=mysql
as.db.url=jdbc:mysql://<Your My SQL DB Host>:3306/<Your DB Name>?
useUnicode=true&characterEncoding=utf8&useSSL=false
as.db.username=<user name>
as.db.password=<password>
as.db.password-plain=true
```

Oracle

To use a Oracle database, you will need the following properties:

Oracle Database Properties (Example)

```
as.db.vendor=oracle
as.db.url=jdbc:oracle:thin:@//<Your Oracle DB Host>:1521/<Your Oracle DB Name>
as.db.username=<user name>
as.db.password=<password>
as.db.password-plain=true
```

Microsoft SQL Server

To use a Microsoft SQL Server database, you will need the following properties:

MS SQL Database Properties (Example)

```
as.db.vendor=sqlserver
as.db.url=jdbc:sqlserver://<Your MSSQL DB Host>:<Your Port
Name>;databaseName=<Your DB Name> or jdbc:sqlserver://<Your MSSQL DB
Host>\<Your Instance Name>;databaseName=<Your DB Name>
as.db.username=<user name>
as.db.password=<password>
as.db.password-plain=true
```

PostgreSQL

To use a PostgreSQL database, you will need the following properties:

PostgreSQL Database Properties (Example)

```
as.db.vendor=postgresql
as.db.url=jdbc:postgresql://<Your PostgreSQL DB Host>:5432/<Your DB name>
as.db.username=<user name>
as.db.password=<password>
as.db.password-plain=true
```

IBM Db2

To use an IBM Db2 database, you will need the following properties:

DB2 Database Properties (Example)

```
as.db.vendor=db2
as.db.url=jdbc:db2://<Your DB2 host>:50000/<Your DB name>
as.db.username=<user name>
as.db.password=<password>
as.db.password-plain=true
```

Web server settings

Web server settings allows you to configure the default server ports, as well as other networking related variables. Depending on your network setup, you can choose to use HTTP or HTTPS. With HTTP, the entry point must be accessible from the Internet, and so a load-balancer or reverse proxy may be required to handle HTTPS traffic as well as SSL termination.

By default, the `server port` serves all web page requests including authentication callback pages and admin UI interface requests.

Server port, protocol and SSL settings

```
## Server port, protocol and SSL settings
## protocol http|https|both
as.server.protocol=http
as.server.http.port=8080
as.server.https.port=8443
as.server.https.key-store=<Your keystore file path>
## keystore type, can be pkcs12 or jks
as.server.https.key-store-type=pkcs12
as.server.https.key-store-password=<Your keystore password>
## Set to false to disable this listening port
# as.server.enabled=false
```

- `as.server.https.key-store=`
  Keystore file path. A keystore is required for HTTPS. The keystore should contain server certificates for the specified HTTPS listening port. Note that for a production instance, the server certificate must be commercially signed by a CA.
- `as.server.https.key-store-type=`
  Keystore type. ActiveServer supports two different keystore types. Possible values are

  `pkcs12` or `jks`. Commercially signed certificates issued by a CA are typically in "pkcs12" format with a file extension of `.p12` or `.pfx`.

- `as.server.enabled=`
  Enable or disable the server listening port. To enable the server listening port, enter `true`. To disable server listening port, enter `false`.

> **Warning**
>
> HTTP is not recommended to access the web pages served by the server port. The HTTP setting for `as.server.protocol` should only be used for SSL termination in a production environment where a front-end load balancer handles TLS/SSL traffic and forwards the request to the back-end AS instance via HTTP.

Depending on your network configuration, you may want to setup administration access via a separate port. To do so, the following settings must be applied. By default, the Admin port number is disabled. If enabled, port numbers set below must not conflict with any other port numbers.

Enabling this setting will restrict all administration UI interface traffic to the specified port. The `server port` will then serve the authentication callback pages only.

Admin port (Example)

```
#Admin port , protocol and ssl config
#By default, Admin port configuration shares with Server port configuration
#Set following setting to true to enable this listening port
as.admin.enabled=false
as.admin.http.port=9090
as.admin.https.port=9443
#protocol http|https|both
as.admin.protocol=http
as.admin.https.key-store=<Your keystore file path>
#keystore type, can be pkcs12 or jks
as.admin.https.key-store-type=pkcs12
as.admin.https.key-store-password=<Your keystore password>
```

Warning

HTTP is not recommended to access your administration UI. The HTTP setting for `as.admin.protocol` should only be used for SSL termination in a production environment where a front-end load balancer handles TLS/SSL traffic and forwards the request to the back-end AS instance via HTTP.

Authentication and Admin API port. Only available in HTTPS with mutual authentication. This port will be enabled once the ActiveServer instance is activated. A server restart is required to enable this port.

Auth API port (Example)

```
#Auth api port, only https port is configurable
as.api.port=7443
```

The following Directory Server listening port settings are for ActiveServer to send and receive requests with the Directory Server in mutual authentication. These connectors are always HTTPS enabled. Server and client certificates for Directory Servers can be configured later on, as described in Manage DS certificates.

Note

Once you complete the certificate settings on the Administrator UI, a server restart is required.

Each Directory Server has the following set of configurable properties:

- `as.<Card Scheme>.port=`

  These are the port numbers for each card scheme for listening on their Directory Servers. Default values can be found below.

  Note

  The port number must not conflict with any other port numbers.

- `# as.<Card Scheme>.enabled=false`

This parameter is commented out by default. Determines the status of the Directory Server listening port, disabled or enabled.
To disable the Directory Server listening port, enter `false` , otherwise, leave it commented out.

American Express

American Express Directory Server Properties Example

```
as.amex.port=9600
## Set to false to disable DS HTTPS listening port
# as.amex.enabled=false
```

China UnionPay

China Union Pay Directory Server Properties Example

```
as.chinaunionpay.port=9601
## Set to false to disable DS HTTPS listening port
# as.chinaunionpay.enabled=false
```

Support for China UnionPay will be added in a future release.

Discover / Diners Club International

Discover / Diners Club International Directory Server Properties Example

```
as.discover.port=9602
## Set to false to disable DS HTTPS listening port
# as.discover.enabled=false
```

JCB

JCB Directory Server Properties Example

```
as.jcb.port=9603
## Set to false to disable DS HTTPS listening port
# as.jcb.enabled=false
```

Mastercard

Mastercard Directory Server Properties Example

```
as.mastercard.port=9604
## Set to false to disable DS HTTPS listening port
# as.mastercard.enabled=false
```

Visa

Visa Directory Server Properties Example

```
as.visa.port=9605
## Set to false to disable DS HTTPS listening port
# as.visa.enabled=false
```

Keystore settings

ActiveServer provides 3 options for storing encryption keys:

- Local keystore (SunJCE)
- Amazon S3 keystore
- PKCS11 HSM

Use the following property to set the keystore type:

```
as.keystore.type=<keystore type>
```

- `as.keystore.type=`
  Keystore type - possible values are `local` , `s3` , `pkcs11` , `kms` .

Local keystore (SunJCE)

To use a local keystore file, use the following property:

Local keystore (SunJCE) (Example)

```
as.keystore.local.path=${AS_HOME}/security/keystores/
```

- `as.keystore.local.path=`
  Keystore file path. Enter the file path to your keystore file, which should point to the folder that contains the keystore files.

Warning

If using a Windows based machine, note that an escape character (`\`) is probably required when setting the full path to the keystore folder.

E.g. if a Windows share folder is being used with the path `\\ActiveServer\keystores` , the keystore path would be set as `as.keystore.local.path=\\\\ActiveServer\\keystores` .

Amazon S3 keystore

ActiveServer supports using Amazon S3 as the keystore. To utilize an Amazon S3 keystore, you need to set the *AWS Bucket*, *AWS Region*, and *AWS Credentials* settings.

AWS BUCKET

1. Please create an empty S3 bucket by following the AWS guide.
2. Set your AWS Bucket path in the following properties:

Amazon S3 keystore (Example)

```
as.keystore.s3.bucket-name=<Your S3 Bucket Name>
```

If only the bucket name is provided e.g. `as.keystore.s3.bucket-name=as-example-bucket`, required keystores will be created by ActiveServer in the root directory. If a path is added to the bucket name, it will create the directory with given name inside the bucket. For example, `as.keystore.s3.bucket-name=as-test-bucket/keystores` will create all the required keystores under `keystores` directory inside the bucket `as-example-bucket`.

AWS REGION

The AWS Region can be set in several ways. A list of region codes can be found in the *Region* column of this table: Amazon AWS - Regions and Availability Zones.

1. Set the AWS Region code in the following properties:

Amazon S3 keystore (Example)

```
as.keystore.s3.region=<Your S3 Region Name>
```

2. Or, set the AWS Region in the AWS config file on your local system. The config file should be located at: `~/.aws/config` on Linux, macOS, or Unix, or `C:\Users\USERNAME\.aws\config` on Windows. This file should contain lines in the following format:

```
[default]
region = <Your S3 Region Name>
```

ACCESS PERMISSIONS

Make sure you have configured the credentials with **Read** and **Write** permission to the bucket. For more information about configuring the credentials refer below.

AWS Key Management Service (KMS)

CONFIGURATION

1. Create a **symmetric** Customer Managed Key (CMK) in KMS by following the guide provided by AWS.
2. Copy and paste the **key ARN** from the AWS KMS dashboard to the properties file:

   AWS KMS (Example)

   ```
   as.keystore.kms.key-arn=<Your AWS Key ARN> e.g. arn:aws:kms:us-
   east-1:123456789012:key/19c7b3dc-c49d-401f-bb97-f10bf3e116c9
   ```

ACCESS PERMISSIONS

Make sure you have configured the credentials with **Read** and **Write** permission to the bucket. For more information about configuring the credentials refer below.

Warning

Note that only **Auth API v2** is supported when using AWS KMS. **Auth API v1** transactions will be result in **error code 1027**.

PKCS11 HSM

ActiveServer supports using a HSM as the keystore. The HSM must support the PKCS11 API. To use hardware encryption with a PKCS11 HSM, you will need the following properties:

PKCS11 HSM (Example)

```
as.keystore.pkcs11.library=<the library to the pkcs11 driver>
as.keystore.pkcs11.slot=<the slot number>
```

- `as.keystore.pkcs11.library=`
  HSM driver library. For Linux, this is typically a `.so` file. For Windows, this is typically a `.dll` file. Consult your HSM's documentation for details on the library that should be use.
- `as.keystore.pkcs11.slot=`
  The slot number on your HSM. Consult your HSM's documentation for details on the slot number that should be use.

Info

Note that setting up and configuring an HSM is out of scope for this document. Please ensure your HSM is fully functional before configuring with ActiveServer.

HSM Token Login is required

When using a HSM for key management, AS requires the HSM to enable *"Always require Token Login"*. This setting is usually set as on automatically for most HSMs, however for HSMs like Safenet, this setting may not be on by default. Please refer to your HSM documentation for instructions.

For SafeNet HSMs, this can be done by setting the `No Public Crypto` security flag. Administrators can use the provided command line utility `ctconf` to set the flag.

AWS Credentials

If `as.keystore.type=s3` or `as.keystore.type=kms` is used, the AWS credentials needs to be configured so that ActiveServer can access the AWS services. AWS Credentials include an `access_key_id` and a `secret_access_key`. AWS Credentials can be set in a number of ways:

1. Set your AWS Credentials in the following properties if you want to use different credentials for the S3 bucket:

   Amazon S3 keystore (Example)

   ```
   as.keystore.s3.credentials.access-key-id=<Your Amazon S3 access key ID>
   as.keystore.s3.credentials.secret-access-key=<Your Amazon S3 secret access
   key>
   ```

   Warning

   This is no longer recommended but these properties are maintained for anyone using an ActiveServer version older than 1.3.0. It is strongly recommended to use the other credential options available.

2. Set your AWS credentials in the following properties:

AWS credentials (Example)

```
as.aws.credentials.access-key-id=<Your AWS access key ID>
as.aws.credentials.secret-access-key=<Your AWS secret access key>
```

3. Or, set the AWS Credentials in the AWS credentials profile file on your local system. The credentials profile file should be located at: `~/.aws/credentials` on Linux, macOS, or Unix, or `C:\Users\USERNAME\.aws\credentials` on Windows. If using this method, the **ActiveServer** properties file can be left blank. This file should contain lines in the following format:

```
[default]
aws_access_key_id = <Your Amazon S3 access key ID>
aws_secret_access_key = <Your Amazon S3 secret access key>
```

4. Or, if you deploy **ActiveServer** on an AWS EC2 instance, you can specify an IAM role and then give your EC2 instance access to that role. In this case, you need to follow the Amazon AWS - Using IAM Roles to Grant Access to AWS Resources on Amazon EC2 guide. If using this method, the **ActiveServer** properties file can be left blank.

Email server settings

ActiveSever allows you to send email notifications to users. Email notifications can be used to notify a user of their activation URL, remind users when a license is about to expire etc.

You will need an email account with its associated credentials and server details to setup email notifications.

Email Server Properties (Example)

```
as.mail.host=<Your SMTP server host>
as.mail.port=<Your SMTP server port>
as.mail.user-name=<Email address>
as.mail.password=<Email password>
as.mail.auth=true
as.mail.start-tls=true
```

- `as.mail.host=`
  The SMTP domain of your email server.
- `as.mail.port=`
  The port number of your email server.
- `as.mail.user-name=`
  The email address of the account from which emails will be sent from.
- `as.mail.password=`
  The password of the email account.
- `as.mail.auth=`
  Whether the email account requires SMTP authentication. If the email account requires authentication, enter `true` . Otherwise, enter `false` . If unsure, consult your email server administrator for details.
- `as.mail.start-tls=`
  TLS required by the SMTP server or not. If SMTP server requires TLS, enter `true` . Otherwise, enter `false` . If unsure, consult your email server administrator for details.

Info

Note that setting up and configuring email servers is out of scope for this document. Please ensure your email server is fully functional before configuring with ActiveServer.

Log settings

By default, **ActiveServer** will output all log files to the `{AS_HOME}/logs/` folder, which will be created if it does not exist. If you would like to specify a different log folder location, uncomment and edit the following setting with the desired path:

```
# as.logging.path=<Your log file path>
```

If using a multi node setup for the instance, separate folders are required for each node due to the file naming conventions.

Warning

If this value is changed, note that any errors in setup may result in the log files not being recorded correctly. Ensure that the path is correct and that the **ActiveServer** instance can access and has read/ write permissions for the specified path.

Setting TLS version

ActiveServer specifies HTTPS connectors to use TLS 1.2 only by default. However, due to a reported issue with Java 1.8, the TLS 1.0 and 1.1 protocols are also enabled. If you wish to disable protocols, a workaround is possible by editing the Java security file directly and is shown below.

Warning

Note that editing the `java.security` file will affect all Java applications on the server. GPayments cannot be held responsible for any issues that may result from this change.

To disable a specific protocol, edit the `java.security` file located at `<jdk directory>/jre/lib/security` and update the `jdk.tls.disabledAlgorithms` entry. The original entry might look like the below:

```
jdk.tls.disabledAlgorithms=SSLv3, RC4, DES, MD5withRSA, DH keySize < 1024, \
    EC keySize < 224, 3DES_EDE_CBC, anon, NULL
```

To disable protocols considered weak and not included by default, such as `SSLv2Hello, TLSv1, TLSv1.1`, append those values to the entry like below:

```
jdk.tls.disabledAlgorithms=SSLv3, RC4, DES, MD5withRSA, DH keySize < 1024, \
    EC keySize < 224, 3DES_EDE_CBC, anon, NULL,SSLv2Hello, TLSv1, TLSv1.1
```

Important

The suggested protocols to disable are just suggestions, consult your security team when deciding your security configuration for **ActiveServer**.

After editing the `java.security` file, restart any running **ActiveServer** instances for the change to take effect. You can check what protocols are enabled by running the following command on a linux terminal:

```
nmap --script +ssl-enum-ciphers -p 7443 127.0.0.1
```

Startup ActiveServer

Now that all properties are correctly configured, you can startup an instance of ActiveServer.

If you use Linux, open **Terminal**. If you use Windows, open **Command Prompt**.

Change your working directory to the folder that contains the startup scripts (`.sh` or `.bat` file).

Now you can startup ActiveServer with the following command.

Linux Windows

```
./startup.sh
```

Make sure the `as.jar` file is in the same directory as the startup scripts

The startup command will not work if the `as.jar` file is not in the same directory as the startup scripts.

You should now see the following output in **Terminal** or **Command Prompt**. Make sure to take note of the **Administration** URL, as it will be needed in the next step.

ActiveServer Instance Info

```
----------------------------------------------------------------------------------

ActiveServer by GPayments

_______        _____ _____              ________
___    |_________  /____(_)___   _______ __  ___/_____ ___________   _______
________
__  /| |_  ___/_  __/__  / __ | / /_  _ \_____ \ _  _ \__  ___/__ | / /_  _
\__  ___/
_  ___ |/ /__  / /_  _  /  __ |/ / /  __/____/ / /  __/_  /    __ |/ / /  __/
_  /
/_/  |_|\___/  \__/  /_/   _____/  \___/ /____/  \___/ /_/     _____/  \___/ /_/

----------------------------------------------------------------------------------

.
.
.

--------------------------------------------------------------------------------
        ActiveServer by GPayments is up and running.

        Version:                      1.0.0
        Git Commit Id:                da369ec

        Activation:                   NOT ACTIVATED, please contact GPayments
        Authentication API Port:      7443

        Server:                       http://10.0.75.1:8081
        Administration:               http://10.0.75.1:8081

        Key Store Type:               SUNJCE

        Profile(s):                   [prod]
-------------------------------------------------------------------------------
```

Startup script

In the startup scripts, the environment variable `AS_HOME` is set to the directory in which `application-prod.properties` is located. ActiveServer uses `AS_HOME` to find the configuration files as well as maintain keystores and output logs. By pointing `AS_HOME` to different locations, it is possible to run multiple ActiveServer instances on the same server. This can be done by copying the package to a different directory, or creating different startup scripts, and within those scripts, pointing `AS_HOME` to different directories.

> **Note**
>
> Where there are multiple instances on the same server, the port numbers must not conflict in any of the `application-prod.properties` files.

ActiveServer profile

In addition to setting `AS_HOME`, the startup script also sets the environment variable `AS_PROFILES`. This is a convenient mechanism to specify profile based configurations.

By default, the profile is set to `prod`.

ActiveServer uses the pattern `application-<profile>.properties` to load the profile's configuration file. Therefore under the default `prod` profile, `application-prod.properties` will be loaded. However, you can create new profiles (such as `test`) to setup different configurations for ActiveServer.

To create a new profile:

- Create a new configuration file named `application-test.properties` and place it in the same directory as the `prod` configuration file.
- Open the startup scripts and set the value of `AS_PROFILES` to `test`.

  ActiveServer will load the new profile instead of the old `prod` properties.

  ActiveServer can also load multiple profiles at the same time, to do this set the value of `AS_PROFILES` to `prod,test` and ActiveServer will load properties files from both `prod` and `test` profiles.

  With these options available, you can maintain settings separately under separate `.properties` files.

> **Tip**
>
> If you maintain all database settings in `application-db.properties`, and web server settings in `application-web.properties`, by setting the value of `AS_PROFILES` to `db,web`, ActiveServer settings can be segregated for different administrators to manage.

Setup Wizard

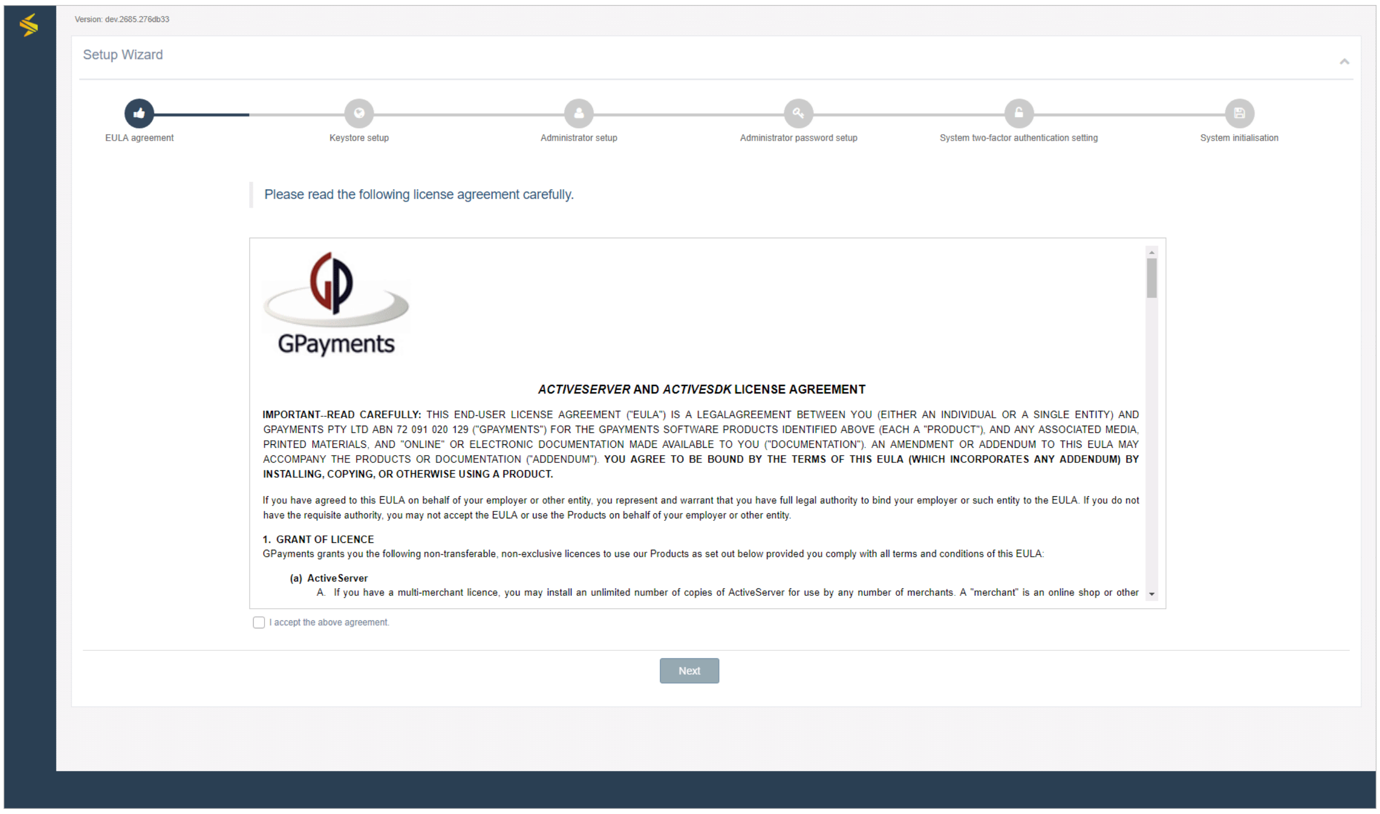
Once ActiveServer is up and running, you can access the Administrator UI via the **Administration** URL.

If this is your first time running ActiveServer, the Setup Wizard will appear and guide you through the setup process.

The Setup Wizard involves the following steps:

- EULA agreement
- Keystore setup
- Administrator setup
- Administrator password setup
- System two-factor authentication setting
- System initialisation

EULA agreement


Version: dev.2685.276db33
Setup Wizard
EULA agreement
Keystore setup
Administrator setup
Administrator password setup
System two-factor authentication setting
System initialisation
GPayments
I accept the above agreement.
Next

Please read the following license agreement carefully.

***ACTIVESERVER* AND *ACTIVESDK* LICENSE AGREEMENT**

IMPORTANT--READ CAREFULLY: THIS END-USER LICENSE AGREEMENT ("EULA") IS A LEGALAGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR A SINGLE ENTITY) AND GPAYMENTS PTY LTD ABN 72 091 020 129 ("GPAYMENTS") FOR THE GPAYMENTS SOFTWARE PRODUCTS IDENTIFIED ABOVE (EACH A "PRODUCT"), AND ANY ASSOCIATED MEDIA, PRINTED MATERIALS, AND "ONLINE" OR ELECTRONIC DOCUMENTATION MADE AVAILABLE TO YOU ("DOCUMENTATION"). AN AMENDMENT OR ADDENDUM TO THIS EULA MAY ACCOMPANY THE PRODUCTS OR DOCUMENTATION ("ADDENDUM"). **YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA (WHICH INCORPORATES ANY ADDENDUM) BY INSTALLING, COPYING, OR OTHERWISE USING A PRODUCT.**

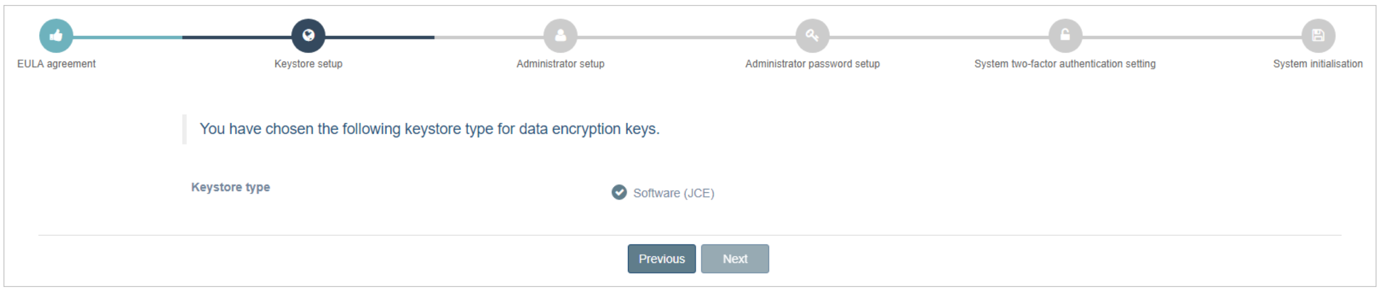
If you have agreed to this EULA on behalf of your employer or other entity, you represent and warrant that you have full legal authority to bind your employer or such entity to the EULA. If you do not have the requisite authority, you may not accept the EULA or use the Products on behalf of your employer or other entity.

1. GRANT OF LICENCE
GPayments grants you the following non-transferable, non-exclusive licences to use our Products as set out below provided you comply with all terms and conditions of this EULA:

(a) ActiveServer
A. If you have a multi-merchant licence, you may install an unlimited number of copies of ActiveServer for use by any number of merchants. A "merchant" is an online shop or other

Read the EULA Agreement. If you agree to the terms and conditions, select the **I accept the above agreement.** checkbox to proceed.

Keystore setup



You have chosen the following keystore type for data encryption keys.

Keystore type ✓ Software (JCE)

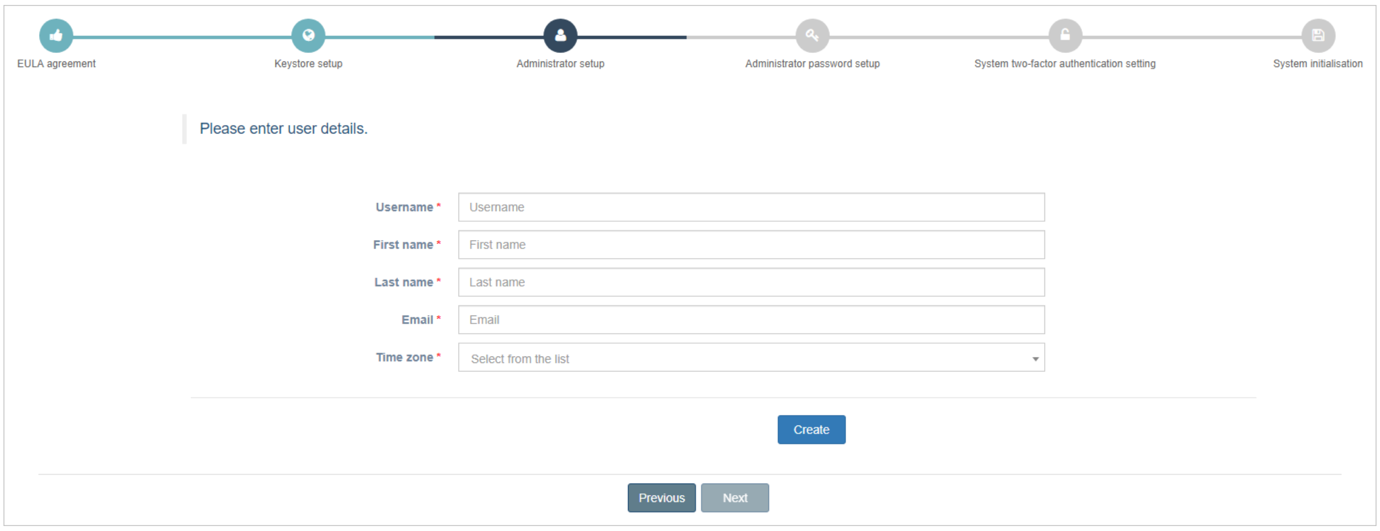
- Select the **Keystore type**.

  If **Software** was selected by during setup, SUNJCE will be utilised.

  There is also the option to use a PKCS#11 HSM by entering the appropriate details in the `application-prod.properties` file. See the Encryption Module on how to setup a PKCS#11 HSM.

- Select the ***Next*** button to continue.

Administrator setup

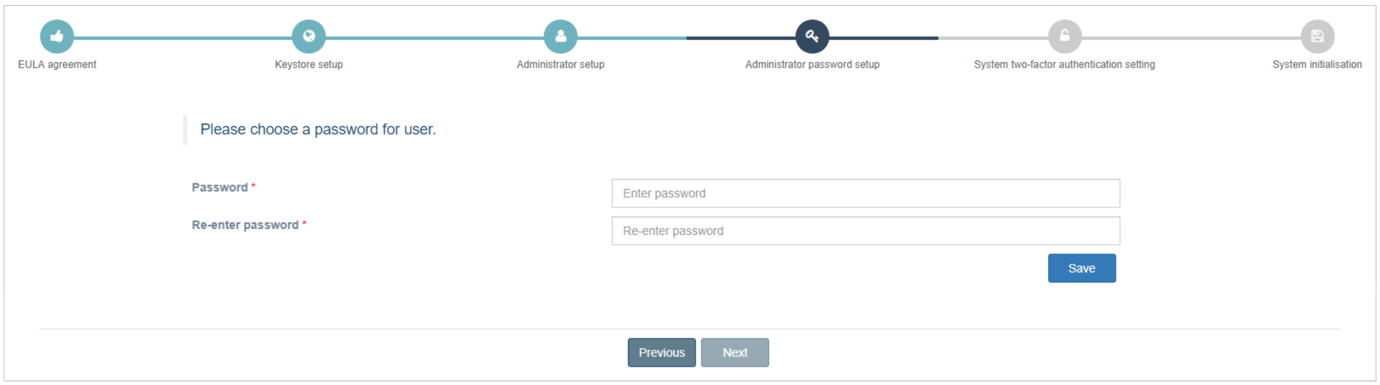


Please enter user details.

Username *	Username
First name *	First name
Last name *	Last name
Email *	Email
Time zone *	Select from the list

- Enter the user details for your Administrator account.
- Select the ***Create*** button to create the account.
- Select the ***Next*** button to continue.

Administrator password setup



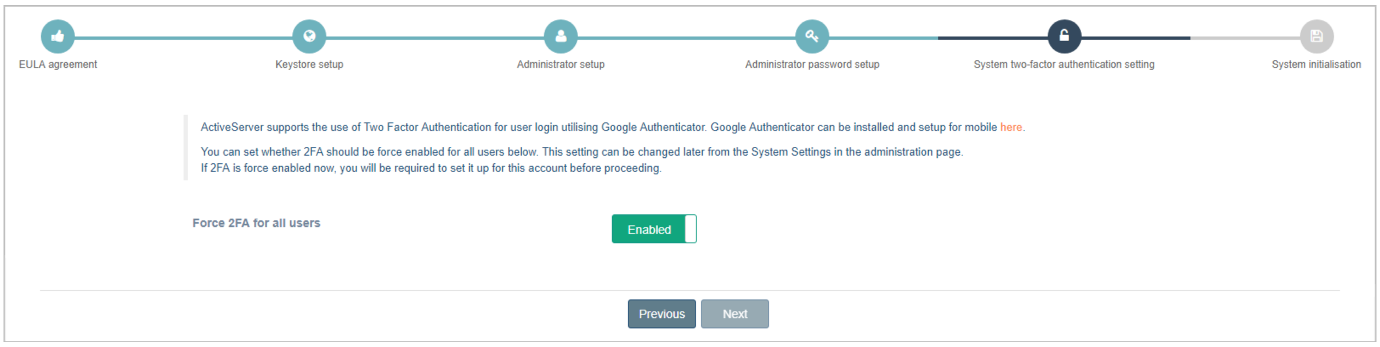
Please choose a password for user.

Password *: Enter password

Re-enter password *: Re-enter password

- Enter a password for the Administrator account.
- Re-enter the password to confirm it.
- Select the ***Save*** button to create it.
- Select the ***Next*** button to continue.

System two-factor authentication setting



ActiveServer supports the use of Two Factor Authentication for user login utilising Google Authenticator. Google Authenticator can be installed and setup for mobile here.

You can set whether 2FA should be force enabled for all users below. This setting can be changed later from the System Settings in the administration page.
If 2FA is force enabled now, you will be required to set it up for this account before proceeding.

Force 2FA for all users: Enabled

ActiveServer supports two factor authentication for signing into the Administrator UI.

Note

To use this feature, you must have Google Authenticator installed on a mobile device.

Refer to Install Google Authenticator for setup instructions for Google Authenticator.

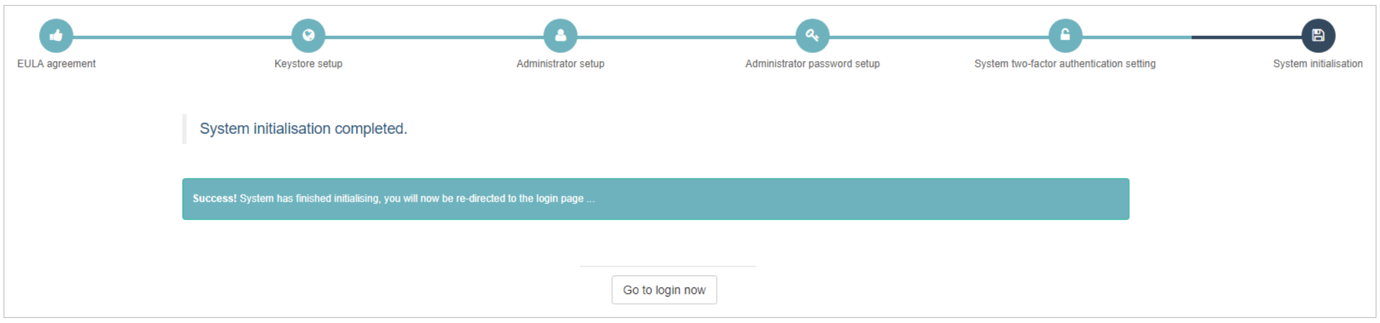
By default, ActiveServer does not force users to use two factor authentication.

To mandate two factor authentication for all users:

- ***Enable*** the toggle, adjacent to **Force 2FA for all users**.

- Select the ***Next*** button to continue.

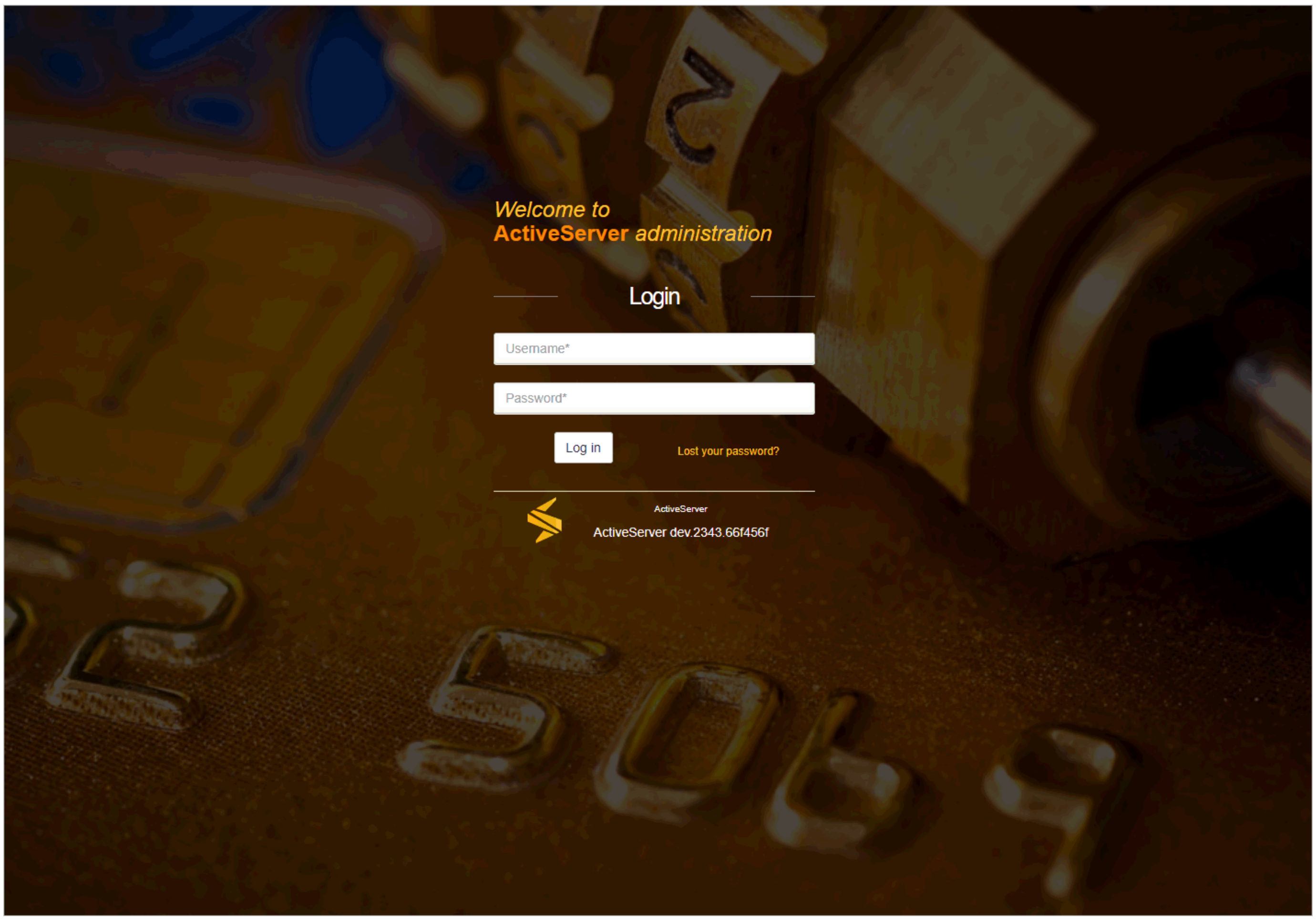
System initialisation


EULA agreement
Keystore setup
Administrator setup
Administrator password setup
System two-factor authentication setting
System initialisation

System initialisation completed.

Success! System has finished initialising, you will now be re-directed to the login page ...

The Setup Wizard will inform you that system initialisation is complete and you will be redirected to the ActiveServer login page.



What next?

After this, you can:

- Activate ActiveServer
- Configure system settings
- Start integrating with the ActiveServer API

Features overview

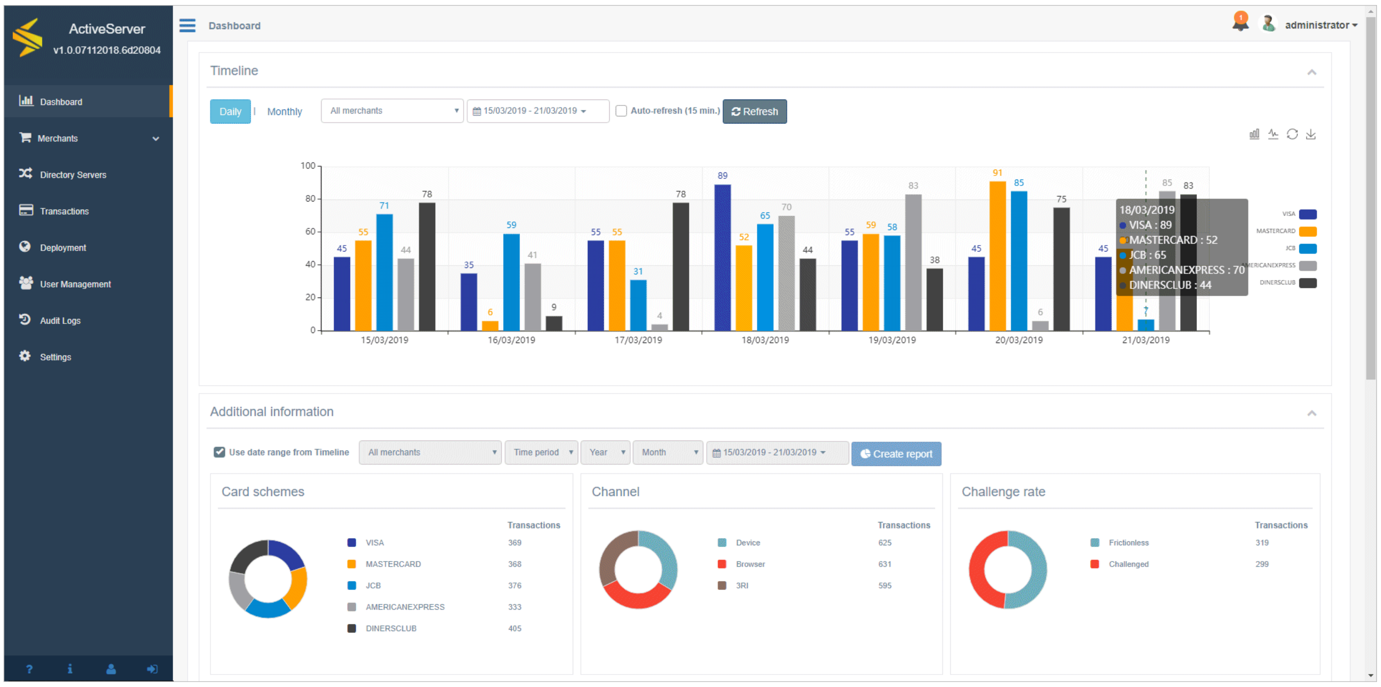
Listed below is the overview for all **ActiveServer** functionality, including user interface outline. Under each section is a link to the relevant guides that will assist you with performing tasks.

Dashboard

The **Dashboard** displays statistical graphics of authentications. These statistics are available for customisable time periods and can be system wide or broken down by merchant.

The Dashboard is only visible to users with roles that are designed for managing merchants, e.g **Business Admin**, **Merchant Admin** and **Merchant**.

For further information, refer to *Guides* > *Administration UI* > Using the Dashboard.



Card schemes

		Transactions
🟦	VISA	369
🟧	MASTERCARD	368
🟦	JCB	376
⬜	AMERICANEXPRESS	333
⬛	DINERSCLUB	405

Channel

		Transactions
🟦	Device	625
🟥	Browser	631
🟫	3RI	595

Challenge rate

		Transactions
🟦	Frictionless	319
🟥	Challenged	299

Merchants

Merchants has two sections: **Merchants** and **Acquirers**.

The **Merchants** page is used to manage merchant entities in **ActiveServer**. Merchants can be *created*, *searched*, *viewed*, *edited* and *deleted*. This is also where the 3DS Requestor **client**

certificate is downloaded from, as well as where the encryption key for the merchant can be rotated.

The **Acquirers** page is used to manage acquirer entities in **ActiveServer**. Acquirers can be *created*, *searched*, *viewed*, *edited* and *deleted*, before being assigned to merchant profiles for 3DS2 authentication requests.

Merchants pages are only visible to users with roles that are designed for managing merchants, e.g **Business Admin**, **Merchant Admin** and **Merchant** roles.

For further information, refer to *Guides > Administration UI*:

- Search merchants
- Manage merchants
- Manage acquirers.

Directory Servers

The **Directory Servers** page is used to manage the various card brand Directory Server settings in **ActiveServer**.
It has tabs for each of the card schemes. You can enter card scheme specific connection details, adjust timeout settings and manage SSL connections via the certificates section.

For further information, refer to *Guides > Administration UI*:

- Manage DS settings
- Manage DS certificates.

Transactions

The **Transactions** page is used to access records of all transactions processed by **ActiveServer**. Transactions can be filtered by various fields and accessed to view both transaction details and 3DS messaging.

This menu item and page is only visible to users who have roles that are designed for managing merchants, e.g **Business Admin**, **Merchant Admin** and **Merchant** roles.

For further information, refer to *Guides > Administration UI* > View transactions.

Deployment

The **Deployment** page is where the online **Node**, or **Nodes** for a multi-node setup, for the instance can be managed. It is also where the instance **Activation status** of the instance is viewed or where the instance is activated initially.

This menu item and page is only visible to users who have roles that are designed for managing system architecture , e.g **System Admin** role.

For further information, refer to:

- *Guides* > *Administration UI* > Manage nodes
- *Guides* > Activate instance.

User Management

The **User Management** page is where user access to the administration interface can granted and managed. Users can be *created*, *searched*, *viewed*, *edited* and *deleted*. In particular, this is where users are granted **roles** to access various system functionality.

This menu item and page is only visible to users who have a role that is designed for managing system wide users, e.g **User Admin**.

For further information, refer to *Guides* > *Administration UI*:

- Manage users
- Roles and permissions.

Audit logs

The **Audit logs** page is where system events and changes are recorded to be viewed.

This menu item and page is only visible to users who have roles that are designed for managing system architecture , e.g **System Admin** role.

For further information, refer to *Guides* > *Administration UI* > Audit logs.

Settings

The **Settings** page is where the user can configure **system**, **security** and **3D Secure 2** related settings for the instance.

This menu item and page is only visible to users who have roles that are designed for managing system architecture , e.g **System Admin** role.

For further information, refer to *Guides* > *Administration UI* > Configure system settings.

About

The **About** page is where the technical specifications of the instance is displayed. This is a useful resource for users when raising technical support queries with the GPayments support team.

This menu item and page is only visible to users who have roles that are designed for managing system architecture , e.g **System Admin** role.

For further information, refer to *Guides* > *Administration UI* > View ActiveServer information.

Notifications

The **Notifications** section is where important system notifications are communicated to the user. The notifications are displayed in the top right hand corner of the administration interface under the 🔔 icon.

For further information, refer to *Guides* > *Administration UI* > Notifications.

User profile

The **User profile** page is where the current user can edit details relating to their account as well as change their password. It can be accessed by selecting **Profile** icon in the bottom left hand corner of the administration interface.

For further information, refer to *Guides* > *Administration UI* > User profile.

Logging

ActiveServer creates daily log files and stores them in the `as_home/logs` directory. The log file names have a ***"as.yyyy-mm-dd.log"*** format (e.g. the log file created on the 23rd November 2019 would be named ***as.2019-11-23.log***).

The log file contains the same messages, warnings and errors as shown on the **ActiveServer** console window.

If you are running **ActiveServer** in debug mode, the log files will contain detailed information about the transactions and can get very large. Always make sure that you have enough disk space for logging purposes. It is recommended that you remove (or archive) old log files every three months.

The verbosity of the log files can be set in the system settings. For further information, refer to *Guides > Administration UI >* Configure system settings.

Activate instance

⚠ **Activation is required**

Any new **ActiveServer** instance needs to be activated before it can process authentication requests.

To activate the ActiveServer instance:

1. Purchase a license from GPayments

You will need to purchase a license from GPayments to access the MyAccount features for activating your instance. For further details, please contact us at sales@gpayments.com.

2. Setup your instance

Follow the Quickstart Guide and ensure your ActiveServer instance is set up and you can access the administration interface.

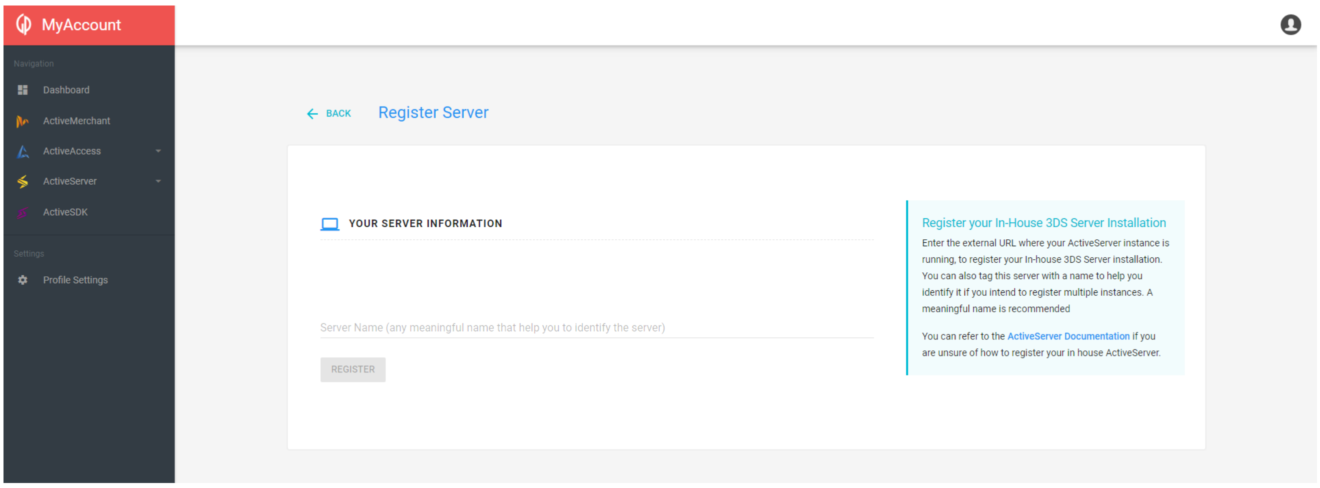
3. Configure the External URL and Auth API URL

1. On the administration interface, navigate to **Settings > System** and enter the **External URL** and **Auth API URL** values
    - **External URL** - publicly accessible URL in which your ActiveServer instance is running and you have configured to listen on the `as.server.https.port`. Note that depending on your load-balancing setup your **External URL** may not not have the port number included e.g. `https://admin.myserverinstance.com`.
    - **API URL** - URL used to receive authentication and administration API calls. The domain name of this URL will also be used to generate client certificates for the authentication of APIs (x.509). If it is not provided by default **ActiveServer** will use the domain name in the External URL for client certificate generation. Note this URL does not have to be publicly accessible. The form of the URL is the same as the **External URL**, with the port number being the API port.

2. Select the ***Save*** button.

4. Register server and choose an Activation Method

1. Login to MyAccount. If you have purchased a license from GPayments, you should already have access to the ActiveServer section.
2. Select **ActiveServer > My Instances** on the side menu.
3. Select ***ADD NEW SERVER***. You should see a screen similar to the one below, which displays the input field for the **Server Name**.



Register your In-House 3DS Server Installation

Enter the external URL where your ActiveServer instance is running, to register your In-house 3DS Server installation. You can also tag this server with a name to help you identify it if you intend to register multiple instances. A meaningful name is recommended

You can refer to the ActiveServer Documentation if you are unsure of how to register your in house ActiveServer.

4. Select ***REGISTER***. You should see the server information displayed that was just entered, along with the **Activation State**. If you made a mistake and would like to remove this instance, select ***REMOVE***.
5. Select ***ACTIVATE 3DS SERVER***. You will be asked to choose one of the activation methods below:

OPTION 1: Activation using session

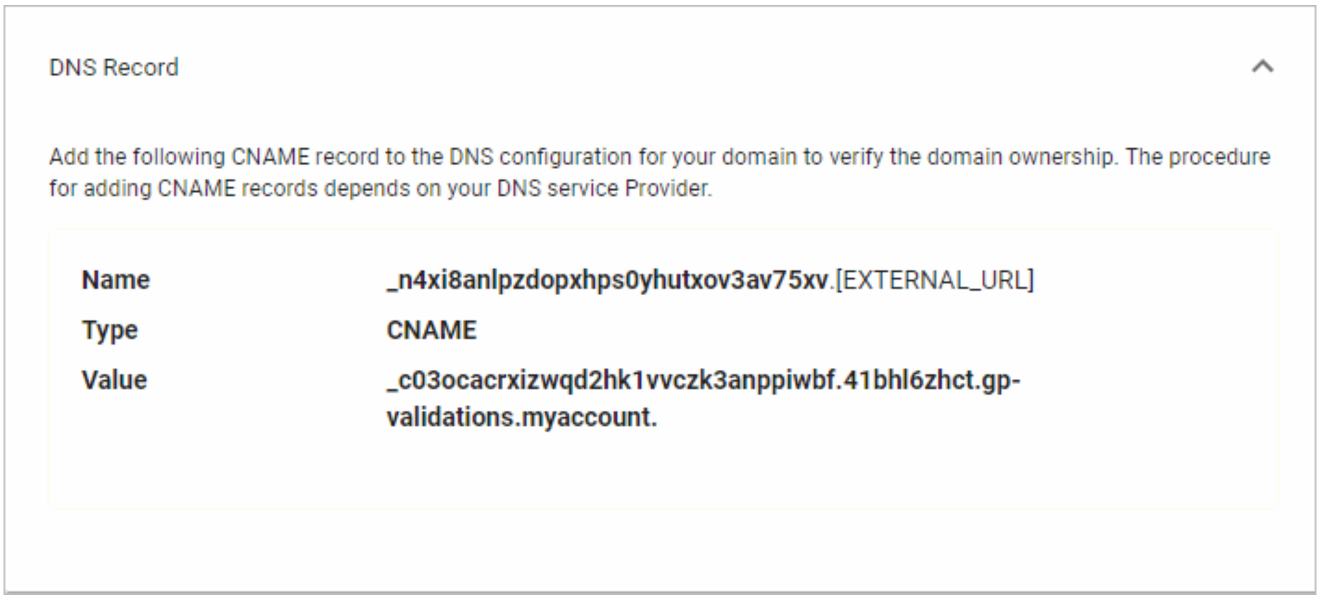
If you choose this method, make sure the **External URL** you specified in the previous step is publicly accessible.

The licensing server will make a request to this **External URL** to verify that your instance is running on the **External URL** you have specified and activate the instance.

OPTION 2: Activation using DNS

This activation process activates your ActiveServer instance by verifying the `CNAME` record generated by GPayments' licensing server.

You should see a DNS record similar to the one below:


DNS Record

Add the following CNAME record to the DNS configuration for your domain to verify the domain ownership. The procedure for adding CNAME records depends on your DNS service Provider.

Name	**_n4xi8anlpzdopxhps0yhutxov3av75xv**.[EXTERNAL_URL]
Type	**CNAME**
Value	**_c03ocacrxizwqd2hk1vvczk3anppiwbf.41bhl6zhct.gp-validations.myaccount.**

To create a DNS record:

a. Go to your domain's DNS records.

b. Add a record to your DNS settings, selecting CNAME as the record type.

c. Copy the value of **Name**, which in the above screenshot is `_n4xi8anlpzdopxhps0yhutxov3av75xv` , and paste it to **Label/Host/Name** in the DNS record depending on your domain host.

d. Copy the value of **Value**, which in the above screenshot is `_c03ocacrxizwqd2hk1vvczk3anppiwbf.41bhl6zhct.gp-validations.myaccount.` , and paste it to **Destination/Target/Value** depending on your domain host.

e. Save your record. The **CNAME** record changes can take up to 72 hours to take effect, but typically they happen much sooner.

Note

Your domain host is typically where you purchased your domain name (e.g. AWS Route 53, GoDaddy®, Enom®, or Name.com).

6. Select the data elements to be sent to the licensing server by either choosing to send all data elements, or customise the data elements sent:

   Transaction data (core): Information that is required for billing purposes, mandatory (or conditional) to send.

ID	Name	Mandatory	Group	Comments
ADE001	Directory Server Type	Y	Core	Used to track if the authentication request was sent to a Production or GPayments TestLabs' directory server.
ADE002	3DS Server Transaction Id	Y	Core	ID assigned by the 3DS Server to a transaction, used for cross referencing a transaction if a billing dispute arises.
ADE003	SDK Transaction Id	C	Core	Conditional: Only assigned for SDK transactions, must be provided if a value is present, used for cross referencing a transaction if a billing dispute arises.
ADE004	ACS Transaction Id	Y	Core	ID assigned by the ACS to a transaction, used for cross referencing if a billing dispute arises.
ADE005	Transaction Status	Y	Core	The transaction status, can be “Y” or “A” or “N”, etc. This is used to determine the final transaction status for billing purposes (i.e. error occurred during transaction).
ADE006	Transaction Status Reason	C	Core	Conditional: Reason for transaction failing, assists with identifying the exact reason for failure for billing purposes, must be provided if a value is present (i.e. transaction has failed).
ADE007	Transaction Start Time	Y	Core	Transaction start time, required when determining the billing cycle.
ADE008	Transaction End Time	C	Core	Conditional: Transaction end time, could be null if the transaction failed or terminated earlier, required if available.

Transaction data (extended): Information that is optional, unless conditionally required for billing purposes. Opting in to this information will allow GPayments to share anonymous industry insights with participating clients.

ID	Name	Mandatory	Group	Comments
ADE009	Payment Network	N	Extended	Payment network used for the transaction, e.g. American Express, China UnionPay, Discover, JCB, Mastercard, Visa, etc. Optional for clients to provide, unless billing structure requires this information.
ADE010	Device Channel	N	Extended	Device used for the transaction, e.g. BRW, APP, 3RI. Optional for clients to provide, unless billing structure requires this information.
ADE011	Authentication Type	N	Extended	Authentication type used for the transaction e.g. NPA (Non-payment) or PA (Payment). Optional for clients to provide, unless billing structure requires this information.
ADE012	Merchant Id	C	Extended	The internal Merchant ID (not acquirer assigned ID). Conditional for clients to provide if billing structure requires this information, used for Licensing Server to determine the size of the payment gateway (By calculating distinct merchant IDs).
ADE013	Merchant Acquirer Id Index	C	Extended	The index number of the Acquirer Merchant ID of the Merchant. Conditional for clients to provide if billing structure requires this information, used for Licensing Server to determine the size of the payment gateway (By calculating distinct merchant IDs).

Tech support data (core): Information used by GPayments for troubleshooting and planning purposes, required to send unless conditionally not available on instance server.

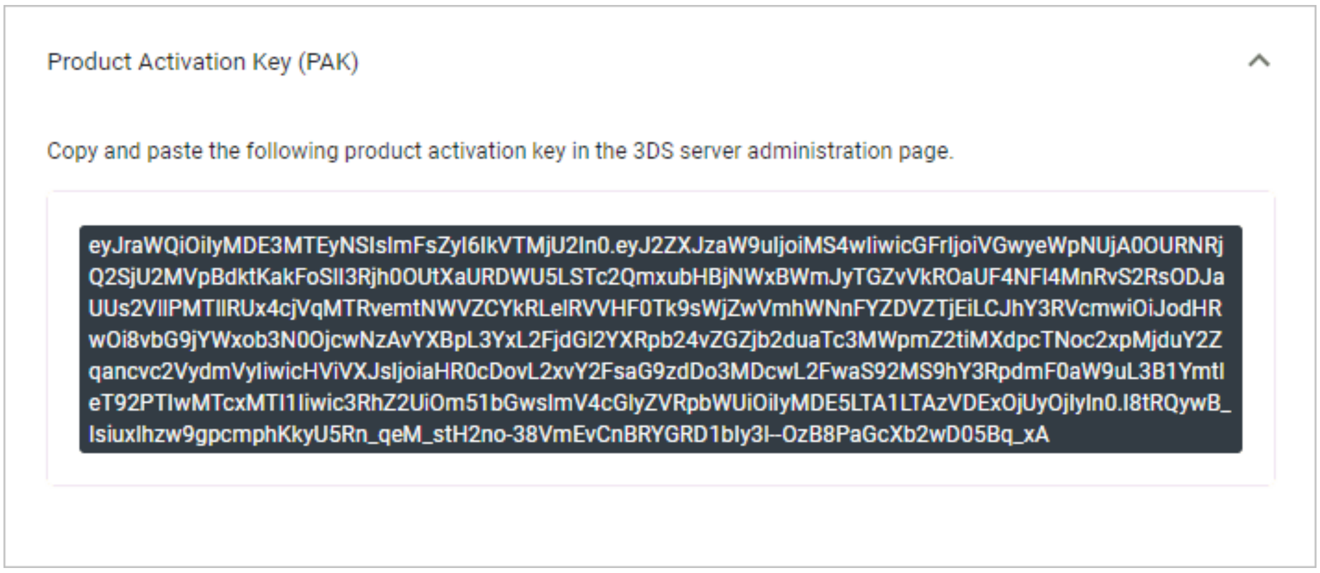
ID	Name	Mandatory	Group	Comments
AD001	ActiveServer Version	Y	Core	Version of ActiveServer, e.g. v1.0
AD002	OS Name	C	Core	Name of the OS, e.g. Ubuntu
AD003	OS Version	C	Core	Version of the OS, e.g. 16.04.5 LTS
AD004	Database Name	C	Core	Name of the database e.g. MySQL
AD005	Database Version	C	Core	Version of the database e.g 5.7
AD006	Java Edition and Version	C	Core	Edition of version of Java used e.g. OpenJDK 1.8.120
AD007	Node Count	C	Core	Number of nodes for the instance e.g. 2

7. Review the information provided to activate the instance and select ***BACK*** if any changes are required, otherwise select ***FINISH***.

5. Activate

You should see a product activation key (PAK), similar to the one below.

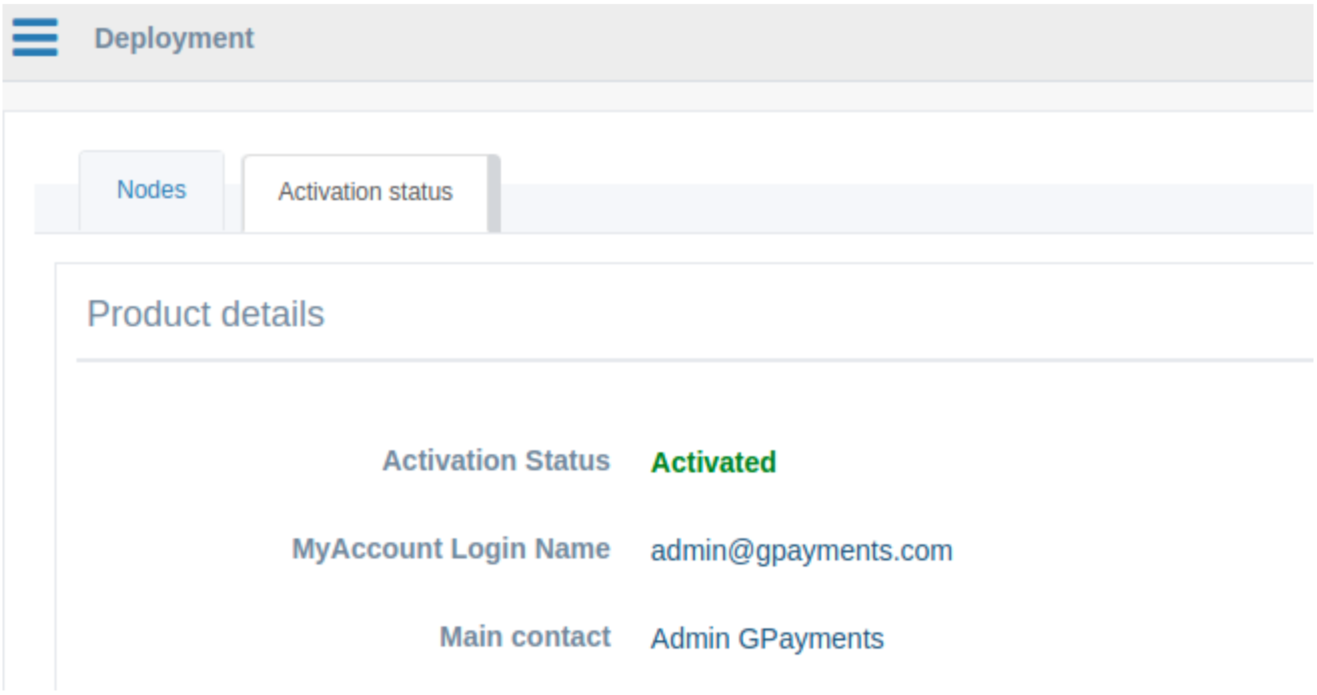
1. Copy this value to your clipboard to use shortly.



Copy and paste the following product activation key in the 3DS server administration page.

eyJraWQiOilyMDE3MTEyNSIslmFsZyl6IkVTMjU2In0.eyJ2ZXJzaW9uljoiMS4wliwicGFrljoiVGwyeWpNUjA0OURNRj
Q2SjU2MVpBdktKakFoSlI3Rjh0OUtXaURDWU5LSTc2QmxubHBjNWxBWmJyTGZvVkROaUF4NFI4MnRvS2RsODJa
UUs2VllPMTIlRUx4cjVqMTRvemtNWVZCYkRLelRVVHF0Tk9sWjZwVmhWNnFYZDVZTjEiLCJhY3RVcmwiOiJodHR
wOi8vbG9jYWxob3N0OjcwNzAvYXBpL3YxL2FjdGl2YXRpb24vZGZjb2duaTc3MWpmZ2tiMXdpcTNoc2xpMjduY2Z
qancvc2VydmVyliwicHViVXJsIjoiaHR0cDovL2xvY2FsaG9zdDo3MDcwL2FwaS92MS9hY3RpdmF0aW9uL3B1YmtI
eT92PTIwMTcxMTI1liwic3RhZ2UiOm51bGwslmV4cGlyZVRpbWUiOilyMDE5LTA1LTAzVDExOjUyOjIyIn0.I8tRQywB_
IsiuxIhzw9gpcmphKkyU5Rn_qeM_stH2no-38VmEvCnBRYGRD1bIy3I--OzB8PaGcXb2wD05Bq_xA

2. Go back to your ActiveServer dashboard, navigate to **Deployment > Activation Status** to fill in the details from MyAccount:
   - **MyAccount Login Name:** Email address registered for the account activating the instance.
   - **PAK:** Product activation key, which you have copied to your clipboard.
3. Select the ***ACTIVATE*** button. The **Activation Status** will change to *Waiting to restart* if successful.
4. **Restart** your instance for the changes to take effect and the activation process to complete. The screenshot below shows an example **Activation status** in section `Deployment` -> `Activation status` on Administration UI after restart.



Product details

Activation Status	Activated
MyAccount Login Name	admin@gpayments.com
Main contact	Admin GPayments

Success

Congratulations! You've successfully activated your **ActiveServer** instance.

Upgrade instance

Upgrade

Upgrading **ActiveServer** is a simple process that only requires the `as.jar` to be replaced with the new version.

To ***upgrade*** your existing **ActiveServer** instance to the latest version:

1. Stop the **ActiveServer** instance node.
2. Open the **ActiveServer** directory and back up the old `as.jar` file (simply copy or save in an archive) in case a roll back is required.
3. Back up the **ActiveServer** database (refer to your database documentation for back up processes and requirements specific to your database).
4. Download and extract the new **ActiveServer** package into a temporary directory and copy the `as.jar` file to your **ActiveServer** directory.
5. Start up the **ActiveServer** instance node. **ActiveServer** will automatically upgrade the database as necessary during the startup. Then the upgrade process is complete.

Clustering environment upgrade

In a clustering environment where multiple ActiveServer nodes are deployed with the same database, the database upgrade process will be handled automatically so that only one node at a time can migrate the database. The startup process of the rest of the nodes in the cluster will be blocked until the database migration is done by the first node.

Rollback

In case you need to ***roll back*** to your previous version of **ActiveServer**:

1. Stop the **ActiveServer** instance node.
2. Open the **ActiveServer** directory and back up the old `as.jar` file (simply copy or save in an archive) if required.
3. Restore your previous `as.jar` into the **ActiveServer** directory.

4. Restore the **ActiveServer** database that you have previously backed up.
5. Start **ActiveServer**.

TestLabs

The **GPayments TestLabs** consists of a live **Directory Server** and **Access Control Server**. It has different cardholder scenarios setup for clients to perform functional testing with their **ActiveServer** instance. All card schemes supported by **ActiveServer** are supported by **TestLabs**.

Integrating with ActiveServer

For information on integrating with **ActiveServer** to make a test transaction, you can refer to the integration guides.

TestLabs setup

For all TestLabs transactions, the default merchant (**Test Merchant**, with Merchant ID **123456789012345**) must be used. This includes using the client certificate available in it's merchant profile.

The following fields must also be used when performing a transaction:

- **Card holder name** - value must either be `Test Card` or an empty value
- **Expiry date (YYMM)** - value must either be `2508` or an empty value

Warning

Not setting the above values correctly will cause the transaction to fail.

TestLabs scenarios

The below scenarios can be initiated by using the listed card numbers when performing a transaction with the **GPayments TestLabs**:

Authentication success - frictionless

- **Description** - Transaction will complete without any challenge from the ACS.

- **ARes result:**
  - Transaction Status = Y
  - ECI = 05 or 02 (Mastercard)
  - Authentication Value is present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000000100	5100000000000107	3528000000000106	340000000000108	6440000

Authentication success - challenge

- **Description** - Transaction will step up to a challenge using a password. Enter the password "123456" and submit to complete the transaction.
- **ARes result:**
  - Transaction Status = C
- **RReq result:**
  - Transaction Status = Y
  - ECI = 05 or 02 (Mastercard)
  - Authentication Value is present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000005000	5100000000005007	3528000000005006	340000000005008	6440000

Authentication attempt

- **Description** - Transaction will attempt to perform authentication, before returning an attempts response.
- **ARes result:**
  - Transaction Status = A
  - ECI = 06 or 01 (Mastercard)

- Authentication Value is present

- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000100009	5100000000100006	3528000000100005	340000000100007	64400000

Card not enrolled

- **Description** - Transaction is performed using a card that is not enrolled in 3DS2, that will return a not authenticated response.
- **ARes result:**
  - Transaction Status = N
  - Transaction Status Reason = 08 (No card record)
  - ECI is not present
  - Authentication Value is not present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000200007	5100000000200004	3528000000200003	340000000200005	64400000

Authentication failed

- **Description** - Transaction will step up to a challenge using a password. Enter the password "111111" and submit to simulate the cardholder not being authenticated.
- **ARes result:**
  - Transaction Status = C
- **RReq result:**
  - Transaction Status = N
  - ECI = 00
  - Authentication Value is not present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000300005	5100000000300002	3528000000300001	340000000300003	64400000

Authentication unavailable

- **Description** - Transaction will end with authentication unavailable, due to a simulated technical error with the ACS.
- **ARes result:**
  - Transaction Status = U
  - ECI is not present
  - Authentication Value is not present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000400003	5100000000400000	3528000000400009	340000000400001	64400000

Authentication rejected

- **Description** - Transaction will end with the authentication being rejected by the ACS.
- **ARes result:**
  - Transaction Status = R
  - ECI is not present
  - Authentication Value is not present
- **Card numbers:**

Visa	Mastercard	JCB	American Express	Discover
4100000000500000	5100000000500007	3528000000500006	340000000500008	64400000

Roles and permissions

Roles and permissions are used to give users the correct access to various system functionality relating to their business roles. A user may have multiple roles. ActiveServer comes with pre-defined user roles:

- **System admin** - for managing the technical upkeep for the instance, including deployment and licensing, directory server connection management, system settings management plus monitoring system notifications.
    - **Page access:** *Directory servers, Deployment, Audit logs, Settings, About, Profile, System notifications.*
- **User admin** - for managing users for the instance, including assigning roles. This role is able to see all merchants in the system to enable it to assign a merchant to a single scope user. There must always be one user with this role.
    - **Page access:** *Merchants, User Management*.
- **Business admin** - for managing the business processes for **all** merchants on the instance, including viewing dashboard statistics, managing merchant functions and viewing transaction history.
    - **Page access:** *Dashboard, Merchants, Transactions, Profile*.
- **Merchant admin** - for managing the business processes for a **single** merchant on the instance, including viewing dashboard statistics, managing merchant details and viewing transaction history.
    - **Page access:** *Dashboard, Merchants, Transactions, Profile.*
- **Merchant** - for users who require read only access to a **single** merchant on the instance, including viewing dashboard statistics, viewing merchant details and viewing transaction history.
    - **Page access:** *Dashboard, Merchants, Transactions, Profile*.

Permission scope

Each user role has a level of scope attached to allow a **User admin** user to define the correct level of access to entities within the system.

Merchant scope

In relation to **Merchants**, scope indicates whether the user will be able to access **all** merchants and their information (e.g. statistics, details, transactions), or just allow access to a **single** merchant's information:

- ***All* scope** - The ***Business admin*** role has authority over **all** merchants. This allows them to select **all merchants** when viewing dashboard statistics, search/edit/create/delete all merchants and view transactions for all merchants in the system. The ***User admin*** has access to viewing merchant details for the purposes of assigning merchants to single scope users.
- ***Single* scope** - The ***Merchant admin*** and ***Merchant*** roles have authority over a **single** merchant only. After a merchant is assigned to their profile, they can access **only** that merchant's dashboard statistics, merchant details and transactions.
- ***No* scope** - The ***System admin*** role does not have any permissions relating to managing merchants, and therefore is not able to access any pages with merchant functionality.

This separation of duties allows clients managing multiple merchants in a single system, such as Payment Service Providers, to give granular control to individual merchants if required.

Important

If a user is assigned roles that have both **All** and **Single** scope, the **All** scope will take precedence.

Assigning merchants

If a user has **Single** scope level access in relation to merchants, a ***User admin*** can assign them an already created merchant to manage.

If the user already has a merchant assigned to them, this can be overwritten in their profile but they cannot have more than merchant at a time.

Permission list table

The following table provides a detailed view of the specific permissions granted to the user roles. The **Scope** column indicates permissions that have a scope attached to it where appropriate.

User Note

Through this document you will see these **User Note** boxes, which indicate what features are available to specific user roles.

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
Dashboard		View all merchant statistics	All merchants			✔		
		View merchant statistics	Single merchant				✔	✔
Merchants	Search	View all merchant details	All merchants		✔	✔		
		View merchant details	Single merchant				✔	✔

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
		Create multiple merchants	All merchants			✔		
		Delete multiple merchants	All merchants			✔		
	Merchant Settings	View all merchant details	All merchants		✔	✔		
		View merchant details	Single merchant				✔	✔
		Edit all merchant details	All merchants			✔		
		Edit merchant details	Single merchant				✔	
		View all merchant notes	All merchants			✔		
		Edit all merchant notes	All merchants			✔		
		Edit all merchant enabled status	All merchants			✔		

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
		Download all merchant certificates	All merchants			✔		
		Download merchant certificate	Single merchant				✔	✔
		Revoke all merchant certificates	All merchants			✔		
		Revoke merchant certificates	Single merchant				✔	
		Rotate all merchants encryption key	All merchants			✔		
		Rotate merchant encryption key	Single merchant				✔	
	Acquirer	View acquirers				✔		
		Create acquirer				✔		
		Edit acquirer				✔		

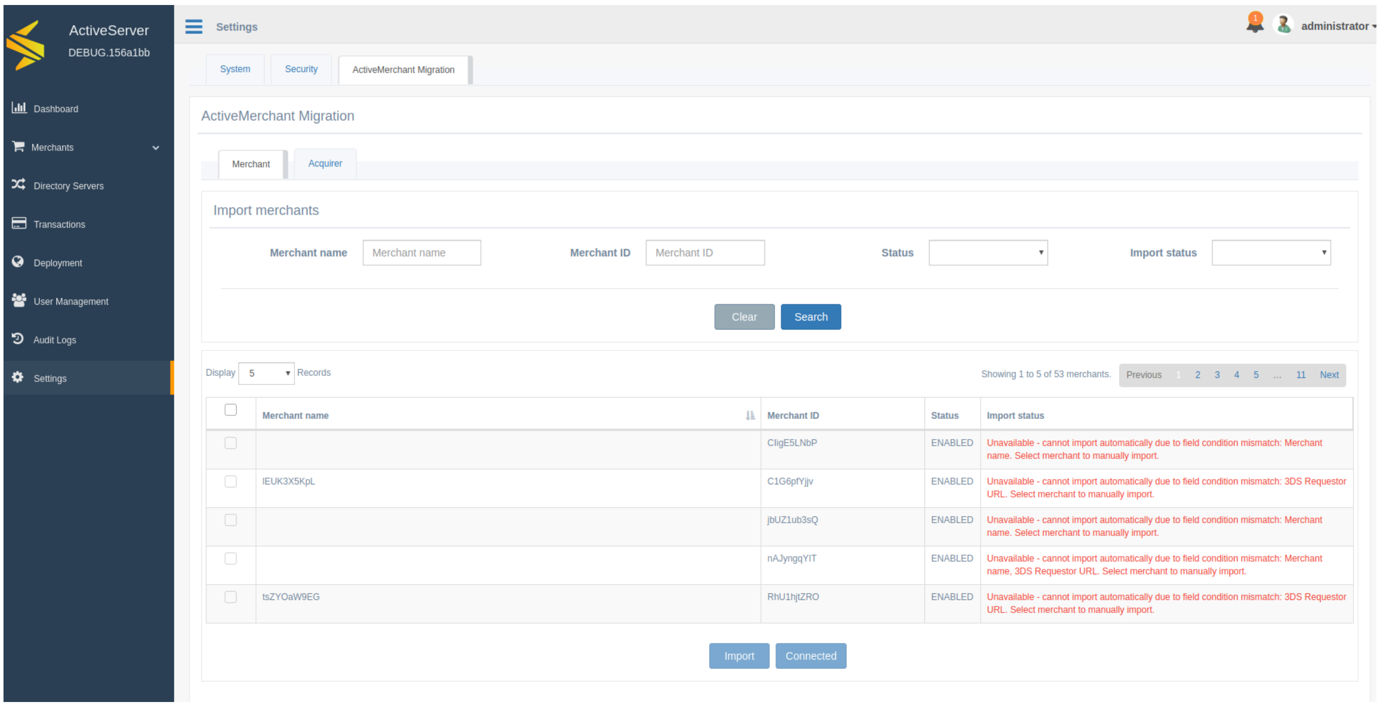
Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
		Delete acquirer				✔		
Directory Servers		View Directory Server settings		✔				
		Edit Directory Server settings		✔				
		View Directory Server certificates		✔				
		Edit Directory Server certificates		✔				
Transactions		View all merchant transactions	All merchants			✔		
		View merchant transactions	Single merchant				✔	✔
Deployment	Nodes	View deployment information		✔				
		Edit deployment information		✔				

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
	Activation Status	View activation details		✔				
		Edit product activation information		✔				
User Management	Search	View all users details	All users		✔			
		Add users			✔			
		Delete users			✔			
	Details	Edit all users details	All users		✔			
		Edit all users roles	All users		✔			
		Edit all users status	All users		✔			
Audit Logs		View all audit logs		✔				
Settings	System	View system settings		✔				
		Edit system settings		✔				

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
	Security	View security settings		✔				
		Edit security settings		✔				
	3D Secure 2	View 3D Secure 2 settings		✔				
		Edit 3D Secure 2 settings		✔				
About		View details		✔	✔	✔		
User profile	Edit profile	View user details	Single user	✔	✔	✔	✔	✔
		Edit user details	Single user	✔	✔	✔	✔	✔
Notifications		View system notifications		✔				
		View user notifications		✔	✔	✔	✔	✔
Reset Password		Reset password		✔	✔	✔	✔	✔

ActiveMerchant migration

The **ActiveMerchant Migration** feature allows a **Business Admin** user to import merchants and acquirers from **GPayments ActiveMerchant** (3DS1 MPI) to assist with the transition from 3DS1 to 3DS2. It can be accessed from the **Administration interface > Settings > ActiveMerchant Migration** tab.



ActiveMerchant Migration

Import merchants

Merchant name	Merchant name	Merchant ID	Merchant ID	Status		Import status	

Display 5 Records

Showing 1 to 5 of 53 merchants. Previous 1 2 3 4 5 ... 11 Next

☐	Merchant name	Merchant ID	Status	Import status
☐		CligE5LNbP	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.
☐	lEUK3X5KpL	C1G6pfYjjv	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: 3DS Requestor URL. Select merchant to manually import.
☐		jbUZ1ub3sQ	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.
☐		nAJyngqYIT	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name, 3DS Requestor URL. Select merchant to manually import.
☐	tsZYOaW9EG	RhU1hjtZRO	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: 3DS Requestor URL. Select merchant to manually import.

Supported ActiveMerchant versions

ActiveMerchant v5.1.12 or above is supported.

Migrate Merchants

All merchants will be assigned an **Import status** which shows if the merchant can be imported or not. They will also have a **Status** to indicate if they were ***Enabled*** or ***Disabled*** in the **ActiveMerchant** database. Both of these options can be used to filter the search results.

Import status

Import status can be one of the following values:

- **Already imported** - The merchant is already imported. ***Merchant name*** and ***Merchant ID*** pair already exist in the system. A merchant with this status will have a disabled checkbox.
- **Unavailable** - The merchant cannot be imported automatically and needs to be imported manually, see the manual import section. A merchant with this status will have a disabled checkbox.
- **Warning** - The merchant is missing a ***Country*** and/or ***Default Currency*** value and can be imported with default values. Default values can be assigned in the popup dialog when initialising the import. The ***Country*** and/or ***Default Currency*** value will be overwritten for this merchant during the import process.
- **Available** - The merchant can be imported automatically with its normal values.

Import merchants manually

When you click on a merchant row you can import the merchant manually by assigning the missing or incorrect fields, or edit any other fields. For merchants with an ***Unavailable*** status, this is the only option for importing.

Migrate Acquirers

All acquirers are assigned an **Import status** which shows if the acquirer can be imported or not.

Import status

Import status can be one of the following values:

- **Already imported** - The acquirer is already imported. Acquirer name already exists in the system. Acquirers with this status will have a disabled checkbox.
- **Available** - The acquirer can be imported automatically with its normal values.

Step-by-step migration

The migration process of merchants and acquirers is very similar. The following steps outline the migration process for merchants.

1. Make sure you have configured the `application-prod.properties` file with your ActiveMerchant database details:

```
as.migration.db.vendor=<ActiveMerchant database vendor> e.g. mysql, oracle, 
mssql, db2 or postgres
as.migration.db.url=<ActiveMerchant database JDBC url> e.g. jdbc:mysql://
<Your My SQL DB Host>:3306/<Your DB Name>
as.migration.db.username=<ActiveMerchant database username>
as.migration.db.password=<ActiveMerchant database password>
```

2. If changes were made to the `application-prod.properties` file, restart your ActiveServer instance.

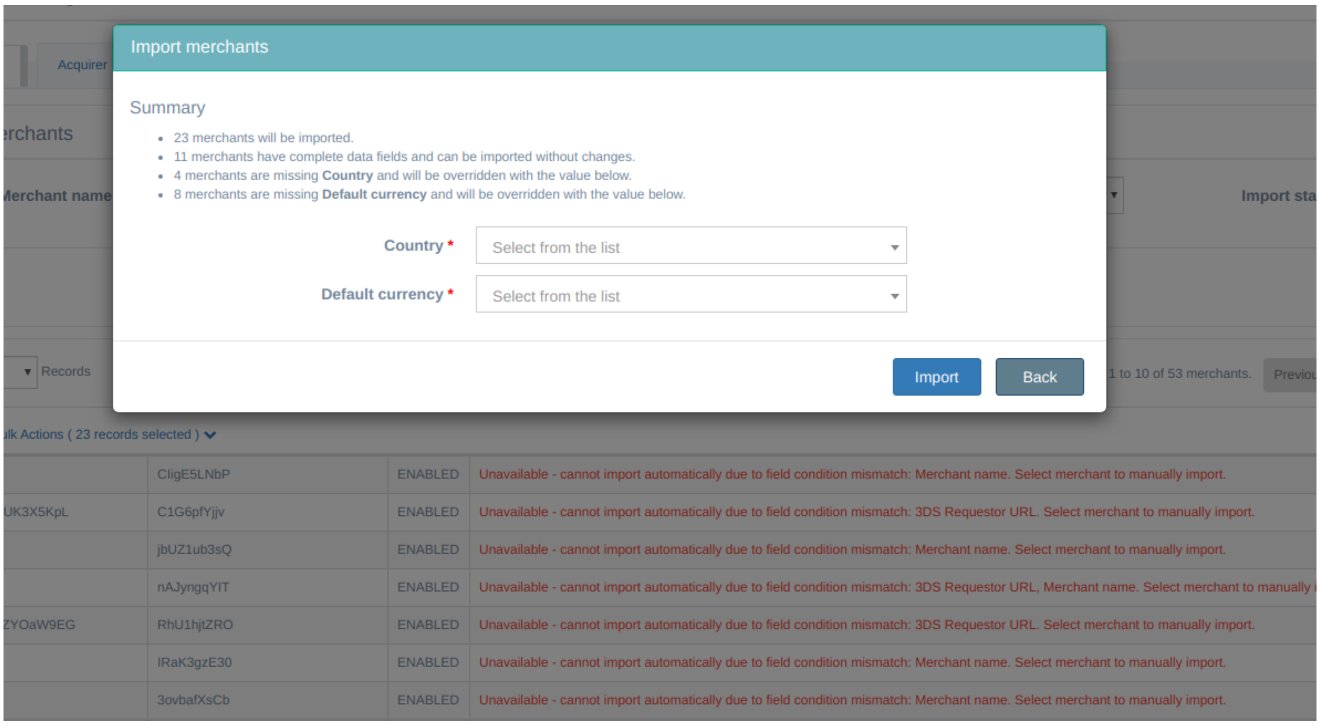
3. Go to the **Administration interface > Settings > ActiveMerchant Migration** tab and select `Connect` to establish a connection between **ActiveServer** and the **ActiveMerchant** database. If connection is unable to be made an error will be shown.

4. Check the checkbox of the merchants/acquirers you would like to import. Check the checkbox in the table header if you want to import all merchants, this will select all entries in the table.

Unavailable merchants

Any merchants that have an import status of ***Unavailable*** cannot be imported automatically as one of the required fields is either missing or not valid. To import these merchants see the manual import section.

5. Select the import button which will popup a dialog similar to below:



Import merchants

Summary

- 23 merchants will be imported.
- 11 merchants have complete data fields and can be imported without changes.
- 4 merchants are missing **Country** and will be overridden with the value below.
- 8 merchants are missing **Default currency** and will be overridden with the value below.

Country * Select from the list

Default currency * Select from the list

Import Back

	CligE5LNbP	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.
UK3X5KpL	C1G6pfYjjv	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: 3DS Requestor URL. Select merchant to manually import.
	jbUZ1ub3sQ	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.
	nAJyngqYIT	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: 3DS Requestor URL, Merchant name. Select merchant to manually i
ZYOaW9EG	RhU1hjtZRO	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: 3DS Requestor URL. Select merchant to manually import.
	IRaK3gzE30	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.
	3ovbafXsCb	ENABLED	Unavailable - cannot import automatically due to field condition mismatch: Merchant name. Select merchant to manually import.

If you have selected merchants with a **Warning** import status, the dialog will ask you to pick default values to be used for importing. This feature is useful if you have several merchants with missing fields and requires the same default currency or country.

6. Select the ***Import*** button to start the import process. Wait until the process is finished and the confirmation dialogue is shown before leaving the page. However, if the process is interrupted, all merchants already imported will be saved and have a **Import Status** of ***Already imported*** the next time import is run again.

FAQ

Should ActiveMerchant be running?

During migration, ActiveMerchant does not need to be running, simply make sure that **ActiveServer** can access the **ActiveMerchant** database.

Does migration affect my ActiveMerchant database?

No, the migration process will only read from the ActiveMerchant database you have configured and will not alter the current **ActiveMerchant** installation.

Do all the merchants or acquirers need to be imported in one session?

No, the migration process can be done multiple times. Each time **ActiveServer** connects to the **ActiveMerchant** database it will automatically set the **Import Status** to ***Already imported*** if existing merchant or acquirer information is detected.

Integration overview

To integrate 3DS2 authentication with a merchant's or payment gateway's eCommerce site, the checkout process of the eCommerce site needs to implement a **3DS Requestor** as per the EMV 3D Secure 2.0 specifications. The **3DS Requestor** implementation in the checkout process will communicate with **ActiveServer** via it's Authentication API and assist with the browser information collecting and 3DS Method process (if any) to finish a 3DS2 authentication.

ActiveServer provides a reference implementation of a **3DS Requestor** in the form of source code to help clients implement the **3DS Requestor** process in their existing checkout process.

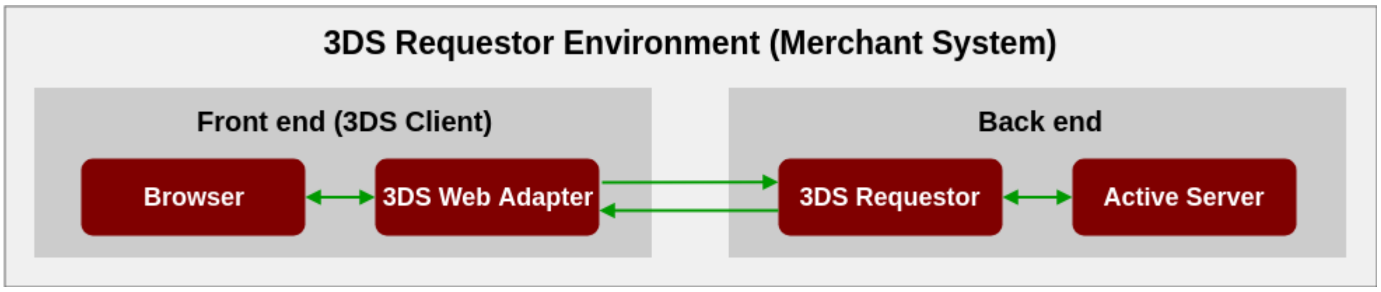
ActiveServer allows external components such as the **3DS Requestor** to access it's features via Restful APIs. These API calls are operations that an application can invoke at runtime to perform certain tasks. All API requests and responses are in JSON format, which is a lightweight format for transporting data.

API documentation

For details of the API documentation, refer to the API document overview.

The **Integration** section of the documentation provides an introductory guide on how to implement a **3DS Requestor** in your merchant site and integrate with **ActiveServer**, then perform a test transaction. For information regarding merchant **App** integration, refer to the **ActiveSDK documentation**.

To utilise 3DS2, the merchant site needs to implement two parts: a **3DS web adapter** at the front end and a **3DS Requestor** at the back end. The following diagram shows the relationship between the browser, the 3DS web adapter, the 3DS Requestor and **ActiveServer**:



- **3DS web adapter** - The 3DS web adapter is a javascript component provided by the **GPayments 3DS Requestor Demo** and is used to pass 3D Secure data from the consumer device to the 3DS Requestor and assist with the browser information collecting/3DS Method process. This component also processes callback events and page forwarding from **ActiveServer**.
- **3DS Requestor** - The 3DS Requestor is the backend component implemented to act as a bridge between the 3DS web adapter and **ActiveServer**. It receives the 3DS authentication requests from the 3DS web adapter, formulates the requests, and sends the requests to **ActiveServer**. It also receives the authentication results from **ActiveServer** and forwards the results to the 3DS web adapter.

Making a transaction

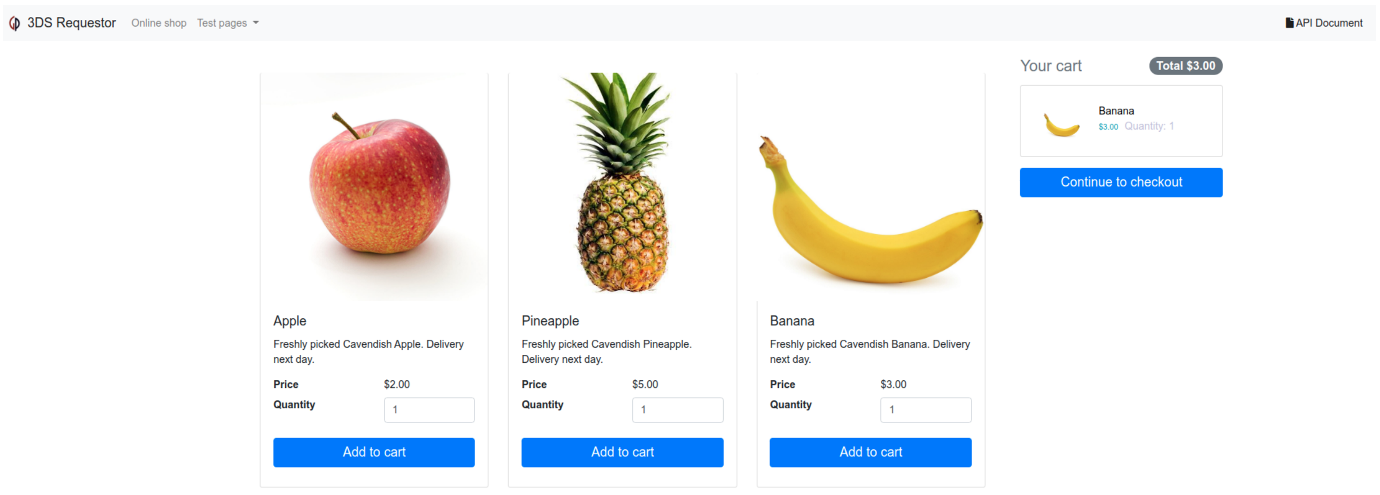
To simulate a transaction with 3DS2, you can use this demo merchant website to see how the Authentication API works.

Tip

As this demo merchant website is used as an example throughout this integration guide, please try using it before now before continuing with integration. All the features are explained here for reference.

Frictionless flow

To initiate a frictionless transaction, open the demo merchant website, launch the **Online shop** page, and add an item to the cart.



Banana
$3.00 Quantity: 1

Apple
Freshly picked Cavendish Apple. Delivery next day.

Price	$2.00
Quantity	1

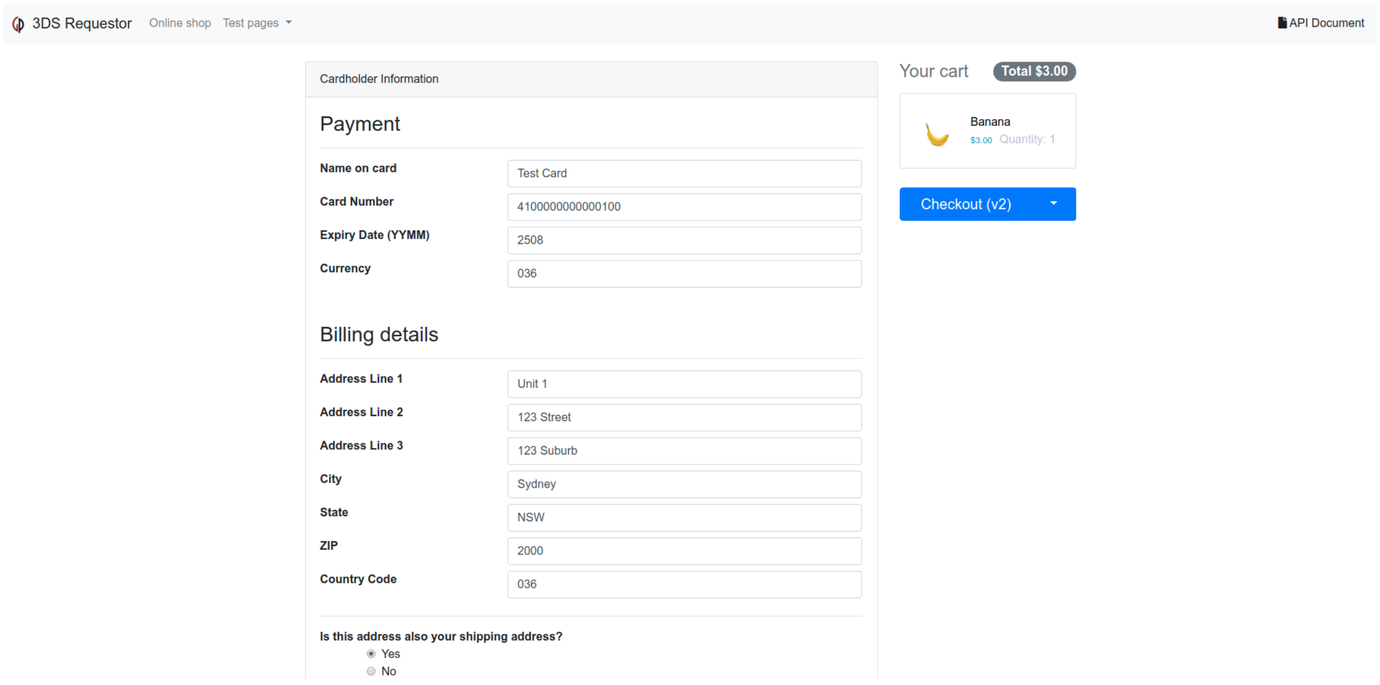
Pineapple
Freshly picked Cavendish Pineapple. Delivery next day.

Price	$5.00
Quantity	1

Banana
Freshly picked Cavendish Banana. Delivery next day.

Price	$3.00
Quantity	1

Select the ***Continue to checkout*** button to move to the checkout page.



Banana
$3.00 Quantity: 1

Cardholder Information

Payment

Name on card	Test Card
Card Number	4100000000000100
Expiry Date (YYMM)	2508
Currency	036

Billing details

Address Line 1	Unit 1
Address Line 2	123 Street
Address Line 3	123 Suburb
City	Sydney
State	NSW
ZIP	2000
Country Code	036

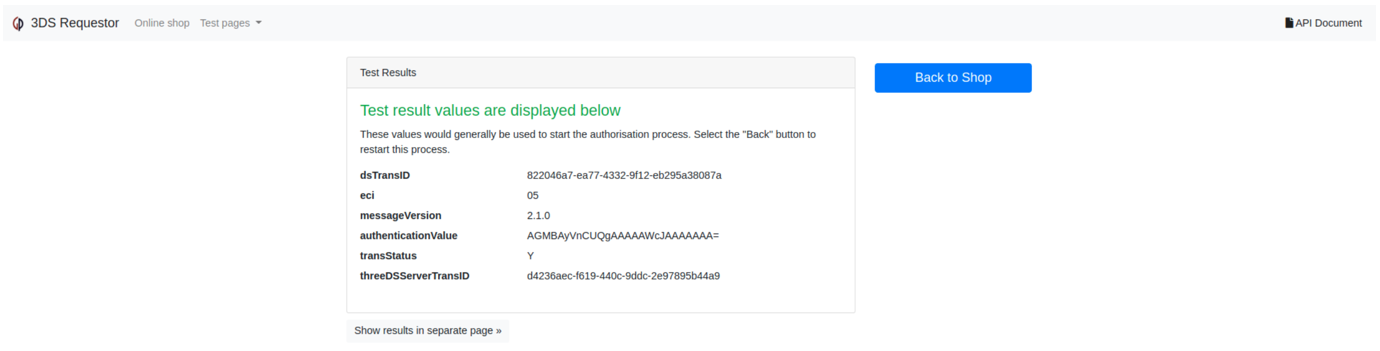
Is this address also your shipping address?
[✓ Yes]
No

Default payment and billing information has been pre-filled, including a card number, which can be used to complete the transaction. Select the ***Checkout*** button to trigger the 3DS2 authentication process.

Note

You can select the API version to perform the 3DS2 authentication process by clicking the arrow at the right of the ***Checkout*** button.

The **3DS web adapter** will collect the cardholder information and send it to the **3DS Requestor**. The **3DS Requestor** will formulate this into an API request and forward it to **ActiveServer**, which will initiate 3DS2 messaging. The **3DS Requestor** will then wait for the authentication result and forward the result back to the **3DS web adapter**, to be displayed on the following web page.



Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

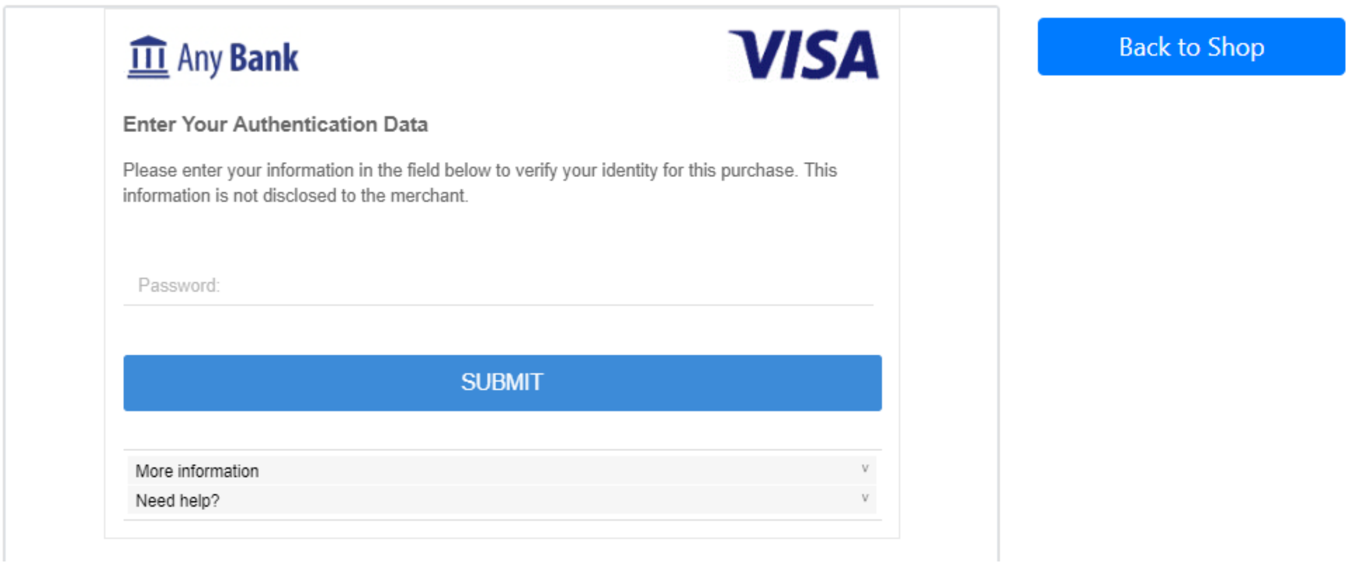
dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

Show results in separate page »

This completes a transaction using the **frictionless flow**. The simulated transaction was deemed as low risk and hence, no challenge was required.

Challenge flow

To test the challenge flow, select the ***Back to Shop*** button and again add an item to the cart and go to the checkout page. This time, use the card number **4100000000005000** and checkout. In this simulation, the transaction has been deemed as high risk and further cardholder interaction is required, thereby initiating the **challenge flow**. The following challenge screen will be displayed, for this demo the password is **123456**.



Enter Your Authentication Data

Please enter your information in the field below to verify your identity for this purchase. This information is not disclosed to the merchant.

Entering the password should result in a successful transaction. In a production scenario, this challenge method could be a variety of different methods, such as **OTP** or **biometrics**, depending on the issuer's ACS and authentication methods registered with the cardholder.

What's next?

Select ***Next*** to learn more about the **Authentication processes**.

Authentication processes

The **3DS Requestor** performs three processes during an authentication:

1. **Initialise Authentication** - the 3DS Requestor sends a request to **ActiveServer** to initialise the authentication, preparing **ActiveServer** for the authentication.
2. **Execute Authentication** - **ActiveServer** executes the authentication. There are two main authentication flows in 3DS2, **frictionless flow** and **challenge flow**, which are described in the Process Flows section.
3. **Get Authentication Result** - the authentication result is returned to the 3DS Requestor.

Process 1: Initialise Authentication

In this step the **3DS web adapter** collects the information that the cardholder has entered at the front end and passes it to the **3DS Requestor** at the back end. The **3DS Requestor** then passes all the information required by 3DS2 to **ActiveServer** to start the authentication.

Using our example in the integration overview, when the customer selects ***Checkout***, the **3DS web adapter** sends an `initialise authentication` message to the 3DS Requestor.

The 3DS Requestor receives the `initialise authentication` message, formats it according to the **ActiveServer** API, and generates and adds a unique **3DS Requestor transaction ID** (`threeDSRequestorTransID`) to the message. Once the message is populated, it will be sent to **ActiveServer**.

When **ActiveServer** receives the `initialise authentication` message it will return the callback URLs, i.e. `threeDSServerCallbackUrl` for the **3DS Requestor** to setup page forwarding mechanisms on the checkout page (hidden iframes are used for current implementation of the callbacks in `3ds Web Adapter`). Once the iframes are setup, **ActiveServer** will be able to start the browser information collecting process and be ready for the authentication process.

Note

ActiveServer and the ACS automatically collect the browser information, and this process is **NOT** part of the **3DS Requestor**.

The only thing that may be related to this process in the **3DS Requestor** environment is that the `3DS web adapter` sets up the required hidden iframes automatically in the checkout page in order for the callback pages to be executed.

Process 2: Execute Authentication

Once the browser information collection is complete, authentication can be executed. This will trigger **ActiveServer** to initiate the 3DS2 messaging process.

What if browser information collection failed?

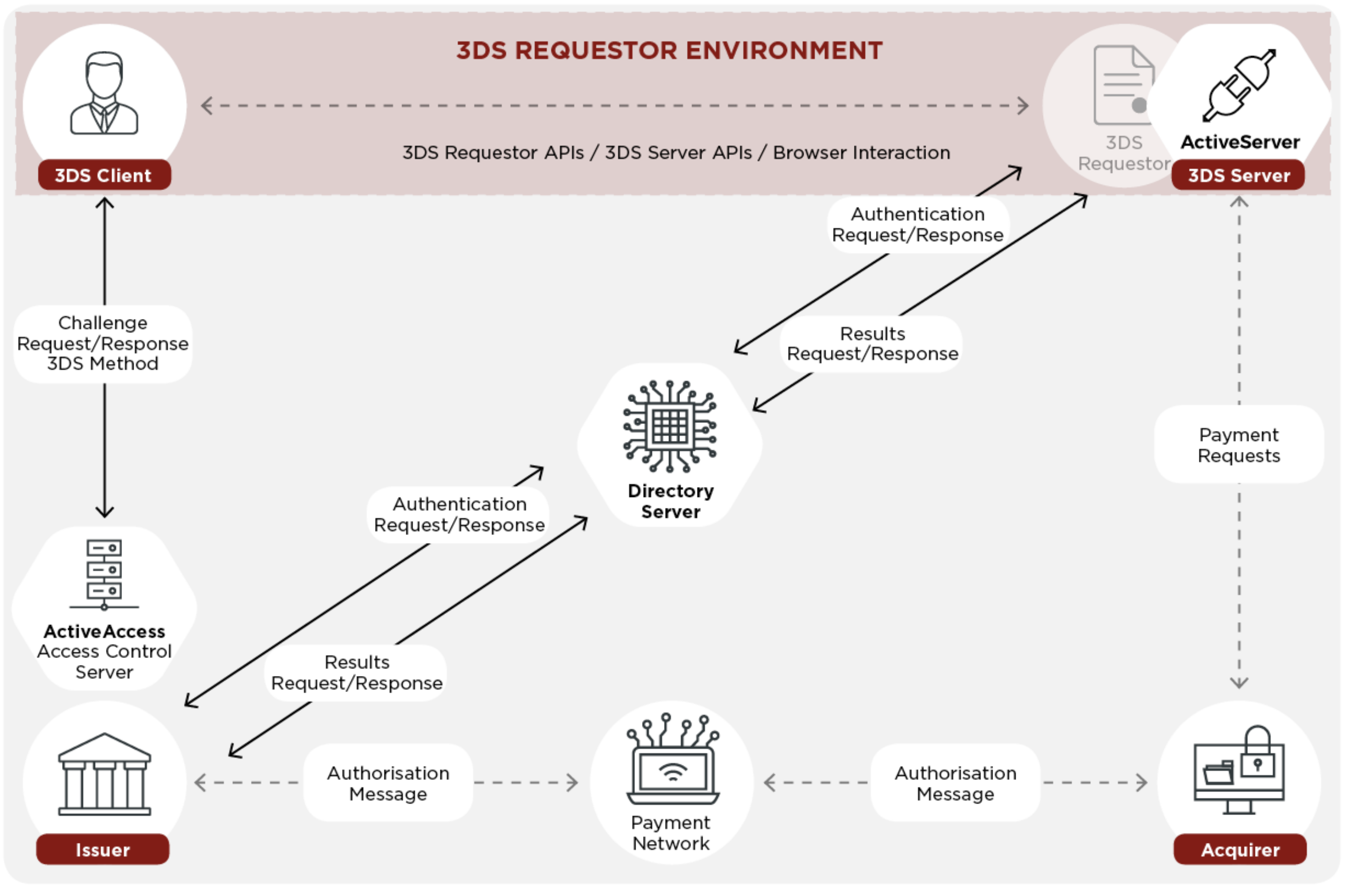
It is not uncommon that the browser information collection process may fail or not be started at all. In this scenario, to improve user experience, **ActiveServer** will return an `InitAuthTimedOut` event (15 seconds after `InitAuth` request is called and no `brw` call thereafter) to the `3ds web adapter` to indicate that **Process 1** has failed. The checkout page can then handle this error accordingly.

Without this timeout mechanism, the checkout process could be stuck in the middle of processing as there won't be any callbacks sent by **ActiveServer**.

There are two main authentication flows in 3DS2, **frictionless flow** and **challenge flow**.

- **Frictionless flow** - initiates a 3D Secure authentication flow, which consists of the **AReq**/**ARes** authentication messages. If the ACS determines the transaction is low risk based on the information provided, authentication is approved immediately.
- **Challenge flow** - if the ACS determines that the transaction is high risk, above certain thresholds, or subject to regulatory mandates which requires further Cardholder interaction, the **frictionless flow** transitions into the **challenge flow**. In addition to the **AReq** and **ARes** messages that comprise **frictionless flow**, the **challenge flow** consists of the **CReq**/**CRes** challenge messages and the **RReq**/**RRes** result messages.

This diagram shows the challenge flow:


3DS REQUESTOR ENVIRONMENT
3DS Client
3DS Requestor APIs / 3DS Server APIs / Browser Interaction
3DS Requestor
ActiveServer
3DS Server
Authentication Request/Response
Results Request/Response
Challenge Request/Response 3DS Method
Directory Server
Payment Requests
Authentication Request/Response
ActiveAccess Access Control Server
Results Request/Response
Issuer
Authorisation Message
Payment Network
Authorisation Message
Acquirer

> *The dotted lines indicate messaging outside the scope of the 3DS2 protocol, including communication between the Client/3DS Requestor and Authorisation*

Process 3: Get Authentication Result

Once the 3DS2 process is complete, the merchant will get the authentication result. The authentication result (from the ARes or RRes depending on challenge status) contains information such as as the ECI, Authentication Value (e.g. CAVV) and final Transaction Status.

Note that the authentication result is also available in the response of **Process 2** if the authentication process is frictionless. However, the merchant site can always get the authentication result by calling the `Get Authentication Result` API.

What's next?

Select **Next** to learn more about the **Authentication sequence**.

Authentication sequence

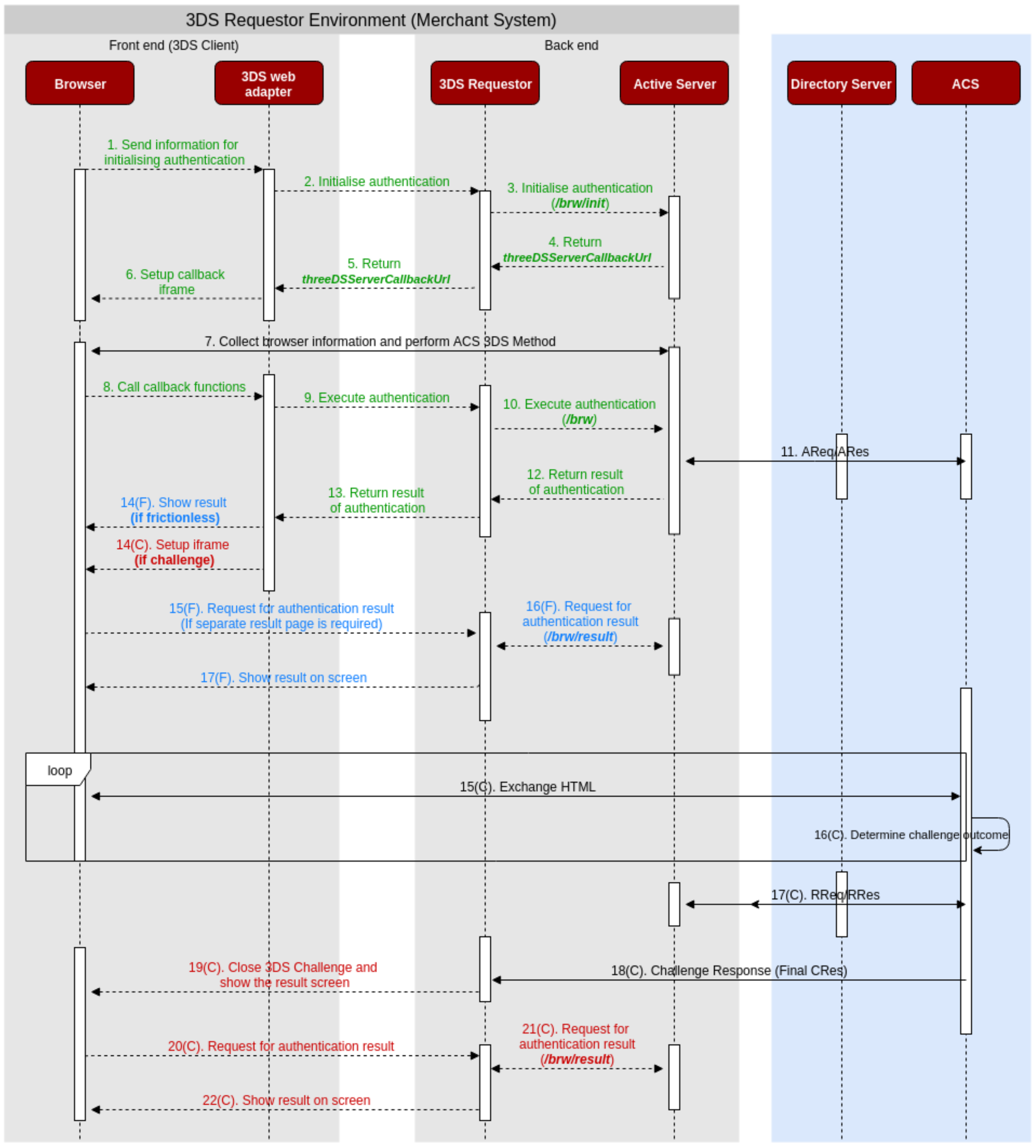
The sequence diagram below breaks down the process of a 3DS2 authentication, step by step, explicitly focusing on how the **3DS Requestor** functions inside the 3DS2 flow, using **GPayments'** APIs.

If any of the steps are part of the **3DS Requestor environment** process, they will be marked as ←3DS Requestor process for your reference, as these steps are provided as demo code and may require customisation to fit your requirements.

Note

Implementation of the **3DS web adapter** at the front end, and the **3DS Requestor** at the back end, is required to integrate ActiveServer.

- **Process 1: Initialise Authentication**
    - Step 1 to Step 7
- **Process 2: Execute Authentication**
    - Frictionless flow - Step 8 to Step 13, and Step 14(F)
    - Challenge flow - Step 8 to Step 13, and Step 14(C) to Step 19(C)
- **Process 3: Get Authentication Result**
    - Frictionless flow - Step 15(F) to Step 17(F)
    - Challenge flow - Step 20(C) to Step 22(C)


3DS Requestor Environment (Merchant System)
Front end (3DS Client)
Back end
Browser
3DS web adapter
3DS Requestor
Active Server
Directory Server
ACS
1. Send information for initialising authentication
2. Initialise authentication
3. Initialise authentication (/brw/init)
4. Return threeDSServerCallbackUrl
5. Return threeDSServerCallbackUrl
6. Setup callback iframe
7. Collect browser information and perform ACS 3DS Method
8. Call callback functions
9. Execute authentication
10. Execute authentication (/brw)
11. AReq/ARes
12. Return result of authentication
13. Return result of authentication
14(F). Show result (if frictionless)
14(C). Setup iframe (if challenge)
15(F). Request for authentication result (If separate result page is required)
16(F). Request for authentication result (/brw/result)
17(F). Show result on screen
loop
15(C). Exchange HTML
16(C). Determine challenge outcome
17(C). RReq/RRes
18(C). Challenge Response (Final CRes)
19(C). Close 3DS Challenge and show the result screen
20(C). Request for authentication result
21(C). Request for authentication result (/brw/result)
22(C). Show result on screen

- Dashed arrows are messages that are part of 3DS Requestor
- Solid arrows are messages that are not part of 3DS Requestor
- Messages in green are common to both frictionless and challenge flow
- Messages in blue are frictionless flow specific, marked with (F)
- Messages in red are challenge flow specific, marked with (C)

1. **Send information for initialising authentication** ←3DS Requestor process
    - Information for initialising authentication obtained from the checkout page, such as the card number, is sent to the **3DS web adapter**. This is a simple JavaScript simulating how the front end system of 3DS Requestor works.

2. **Initialise authentication** ←3DS Requestor process
    - **3DS web adapter** makes a POST request to the **3DS Requestor** with information collected from the checkout page and requests the **3DS Requestor** to initialise authentication.
3. **Initialise authentication** ←3DS Requestor process
    - The **3DS Requestor** obtains the information from the front end and makes a POST API call to the **`initAuth`** end point to initialise authentication.
    - An important field sent here is the `eventCallbackUrl`, which will be required to start Step 8 to allow **ActiveServer** to callback to this URL to notify the end of the browser information collection.
4. **Return `threeDSServerCallbackUrl`**
    - A successful response from **`initAuth`** end point contains `threeDSServerCallbackUrl` and `threeDSServerTransID`.
5. **Return `threeDSServerCallbackUrl`** ←3DS Requestor process
    - The **3DS Requestor** returns the `threeDSServerCallbackUrl` back to the **3DS web adapter**.
6. **Setup callback `iframe`** ←3DS Requestor process
    - Insert a hidden `iframe`, with `src` set to `threeDSServerCallbackUrl`. This will allow **ActiveServer** to connect to the **3DS Requestor**. **ActiveServer** will callback to this `iframe`.
7. **Collect browser information**
    - **ActiveServer** collects the browser information via the `iframe` and then facilitates the 3DS method data collecting for the **ACS**. The **ACS** then collects the 3DS method data via the prepared `iframe`.
8. **Call callback function** ←3DS Requestor process
    - During authentication initialisation, the **3DS Requestor** sends the `eventCallBackUrl` so that the ACS can notify the **3DS Requestor** when the 3DS method is finished or skipped. This is done through the `iframe` setup in Step. 7.
    - Once the **3DS Requestor** receives a notification, it will pass required parameters including `callbackFn` and render `notify-3ds-events.html` to the `iframe`. When `notify_3ds_events.html` is rendered it will simply call the `callbackFn` defined in the `3ds-web-adapter`.

9. **Execute authentication** ←3DS Requestor process
    - `callbackFn` can be either `_on3DSMethodSkipped()` or `_on3DSMethodFinished()`, both of which will end up calling `doAuth()`. 3DS-web-adapter will call `doAuth()`, which will ask the 3DS Requestor to execute the authentication.
    - `_on3DSMethodSkipped` means that ACS did not perform browser information collection for some reason. Therefore, if you want to perform 3DS authentication when browser information is collected, you can choose to terminate the transaction here.
10. **Execute authentication** ←3DS Requestor process
    - The **3DS Requestor** will make a call to the `auth` end point, which will initiate the authentication processes.
11. **AReq/ARes**
    - An Authentication Request (AReq) is sent from **ActiveServer** via the Directory Server to an ACS. An Authentication Response (ARes) containing the authentication results is sent from the ACS to **ActiveServer**.
12. **Return result of authentication**
    - `auth` end point returns a `tranStatus` to the **3DS Requestor**.
13. **Return result of authentication** ←3DS Requestor process
    - Return the result of authentication back to the web adaptor.

Info

If the `transStatus` returned is "Y" go to `Step 14(F)`, if "C" go to `Step 14(C)`.

Frictionless flow specific

14(F). **Show Result (if frictionless)** ←3DS Requestor process

- If the the authentication result has a `transStatus` of "Y", `authSuccess()` is called, which redirects the page to `/auth/result?transId`.

15(F). **Request for authentication result (if separate result page is required)** ←3DS Requestor process

- The browser notifies the **3DS Requestor** with the `transId`, and the the transaction result is available for request.

16(F). **Request for authentication result** ←3DS Requestor process

- The **3DS Requestor** will ask for a result receipt from **ActiveServer** by calling the `result` end point.

17(F). **Show result on screen** ←3DS Requestor process

- The result screen is rendered using the authentication result.

Challenge flow specific

14(C). **Setup `iframe` (if challenge)** ←3DS Requestor process

- If the authentication result has a `transStatus` of "C", `startChallenge()` is called, which will insert an `iframe` for challenge.

15(C). **Exchange HTML**

- The ACS will embed the challenge screen inside the `iframe`, then the cardholder completes the authentication challenge.

16(C). **Determine challenge outcome**

- The ACS determines if the challenge performed is successful or not.

17(C). **RReq/RRes**

- The ACS sends a Result Request (RReq) containing the authentication results via the Directory Server to **ActiveServer**. **ActiveServer** will then acknowledge its receipt with a Result Response (RRes).

18(C). **Challenge Response (final CRes)**

- The ACS sends the final Challenge Response (CRes) to the **3DS Requestor**.

19(C). **Close 3DS Challenge and show the result screen** ←3DS Requestor process

- Since the challenge is finished, the **3DS Requestor** redirects the page to `/auth/result?transId`.

20(C). **Request for authentication result** ←3DS Requestor process

- The browser notifies the **3DS Requestor** with the `transId`, and the the transaction result is available for request.

21(C). **Request for authentication result** ←3DS Requestor process

- The **3DS Requestor** asks for a result receipt from the `result` end point, same as Step 16(F).

22(C). **Show result on screen** ←3DS Requestor process

- The result screen is rendered using the authentication result on the browser.

What's next?

Access the **Integration guide** and go through the process of integrating a merchant checkout process with **ActiveServer**.

API migration from v1 to v2

This is a reference guide for migrating from **ActiveServer API v1** to **API v2**.

The sunset timing for using **API v1** for on-premise clients is currently planned for Q4 2020, which will be communicated to clients closer to the date. Any clients using the **GPayments Hosted SaaS** will be required to use **API v2** from service launch.

Major change overview

- Added `/api/v2/auth/` authentication API entry points.
- **ActiveServer** no longer stores encrypted cardholder PANs in the database, only the truncated PAN is stored (first 6 and last 4 digits). This improves security by significantly reducing the impact of a breach and potentially reduces some PCI compliance requirements. PAN search functionality will be limited to the first 6 and last 4 digits when searching for **API v2** transactions.
- Due to encrypted PANs no longer being stored, per merchant encryption keys are no longer required when using **API v2**. They are still required for **API v1** transactions.
- **GPayments** 3DS Requestor demo code has been upgraded to support both **API v1** and **API v2**.

ActiveServer changes

As of **ActiveServer** version 1.3.0, **ActiveServer** will no longer store an encrypted PAN for transactions executed via **API v2**, and instead will only store a truncated PAN. Only the first 6 digits and last 4 digits will be stored in plain text and be visible to users, e.g. a PAN of `4123456789876543` will be stored as `412345XXXXXX6543` in the database and shown as the same on the administration interface.

As a result, per merchant encryption keys are only used for **API v1** transactions and will be removed in a future release.

As **ActiveServer** no longer stores full PANs, the transaction search function in Admin UI will no longer support full PAN matching for those executed via **API v2**. Although, partial PAN search is still available. Users can enter a full PAN in the search box as usual, and the results will return all transactions that match the first 6 and last 4 digits of the entered PAN. Combining partial PAN search along with other attributes e.g. `Transaction ID`, `Date/Time`, `Purchase amount` or `Currency`, should allow a particular transaction to be identified. Transaction search functionality for **API v1** transactions is not affected.

By implementing API v2, clients may enjoy the following benefits:

- Improved performance: ActiveServer no longer needs to call transaction encryption functions
- Increased security: Reduces the impact of a security breach
- Reduced time and costs for PCI auditing: Potentially reduces the scope of the Cardholder Data Environment (CDE).

Important

Note the above changes are only applicable when using **Auth API v2**. Read below for the impact on **API v1**.

Auth API v1 impact and migration plan

Transactions using **Auth API v1** will be unaffected by the changes mentioned above. Current integrations will continue to work, allowing for a stable upgrade path to **API v2** to be planned. The current sunset timing is planned for Q4 2020.

All past **API v1** transactions will still have encrypted PANs using per merchant encryption keys stored in the database. To assist clients transition smoothly into from API v1 to API v2, an API v2 migration utility will be provided in a future release, and will contain the following functionality:

- Convert all **API v1** transactions into the new **API v2** format
- Clear up the encryption keystores
- Disable **API v1**

GPayments encourages all ActiveServer clients to upgrade to **API v2** as soon as possible, to take advantage of all its included benefits.

API Changes - migrating to v2 BRW Auth API

The main change to the **v2 BRW API** is that all cardholder fields are now sent in the `/api/v2/auth/brw` (`Execute Authentication`) API call, rather than in `/api/v2/auth/brw/init` (`Initialise Authentication`). The `Initialise authentication` process will be used simply to collect pre-authentication browser information, and to execute the **3DS Method** (optional additional browser information collection by the ACS) if it is available.

See below for the JSON message changes for the requests.

- **Red highlight** - the field is removed.
- **Green highlight** - the field is added.
- **No highlight** - the field is unchanged from **API v1**.

Please refer to the API documentation for detailed field descriptions.

Changes to `Initialise Authentication`

- The `{messageCategory}` parameter has been removed from the request message URL and added to the request body of `Execute Authentication (/api/v2/auth/brw)`.
- The transaction related fields removed from the request message below have been added to `Execute authentication`.
- The `authUrl` field has been added to the response message, this is now used as the URL to send the `Execute authentication` request.
- The error code fields are no longer included in a successful response message.

Request body

Below shows the difference of the request body between `/api/v1/auth/brw/init/{messageCategory}` and `/api/v2/auth/brw/init`:

```
{
-   "acctID": "ActiveServer 3DS Test Account 000000001",
-   "acctInfo": {
-     "chAccAgeInd": "03",
-     "chAccChange": "20160712",
-     "chAccChangeInd": "04",
-     "chAccDate": "20140328",
-     "chAccPwChange": "20170328",
-     "chAccPwChangeInd": "02",
-     "nbPurchaseAccount": "11",
-       "paymentAccAge": "20160917",
-       "paymentAccInd": "04",
-       "provisionAttemptsDay": "3",
-       "shipAddressUsage": "20160714",
-       "shipAddressUsageInd": "04",
-       "shipNameIndicator": "02",
-       "suspiciousAccActivity": "01",
-       "txnActivityDay": "1",
-       "txnActivityYear": "21"
-   },
    "acctNumber": "7654310438720050",
-   "acctType": "03",
-   "authenticationInd": "01",
-   "authenticationInfo": {
-       "threeDSReqAuthData": "validlogin at UL TS BV",
-       "threeDSReqAuthMethod": "02",
-       "threeDSReqAuthTimestamp": "201711071307"
-   },
-   "cardExpiryDate": 1910,
-   "cardHolderInfo": {
-       "billAddrCity": "Bill City Name",
-       "billAddrCountry": 840,
-       "billAddrLine1": "Bill Address Line 1",
-       "billAddrLine2": "Bill Address Line 2",
-       "billAddrLine3": "Bill Address Line 3",
-       "billAddrPostCode": "Bill Post Code",
-       "billAddrState": "CO",
-       "cardholderName": "Cardholder Name",
-       "email": "example@example.com",
-       "homePhone": {
-         "cc": "123",
-         "subscriber": "123456789"
-       },
-       "mobilePhone": {
-         "cc": "123",
-         "subscriber": "123456789"
-       },
-       "shipAddrCity": "Ship City Name",
-       "shipAddrCountry": "840",
-       "shipAddrLine1": "Ship Address Line 1",
-     "shipAddrLine2": "Ship Address Line 2",
-     "shipAddrLine3": "Ship Address Line 3",
-     "shipAddrPostCode": "Ship Post Code",
-     "shipAddrState": "CO",
-     "workPhone": {
-       "cc": "123",
-       "subscriber": "123456789"
-     }
-   },
-   "challengeInd": "02",
    "eventCallbackUrl": "https://example.requestor.com/3ds-notify",
    "merchantId": "1234567890123456789012345678901234",
-   "merchantName": "string",
-   "merchantRiskIndicator": {
-     "deliveryEmailAddress": "deliver@email.com",
-     "deliveryTimeframe": "01",
-     "giftCardAmount": "337",
-     "giftCardCount": "02",
-     "giftCardCurr": "840",
-     "preOrderDate": "20170519",
-     "preOrderPurchaseInd": "02",
-     "reorderItemsInd": "01",
-     "shipIndicator": "02"
-   },
-   "payTokenInd": true,
-   "priorTransID": "59ae264e-b0f4-43c7-870e-4d14bd52806e",
-   "purchaseAmount": "12345",
-   "purchaseCurrency": "978",
-   "purchaseDate": "20180122153045",
-   "purchaseInstalData": "024",
-   "recurringExpiry": "20180131",
-   "recurringFrequency": "2",
    "threeDSRequestorTransID": "2409a5df-b777-4ebc-ad59-2a61091187f1",
-   "transType": "03"
}
```

Response

Below shows the difference of the response body between `/api/v1/auth/brw/init/{messageCategory}` and `/api/v2/auth/brw/init`:

```
{
- "errorCode": "1005",
- "errorComponent": "A",
- "errorDescription": "Data element not in the required format. Not numeric or
wrong length.",
- "errorDetail": "billAddrCountry,billAddrPostCode,dsURL",
- "errorMessageType": "AReq",
+ "authUrl": "https://demo.3dsserver.com/api/v2/auth/brw?
t=dbb270aa890028817afe58fda659526e",
  "monUrl": "https://demo.3dsserver.com/brw/init/mon?
t=6afa6072-9412-446b-9673-2f98b3ee98a2",
  "threeDSServerCallbackUrl": "https://demo.3dsserver.com/brw/callback?
transId=6afa6072-9412-446b-9673-2f98b3ee98a2",
  "threeDSServerTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2"
}
```

Important

API v2 requires the `Execute Authentication` process to utilise a dynamic URL (`authUrl`) generated by **ActiveServer. The 3DS Requestor is required to use this URL instead of the hardcoded one in the v1 3DS Requestor demo code.**

Changes to `Execute Authentication`

- The entry point of `Execute Authentication (/api/v2/auth/brw)` has been changed to use `authUrl` returned by the response of `Initialise Authentication`.
- Cardholder and transaction information parameters removed from `Initialise Authentication` have been added to the request body.
- The `acctNumber` field has been added.
- The `browserInfo` field has been added, which is the result of the browser info collection process. More information can be found in the 3DS Requestor changes section below.
- The **API v1** `Initialise Authentication {messageCategory}` parameter has been removed from the **API v1** URL and added to the request body.

- The optional `addrMatch` field has been added under the `cardHolderInfo` JSON object. `Y` should be set if billing address and shipping address is the same, otherwise `N` should be set. Billing and shipping address information must still be provided.
- The `threeDSRequestorTransID` field has been removed as it is no longer needed.
- The error code fields are no longer included in a successful response message.

Request body

Below shows the difference of the request body between `/api/v1/auth/brw` and `/api/v2/auth/brw`:

```
{
+  "acctID": "ActiveServer 3DS Test Account 000000001",
+  "acctInfo": {
+   "chAccAgeInd": "03",
+   "chAccChange": "20160712",
+   "chAccChangeInd": "04",
+   "chAccDate": "20140328",
+   "chAccPwChange": "20170328",
+   "chAccPwChangeInd": "02",
+   "nbPurchaseAccount": "11",
+    "paymentAccAge": "20160917",
+    "paymentAccInd": "04",
+    "provisionAttemptsDay": "3",
+    "shipAddressUsage": "20160714",
+    "shipAddressUsageInd": "04",
+    "shipNameIndicator": "02",
+    "suspiciousAccActivity": "01",
+    "txnActivityDay": "1",
+    "txnActivityYear": "21"
+  },
+  "acctNumber": "7654310438720050",
+  "acctType": "03",
+  "authenticationInd": "01",
+  "authenticationInfo": {
+    "threeDSReqAuthData": "validlogin at UL TS BV",
+    "threeDSReqAuthMethod": "02",
+    "threeDSReqAuthTimestamp": "201711071307"
+  },
+  "browserInfo": "Base64 encoded browser informaction", --New field added from 
V2
+  "cardExpiryDate": "1910",
+  "cardHolderInfo": {
+    "addrMatch": "N",
+    "billAddrCity": "Bill City Name",
+    "billAddrCountry": 840,
+    "billAddrLine1": "Bill Address Line 1",
+    "billAddrLine2": "Bill Address Line 2",
+    "billAddrLine3": "Bill Address Line 3",
+    "billAddrPostCode": "Bill Post Code",
+    "billAddrState": "CO",
+    "cardholderName": "Cardholder Name",
+    "email": "example@example.com",
+    "homePhone": {
+      "cc": "123",
+      "subscriber": "123456789"
+    },
+    "mobilePhone": {
+      "cc": "123",
+      "subscriber": "123456789"
+    },
+     "shipAddrCity": "Sh"ip" City Name",
+     "shipAddrCountry": 840,
+     "shipAddrLine1": "Ship Address Line 1",
+     "shipAddrLine2": "Ship Address Line 2",
+     "shipAddrLine3": "Ship Address Line 3",
+     "shipAddrPostCode": "Ship Post Code",
+     "shipAddrState": "CO",
+     "workPhone": {
+       "cc": "123",
+       "subscriber": "123456789"
+     }
+   },
+   "challengeInd": "02",
+   "merchantName": "Test Merchant",
+   "merchantRiskIndicator": {
+     "deliveryEmailAddress": "deliver@email.com",
+     "deliveryTimeframe": "01",
+     "giftCardAmount": "337",
+     "giftCardCount": "02",
+     "giftCardCurr": "840",
+     "preOrderDate": "20170519",
+     "preOrderPurchaseInd": "02",
+     "reorderItemsInd": "01",
+     "shipIndicator": "02"
+   },
+   "messageCategory": "pa",
+   "payTokenInd": true,
+   "priorTransID": "59ae264e-b0f4-43c7-870e-4d14bd52806e",
+   "purchaseAmount": "12345",
+   "purchaseCurrency": "978",
+   "purchaseDate": "20180122153045",
+   "purchaseInstalData": "024",
+   "recurringExpiry": "20180131",
+   "recurringFrequency": "2",
-   "threeDSRequestorTransID": "2409a5df-b777-4ebc-ad59-2a61091187f1",
    "threeDSServerTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2",
+   "transType": "03"
}
```

Response

Below shows the difference of the response body between `/api/v1/auth/brw` and `/api/v2/auth/brw`:

```
{
  "acsChallengeMandated": "Y",
  "acsReferenceNumber": "3DS_GP_ACS_201_13579",
  "acsTransID": "375d90ad-3873-498b-9133-380cbbc8d99d",
  "authenticationType": "02",
  "authenticationValue": "MTIzNDU2Nzg5MDA5ODc2NTQzMjE=",
  "cardholderInfo": "For example, Additional authentication is needed for this
transaction, please contact (Issuer Name) at xxx-xxx-xxxx. \n Length: Maximum
128 characters\n Optional",
  "challengeUrl": "https://demo.acs.com/challenge",
  "dsReferenceNumber": "string",
  "dsTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2",
  "eci": "02",
- "errorCode": "1005",
- "errorComponent": "A",
- "errorDescription": "Data element not in the required format. Not numeric or
wrong length.",
- "errorDetail": "billAddrCountry,billAddrPostCode,dsURL",
- "errorMessageType": "AReq",
  "messageVersion": "string",
  "threeDSServerTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2",
  "transStatus": "Y",
  "transStatusReason": 11
}
```

Changes to `Result`

- The `authenticationType` and `interactionCounter` fields have been added to provide more information about the cardholder challenge that was performed.
- The `acsReferenceNumber` and `dsReferenceNumber` fields have been removed as they are already provided in `Execute Authentication`.
- The error code fields are no longer included in a successful response message.

Request

No changes were made to the request body.

Response

Below shows the difference of the response body between `/api/v1/auth/brw/result` and `/api/v2/auth/brw/result`:

```diff
{
- "acsReferenceNumber": "3DS_GP_ACS_201_13579",
  "acsTransID": "375d90ad-3873-498b-9133-380cbbc8d99d",
+ "authenticationType": "02",
  "authenticationValue": "MTIzNDU2Nzg5MDA5ODc2NTQzMjE=",
- "dsReferenceNumber": "string",
  "dsTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2",
  "eci": "02",
+ "interactionCounter": "02",
- "errorCode": "1005",
- "errorComponent": "A",
- "errorDescription": "Data element not in the required format. Not numeric or 
wrong length.",
- "errorDetail": "billAddrCountry,billAddrPostCode,dsURL",
- "errorMessageType": "AReq",
  "messageVersion": "string",
  "threeDSServerTransID": "6afa6072-9412-446b-9673-2f98b3ee98a2",
  "transStatus": "Y",
  "transStatusReason": 11
}
```

API Changes - migrating to v2 APP Auth API

- Path has been changed from `/api/v1/auth/app/{messageCategory}` to `/api/v2/auth/app`. The `{messageCategory}` field has been moved from the URL to the request body.
- The optional `addrMatch` field has been added under the `cardHolderInfo` JSON object. `Y` should be set if billing address and shipping address is the same, otherwise `N` should be set. Billing and shipping address information must still be provided.
- Added the `interactionCounter` field to the response of `/api/v2/auth/app/result`, this value may be returned by the ACS to indicate the number of authentication cycles attempted by the cardholder.

API Changes - migrating to v2 3RI Auth API

- Path has been changed from `/api/v1/auth/3ri/{messageCategory}` to `/api/v2/auth/3ri`. The `{messageCategory}` field has been moved from the URL to the request body.

3DS Requestor changes

- The 3DS Requestor backend code has been updated to store the returned `authUrl` in the `initAuth` response in the web server sessions and will be reused in the later `Execute Authentication` call in the **BRW** channel.
- When the 3DS Requestor receives `3DSMethodFinished` or `3DSMethodSkipped` event at the `eventCallbackUrl`, the `param` parameter will contain browser information collected by the ActiveServer in Base64 encoded form. A new `3ds-web-adapter` v2 has been added to the 3DS Demo code to cater for this change. The 3DS Requestor is required to include this Base64 encoded browser information unchanged in the field `browserInfo` in the `Execute Authentication` request.
- 3DS Requestor will now pass the PAN twice to the 3DS server through the API calls `/api/v2/auth/brw/init` and `/api/v2/auth/brw` due to **ActiveServer** no longer storing the encrypted PAN in the database.

Introduction

The **Integration guide** set of documents will take you through the process of integrating the 3DS2 authentication flow with a sample merchant checkout site from the beginning. This is to implement the **3DS Requestor and 3DS Client** in the merchant checkout process which is called **3DS Requestor Environment** in the EMV 3DSecure 2.0 specification.

This integration guide is based on the provided demo 3DS requestor project which consists of two parts:

- The HTML/javascript frontend, contains a demo online shop and test pages for `BRW`, `3RI` and `Enrol` API calls.
- The backend, which hosts the frontend pages and resources as well as performing the page forwarding and calling the **ActiveServer** authentication API with `x509` authentication.

For reference, the frontend web pages and javascript implementation uses the following framework/components:

- Bootstrap 4.1.3
- Font Awesome 5.2.0
- Moment.js
- JQuery 3.3.1

Prerequisites

The following are prerequisites to using this guide:

- Web front-end development knowledge (HTML, CSS, Javascript)
- A Git client
- An activated and running ActiveServer instance
- Core programming knowledge of Java, PHP, C# or Go

Checkout the sample code

The 3DS Requestor demo code is hosted in GitHub and can be cloned from the following repository:

https://github.com/gpayments/gp-3ds-requestor-demo.git

To check out the sample code, execute the following command on your local environment:

```
git clone https://github.com/gpayments/gp-3ds-requestor-demo.git
```

Once the repository is cloned, you will find all the required demo code of this tutorial under the directory:

```
$ cd gp-3ds-requestor-demo
gp-3ds-requestor-demo $  ls
dotnet  go  java  php  README.md
```

GPayments currently provides backend example code for **Java**, **PHP**, **C#**, and **Go**. Throughout the integration guides, wherever the instructions differ between backend languages a tab will be shown with the relevant instructions.

All backend examples utilise the same front end code, which is written using **Javascript** and utilises **Mustache** for templating.

Tip

For simplicity and demo purposes, the demo 3DS Requestor code is implemented with most of the processes and page callback logic in the web frontend code. The backend code remains as a simple server side page forwarding/x509 authentication client to connect with the **ActiveServer** authentication API. In your actual implementation, you may design and implement your own 3DS Requestor code to fit your existing checkout process and work flow with different frontend and backend code structure.

Server Side Dependencies

For server side 3DS Requestor demo code, depending on different languages, the following dependencies and libraries are required to be installed prior to your first demo 3DS Requestor run:

Language	Dependencies and tools	Notes
Java	JDK 1.8 Apache Maven - https://maven.apache.org/install.html	
C#	Visual Studio 2013 or later with ASP.NET and web development workload **Nuget** is installed, refer to https://docs.microsoft.com/en-us/nuget/install-nuget-client-tools#visual-studio	Windows system required
PHP	PHP 7.2 with cURL (Client URL Library) Composer - https://getcomposer.org Guzzle - http://docs.guzzlephp.org OpenSSL	
Go	Go 1.12 with gin-gonic	Install Go 1.12 from https://golang.org/

Get client certificate

Before the backend can call the **ActiveServer** authentication API and authenticate itself, a **client certificate** is required to be setup. With this client certificate, the backend can setup a mutually authenticated TLS connection with **ActiveServer**.

To obtain the client certificate from your **ActiveServer** instance, refer here. If you do not have access to an instance and are using the **GPayments TestLabs**, email us at techsupport@gpayments.com to obtain a certificate.

Copy the downloaded certificate and configure the backend for the location you chose. The client certificate does not have to be in the demo 3DS Requestor directory, however you may find that storing the client certificate within the project may be easier for you to manage.

To check the directory structure of the project, refer to Directory Tree for details.

Demo 3DS Requestor Configuration

To run the demo 3DS Requestor, the following configuration needs to be set before booting up the system:

1. The 3DS Requestor needs to know the entry point of **ActiveServer** authentication API and the base url of itself so that it can tell **ActiveServer** which entry point of the 3DS Requestor to send the callback URLs.

   Configure the `AsAuthURL` and `BaseUrl` in your backend resource file. These values will be used when the application is brought up and cannot be changed without restarting the application:

   - **AsAuthURL:** set to the API URL of your **ActiveServer** instance. If you are using the **GPayments TestLabs** to test, leave the default value.
   - **BaseURL:** url that the application will be brought up and accessed on.

   Java C# PHP Go

   ```
   //application.yml
   server:
     port: 8082
   ...
   gpayments:
     asAuthUrl: https://api.as.testlab.3dsecure.cloud:7443
     baseUrl: http://localhost:8082
     certFileName: # Client Certificate file (.p12 or .pfx) path
     groupAuth: false
     merchantToken: # Your merchantToken when groupAuth = true
   ```

   Tip

   If the default port used in the examples is in use you may get an error, in which case configure the `port` and `baseUrl` to a suitable value.

2. Configure the `certFileName` in your backend resource file to your downloaded certificate:

   - If using a Merchant client certificate:
     - Configure the `certFileName` to the full filename and path of the downloaded certificate

    - Set `groupAuth` to `false`
    - Leave the `merchantToken` as blank
- If using a Master Auth API client certificate:
    - Configure the `certFileName` to the full filename and path of the downloaded certificate
    - Set `groupAuth` to `true`
    - Configure the `merchantToken`. Details of the merchant token can be found here

Warning

If `groupAuth = true`, the back-end needs to add a HTTP Header in the HTTP Request with a field of `AS-Merchant-Token` which is set to the merchantToken. To check the details of this implementation, refer to the Back-end implementation (v1) or Back-end implementation (v2) guides.

Java C# PHP Go

```
//application.yml
server:
  port: 8082
...
gpayments:
  asAuthUrl: https://api.as.testlab.3dsecure.cloud:7443
  baseUrl: http://localhost:8082
  certFileName: # Client Certificate file (.p12 or .pfx) path
  groupAuth: false
  merchantToken: # Your merchantToken when groupAuth = true
```

Note

You can place the client certificate in any location you prefer. For example, for a client certificate file located in the `C:/Downloads` directory (for windows platforms) with the file name `client_certificate.p12`, you should change the `certFileName` to be `C:/Downloads/client_certificate.p12`.

Note: For `PHP`, please follow the instruction here to extract the private key and certificate.

3. Configure the client certificate keystore password which was set during certificate download in your backend resource file:

   Java C# PHP Go

```
//RestClientConfig.java
private static final String KEYSTORE_PASSWORD = "123456";
private static final String KEY_ENTRY_PASSWORD = "123456";
private static final String CA_CERTS_FILE_NAME = "certs/cacerts.pem";
```

4. Open a **Terminal** (Linux, for Java, PHP or Go), **Command Prompt** (Windows, for Java, PHP or Go) or **Developer Command Prompt** (Windows, for C#) and execute the following command line on the root directory of the project. Note that this could take a few minutes the first time it is run if any dependencies are required to be download:

   Java C# PHP Go

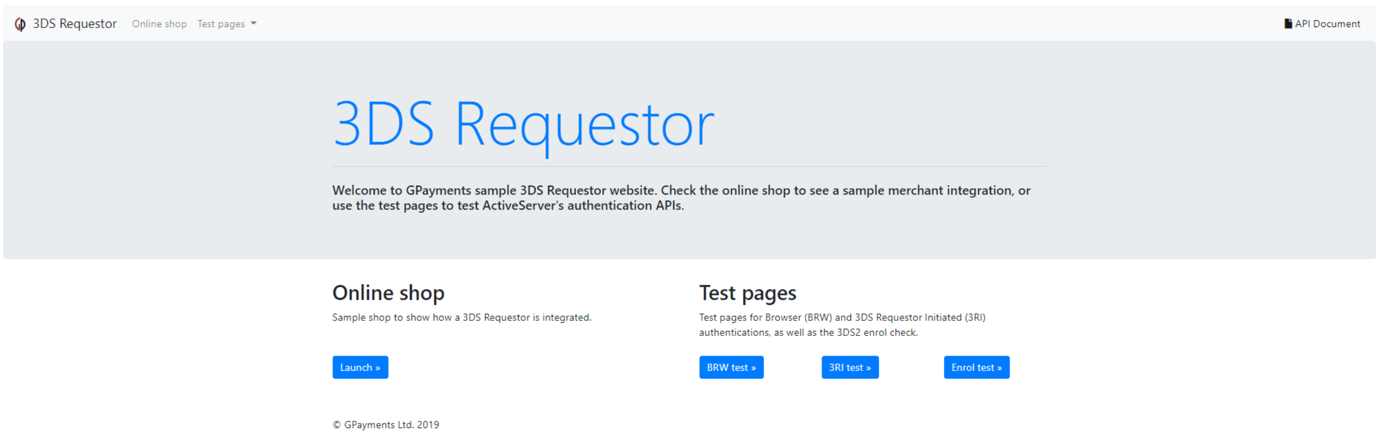
```
$ cd java
$ mvn spring-boot:run
```

Firewall warnings for Java Network access in Windows

In Windows you may encounter a Windows Firewall security alert regarding the Java network access, allow access to continue.

View 3DS Requestor website

Once the client certificate and the URLs are configured properly, and the startup command has been executed, the demo 3DS Requestor should be started up successfully. You can view the 3DS Requestor page by accessing http://localhost:8082 (or the BaseURL specified above):



3DS Requestor

Welcome to GPayments sample 3DS Requestor website. Check the online shop to see a sample merchant integration, or use the test pages to test ActiveServer's authentication APIs.

Online shop

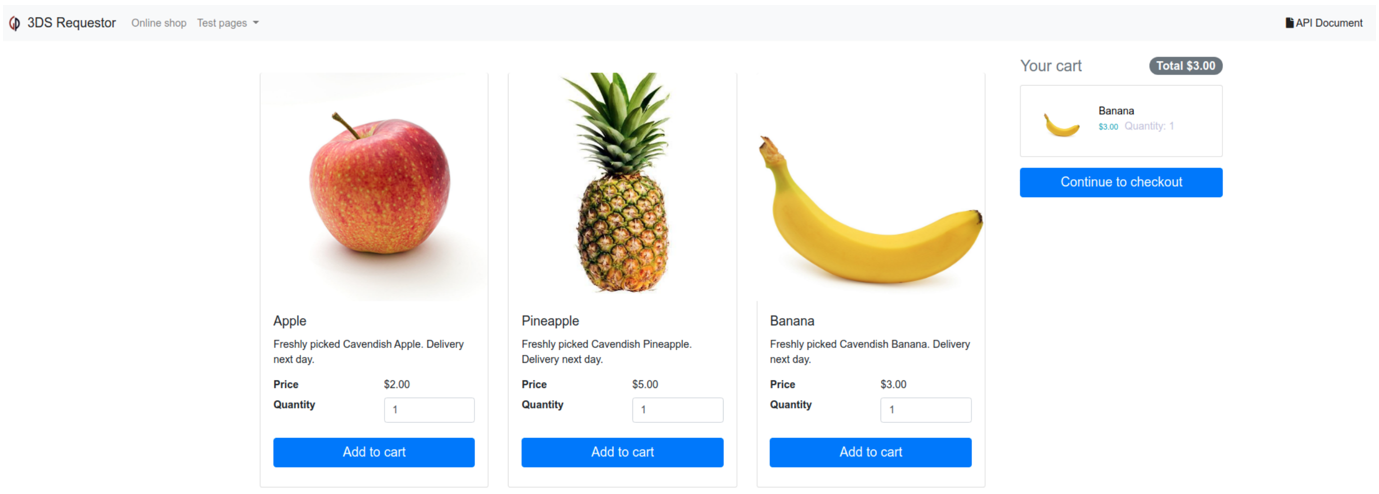
Sample shop to show how a 3DS Requestor is integrated.

Test pages

Test pages for Browser (BRW) and 3DS Requestor Initiated (3RI) authentications, as well as the 3DS2 enrol check.

© GPayments Ltd. 2019

Select the `Launch` button under `Online shop` to open the shop page. Try adding some items to the cart and checkout using the default cardholder information:



Banana $3.00 Quantity: 1

Apple

Freshly picked Cavendish Apple. Delivery next day.

Price $2.00

Quantity 1

Pineapple

Freshly picked Cavendish Pineapple. Delivery next day.

Price $5.00

Quantity 1

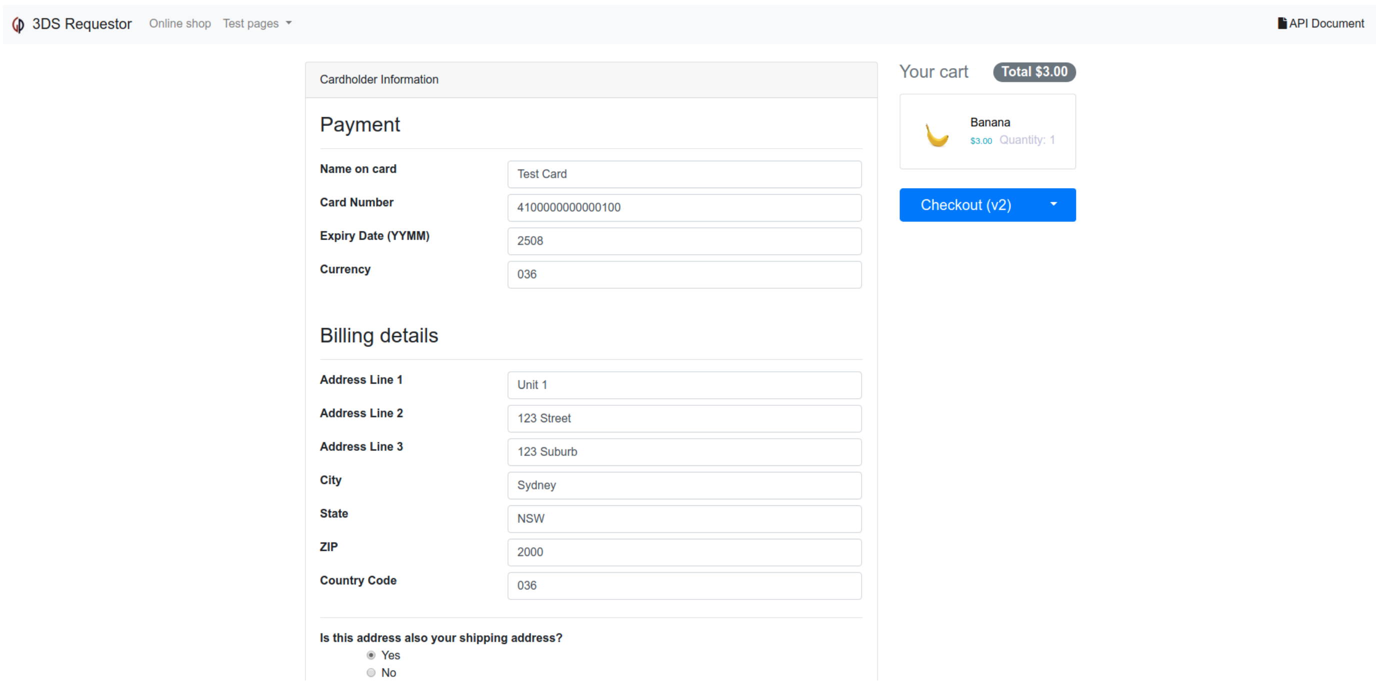
Banana

Freshly picked Cavendish Banana. Delivery next day.

Price $3.00

Quantity 1

The demo 3DS Requestor then shows the checkout page:



Banana $3.00 Quantity: 1

Cardholder Information

Payment

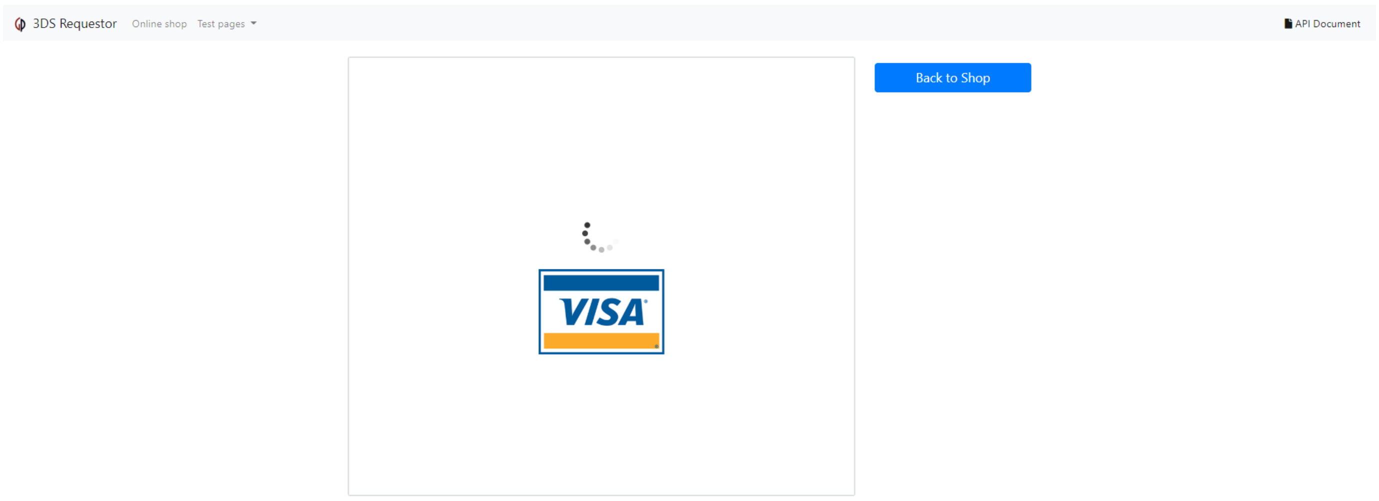
Name on card	Test Card
Card Number	4100000000000100
Expiry Date (YYMM)	2508
Currency	036

Billing details

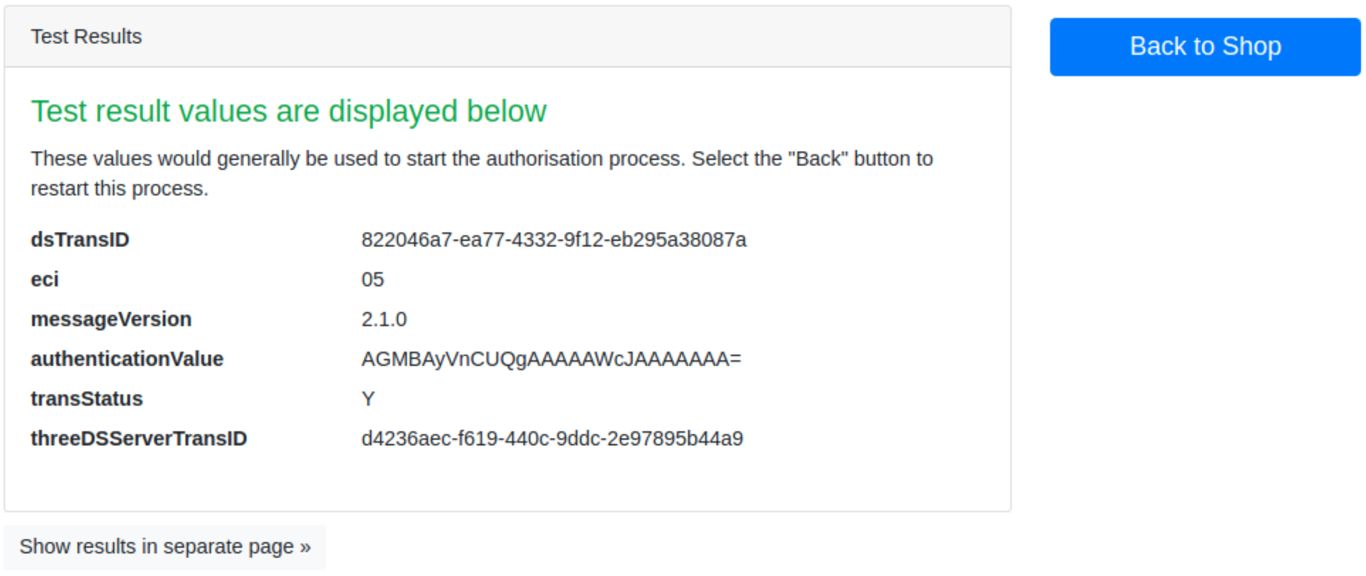
Address Line 1	Unit 1
Address Line 2	123 Street
Address Line 3	123 Suburb
City	Sydney
State	NSW
ZIP	2000
Country Code	036

Is this address also your shipping address?
[✓ Yes]
No

Proceed with the `Checkout` button, the demo 3DS Requestor will show the progress screen:



Then finally show the result page (note depending on the card number used, you may be prompted for a challenge screen, check Sample code feature for details):



Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

Show results in separate page »

Congratulations! You just successfully ran and tested your first demo 3DS Requestor locally.

Warning

You may notice that the demo 3DS Requestor is running on HTTP. This is just for demo purposes and this code is not suitable for production.

Support

If you have any questions after reading this document, we are here to help. Please email us at techsupport@gpayments.com.

Whats next?

Select **Next** to learn about the **v2 Front-end implementation** for a 3DS Requestor.

Front-end implementation (v2)

In this section, we will show the integration implementation for the front-end of the merchant application using our sample code based on the **Auth API version 2**.

Important

GPayments strongly recommends all new implementations of **ActiveServer** to integrate with version 2 of the Auth API from the beginning, as version 1 will be deprecated in a future release. Refer to the API migration from v1 to v2 guide for a summary of the changes and migration process from **v1** to **v2**. To check the existing **API v1** implementation, refer here.

The following files in the sample code package are essential for 3DS2 authentication in the front-end. Check the directory tree for details if required.

- `v2/process.html` : Implements all the authentication sequences.
- `v2/3ds-web-adapter.js` : The core component of the **3DS Client** that passes 3DS2 data from the front-end to the back-end and establishes the required iframes for callback URLs.
- `notify_3ds_events.html` : Call back page used for **ActiveServer** to trigger the next step in authentication (Step. 7 and Step. 18(C)). This page delivers authentication events to the checkout process.

The following sections detail the front-end implementation based on the authentication processes and authentication sequence.

Process 1: Initialise the Authentication

To initialise an authentication, the front-end needs to:

- Send an `Initialise authentication` message to the **3DS Requestor** (Step. 1 and Step. 2).
- Receive the response message and setup a callback `iframe` (Step. 5 and Step. 6).

When a user clicks the "**Checkout (v2)**" button in the checkout page, the browser stores the necessary data to session storage and goes to the `process.html` page.

```
//checkout.html
  function checkout() {
    var apiVersion = getApiVersion();
    switch (apiVersion) {
      ...
      case "v2":
        goApiV2();
        break;
      ...
    }
  }

  function goApiV2() {
    var sessionData = genSessionData();
    sessionData.authData.messageCategory = "pa";
    sessionStorage.setItem("sessionData", JSON.stringify(sessionData));
    window.location.href = "/v2/process";
  }

  function genSessionData() {
    var sessionData = {};
    sessionData.channel = "brw";
    sessionData.authData = genAuthData();
    sessionData.backButtonType = "toShop";
    return sessionData;
  }
```

Note

The `sessionData` contains:

- **`channel`** : `brw` or `3ri` . Here we used the `brw` channel for a browser-based authentication.
- **`authData`** : all the necessary data in JSON format. Note, there is a `messageCategory` in `authData` . The `messageCategory` is either `pa` - payment authentication or `npa` - non-payment authentication. Here we use `pa` as an example. For the data structure, refer to the API document.
- **`backButtonType`** : indicates the `back` button type in the process page. Here we make the `back` button as `Back to shop` .

NOTE: The use of `sessionStorage` for inter-page communications is just for demo purposes. The actual implementation of 3DS Requestor should choose the best approach for transferring parameters between pages for the integration with the existing checkout process.

In the `process.html` page, it receives the `sessionData` and starts the 3DS2 processes. Firstly, it sends information for initialising an authentication to the **3ds-web-adapter** (Step. 1).

```
//process.html
var sessionData = JSON.parse(sessionStorage.getItem("sessionData"));
...
switch (sessionData.channel) {
  case "brw":
    var container = $('#iframeDiv');
    brw(sessionData.authData, container, _callbackFn,
        sessionData.options);
    break;
  ...
```

Note

The `brw()` function in **3ds-web-adapter** takes the following parameters:

- `authData` : all the necessary data in JSON format. For data structure, refer to the API document.
- `container` : pre-defined container for the **3ds-web-adapter** to generate `iframes` .
- `callbackFn` : call back function to deal with the authentication results.
- `options` : optional parameters for 3DS Requestor. For example, the 3DS Requestor can choose to cancel the challenge by setting `options.cancelChallenge=true` .

Next, in the **3ds-web-adapter**, method `brw()` sends information to the 3DS Requestor back-end to initialise authentication (Step. 2).

```
//3ds-web-adapter.js
function brw(authData, container, callbackFn, options) {

  _callbackFn = callbackFn;
  iframeContainer = container;
  _authData = authData;

  if (options) {
    _options = options;
  }

  //generate an random number for iframe Id
  iframeId = String(Math.floor(100000 + Math.random() * 900000));

  //3DS Requestor url for Initialise Authentication
  var initAuthUrl = "/v2/auth/init";

  var initAuthData = {};
  initAuthData.acctNumber = authData.acctNumber;
  initAuthData.merchantId = authData.merchantId;

  console.log('init authentication', initAuthData);

  //Send data to /auth/init to do Initialise authentication (Step 2)
  doPost(initAuthUrl, initAuthData, _onInitAuthSuccess, _onError);
}
```

The `brw()` function makes a POST request to the **back-end** with the `initAuth` API message. The object is posted in JSON format. To check how the **back-end** handles this request, refer here.

The **3ds-web-adapter** uses the `_onInitAuthSuccess()` function to handle the successful response (Step. 5).

```
//3ds-web-adapter.js
function _onInitAuthSuccess(data) {
  console.log('init auth returns:', data);

  if (data.threeDSServerCallbackUrl && data.authUrl) {

    serverTransId = data.threeDSServerTransID;
    $('<iframe id="' + "3ds_" + iframeId
        + '" width="0" height="0" style="visibility: hidden;" src="'
        + data.threeDSServerCallbackUrl + '"></iframe>')
    .appendTo(iframeContainer);

    if (data.monUrl) {
      // optionally append the monitoring iframe
      $('<iframe id="' + "mon_" + iframeId
          + '" width="0" height="0" style="visibility: hidden;" src="'
          + data.monUrl + '"></iframe>')
      .appendTo(iframeContainer);
    }

  } else {
    _onError(data);
  }
}
```

The **3ds-web-adapter** inserts two hidden `iframes` into the checkout page (Step. 6). The first one is for (Step. 7) so that the browser information can be collected by the ACS and ActiveServer using the `threeDSServerCallbackUrl`.

The second one is an optional monitoring iframe to guarantee that an `InitAuthTimedOut` event will be returned by **ActiveServer** when any errors occur during browser info collecting or 3DS Method processing. This event has a timeout of 15 seconds.

Info

Step. 1 to Step. 7 of the sequence diagram are implemented by the end of this process.

Process 2: Execute Authentication

To execute authentication, the front-end needs to first finish the browser information collection and (if available), 3DS Method data collecting. After these 2 processes are done, `notify_3ds_event.html` will notify the `3ds-web-adapter` to continue with the authentication. In this process, if for any reason that the data collecting failed or was not able to finish, the separate monitoring iframe (setup in the InitAuth process) will notify the `3ds-web-adapter` with an event `InitAuthTimedOut`, and then the authentication process will be terminated.

The following are the steps to execute the authentication:

- Implement or re-use the provided `notify_3ds_events.html` to receive events about data collecting.
- Send an `Execute authentication` message to the **3DS Requestor** (Step. 8 and Step. 9) once events `_on3DSMethodSkipped` or `_on3DSMethodFinished` are notified.
- Handle the authentication result with frictionless flow or challenge flow (Step. 13, Step. 14(F) or Step. 14(C)).

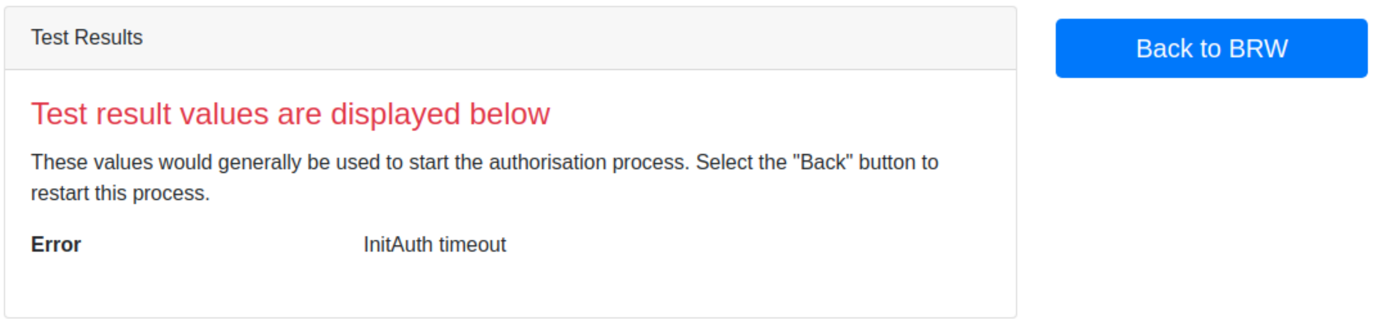
The `notify_3ds_events.html` is used to trigger the authentication process (Step. 8). The **3DS Requestor** will render `notify_3ds_events.html` with the variables `transId`, `callbackName` and `param`. To check the back-end implementation, refer here.

```
<!--notify_3ds_events.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"/>
  <title>3DSecure 2.0 Authentication</title>
</head>
<body>

<form>
  <input type="hidden" id="notifyCallback" name="notifyCallback"
value={{callbackName}}>
  <input type="hidden" id="transId" name="transId" value={{transId}}>
  <input type="hidden" id="param" name="param" value={{callbackParam}}>
</form>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script>
  //notify parent check out page to proceed with rest procedures.

  var callbackFn = parent[$('#notifyCallback').val()];
  //callbackFn is defined in 3ds-notify handler method
  if (typeof callbackFn === 'function') {
    callbackFn($('#transId').val(), $('#param').val());
  }
</script>

</body>
</html>
```

You can see that depending on the `callbackName`, it will call different methods in the **3ds-web-adapter** (Step. 8). The value of `callbackName` should be `_onThreeDSMethodFinished`, `_onThreeDSMethodSkipped`, `_onAuthResult` or `_onInitAuthTimedOut`. An explanation of each method is below:

Event	Description
`_onThreeDSMethodFinished`	Notifies that the 3DS method is finished by ACS and it's time to call `_doAuth()`

Event	Description
`_onThreeDSMethodSkipped`	Notifies that the 3DS method has been skipped (Not available or other reasons) and it's time to call `_doAuth()` , Note regardless of 3DS Method is skipped or not, the 3DS Server browser information collecting is always performed prior to 3DS Method.
`_onAuthResult`	This event notifies that the authentication result is available to fetch. Used in frictionless and challenge flow (Step. 17(F) & 19(C)).
`_onInitAuthTimedOut`	Notifies that errors occurred during 3DS Method or browser info collecting. In this demo, the authentication process will be terminated when this event occurs. The screenshot below shows the error result page:



Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

Error	InitAuth timeout

Here, at Step. 8, the `callbackName` returned by **ActiveServer** will be either `_onThreeDSMethodFinished` or `_onThreeDSMethodSkipped` . The **3ds-web-adapter** will call `_doAuth()` to make a POST request to execute authentication (Step. 9) once these events are received.

The backend then makes a call to the `auth` endpoint. To check how the **back-end** handles the request, refer here.

```
//3ds-web-adapter.js
function _doAuth(transId, param) {

  console.log('Do Authentication for transId', transId);

  //first remove any 3dsmethod iframe
  $("#3ds_" + iframeId).remove();

  var authData = _authData;
  //set the returned param to browserInfo
  authData.browserInfo = param;
  // authData.threeDSRequestorTransID = transId;
  authData.threeDSServerTransID = serverTransId;
  console.log("authData: ", authData);
  doPost("/v2/auth", authData, _onDoAuthSuccess, _onError);
}
```

Note

The `param` variable returned from **ActiveServer** contains the browser information, which should be set in the `authData` for **`auth`** API message.

For demo purposes, the browser information is set in the front end javascript (3ds-web-adapter). For security reasons this process may need to be implemented in the back end on the production environment.

The **3ds-web-adapter** uses `_onDoAuthSuccess()` function to handle the successful response (Step. 13).

```
//3ds-web-adapter.js
function _onDoAuthSuccess(data) {
  console.log('auth returns:', data);

  if (data.transStatus) {
    if (data.transStatus === "C") {
      // 3D requestor can decide whether to proceed the challenge here
      if (_options.cancelChallenge) {
        if (_options.cancelReason) {
          var sendData = {};
          sendData.threeDSServerTransID = serverTransId;
          sendData.status = _options.cancelReason;
          doPost("/v2/auth/challenge/status", sendData, _onCancelSuccess,
              _onCancelError)
        } else {
          var returnData = _cancelMessage();
          _callbackFn("onAuthResult", returnData);
        }
      } else {
        data.challengeUrl ? startChallenge(data.challengeUrl) : _onError(
            {"Error": "Invalid Challenge Callback Url"});
      }

    } else {
      _callbackFn("onAuthResult", data);
    }
  } else {
    _onError(data);
  }
}
```

It performs different flows based on the returned `transStatus`.

Note

The `transStatus` can be `Y`, `C`, `N`, `U`, `A`, and `R`

- `Y`: Authentication/Account Verification Successful
- `C`: Challenge/Additional authentication is required
- `N`: Not Authenticated/Account Not Verified
- `U`: Authentication/Account Verification Could Not Be Performed
- `A`: Attempts Processing Performed
- `R`: Authentication/Account Verification Rejected

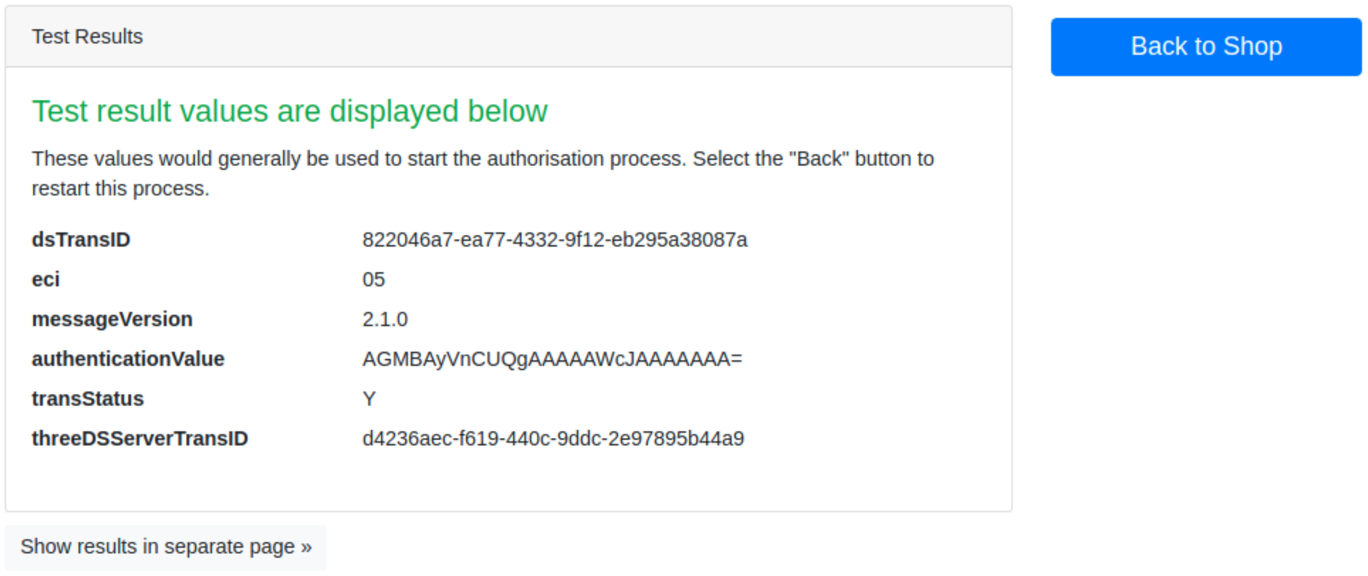
For more information related to `transStatus`, refer to the API document.

Frictionless authentication result

If the `transStatus` is not `C` (i.e. frictionless), the **3ds-web-adapter** will go to the frictionless flow (Step. 14(F)) and show the results using the `_callbackFn("onAuthResult", data)` (line 11).

```
//process.html
function _callbackFn(type, data) {

  switch (type) {
    case "onAuthResult":
      //display "Show results in separate page"
      $("#sepButton").removeClass("d-none");
      showResult(data);
      break;
```

The `showResult(data)` function will display the results in the `process.html` page:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

The frictionless authentication process stops at this point. The merchant checkout process can now move to the authorisation process as normal using the authentication result information.

Info

The authentication result can also be shown in a separate page. This is covered in process 3.

Continue challenge process

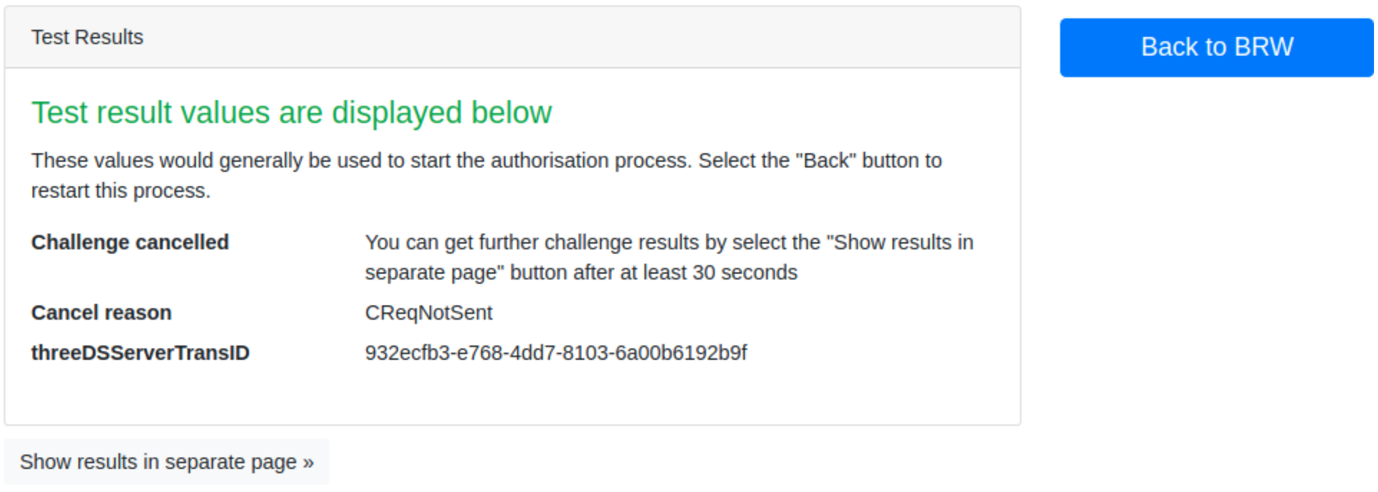
If the `transStatus` is `C`, a challenge is required from the ACS. It is up to the 3DS Requestor to decide whether it wants to continue the 3DS2 authentication and proceed to the challenge process or terminate the authentication process if a challenge is not desired. In this demo, the `options.cancelChallenge` parameter is used to indicate the 3DS Requestors decision on the challenge flow. This feature is outlined on the BRW Test Page.

Note

For demo purposes, the cancel challenge process is implemented in the front end javascript (`3ds-web-adapter`). For security reasons this process may need to be implemented in the back end on the production environment.

The **3ds-web-adapter** will check the `options.cancelChallenge` parameter. If `options.cancelChallenge=true`, the **3ds-web-adapter** will cancel the challenge. Optionally, `options.cancelReason` can also be set and sent to **ActiveServer** depending on the specified Cancel Reason. The specified reason will be sent by the **3ds-web-adapter** to the `challengeStatus` endpoint in the **back-end**. To check how the **back-end** handles this request, refer here.

The cancel challenge results screen should look similar to the screenshot below:



Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

Challenge cancelled	You can get further challenge results by select the "Show results in separate page" button after at least 30 seconds
Cancel reason	CReqNotSent
threeDSServerTransID	932ecfb3-e768-4dd7-8103-6a00b6192b9f

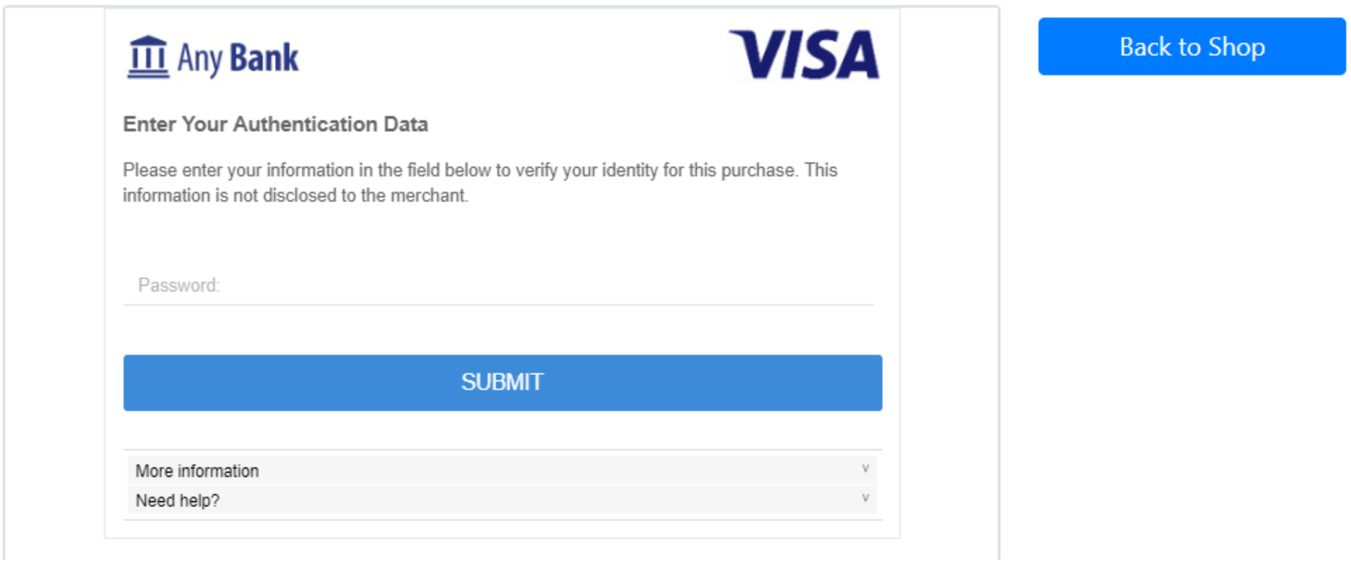
If `options.cancelChallenge=true` is not present (or the value is set to false), the **3ds-web-adapter** will call `startChallenge()` , which inserts an `iframe` for the challenge window with `src` set to `challengeUrl` (Step. 14(C)) in order to show the challenge screen to the card holder.

```
//3ds-web-adapter.js
function startChallenge(url) {

  _callbackFn("onChallengeStart");
  //create the iframe
  $('<iframe src="' + url
      + '" width="100%" height="100%" style="border:0" id="' + "cha_" + iframeId
      + '"></iframe>')
  .appendTo(iframeContainer);

}
```

Note

If you want to test the challenge scenario, follow the guide here.

You can see that the `iframe` class has been set to `width="100%"` and `height="100%"` . This is required because the `iframe` needs to resize according to the content provided by the ACS. The challenge screen should look similar to the screenshot below:


Enter Your Authentication Data

Please enter your information in the field below to verify your identity for this purchase. This information is not disclosed to the merchant.

The cardholder can now input the authentication data such as an OTP password and submit the challenge form. The ACS will authenticate the transaction and the card holder and return the challenge result.

After the challenge flow is finished, **ActiveServer** will call the `3ds-notify` entry point in the **3DS Requestor** backend with event `_onAuthResult` and then the **3DS Requestor** will render the `notify_3ds_events.html` page, similar to before (Step. 19(C)). The **3ds-web-adapter** calls the `_onAuthResult()` function to get the authentication results and displays them on the `process.html` page.

Info

In a production environment the ACS will perform complex risk-based authentication from the information obtained about the cardholder. Similarly, the authentication method (e.g. OTP, biometrics etc) will be determined and implemented by the cardholder's issuing bank.

Process 3: Get Authentication Result

After the challenge process is finished (or if the frictionless process results need to be shown in a separate page), we now need to request for the authentication result so that it can be used for the authorisation process.

Why a separate result request is necessary?

You may wonder why you need to request the result again since you already have the result of authentication available after process 2.

This is because the authentication result is returned to the authentication page in Step. 13 by the 3DS Requestor. It is common that the authentication result page is shown as a separate page from the checkout page. In this case, the **3ds-web-adapter** could transfer the result back to the 3DS Requestor or the result page, however, it is not a recommended data flow as the authentication result is re-transferred by the client side code and is in general considered as insecure.

The server-side of the 3DS Requestor should always have its own mechanism to get the result back from the origin source: **ActiveServer**. Instead of transferring the result to the new result page, the 3DS Requestor server-side can provide a result page that shows the authentication result. Therefore, you need to request for a result receipt in this step.

Here in this demo, you can select `Show results in separate page >>` at the end of process 2 in the `process.html` page to show the result in a separate page.

To get the authentication result and show it in a separate page, the front-end needs to:

- Send a `Request for authentication result` message to the **3DS Requestor** (Step. 15(F) or Step. 20(C)).
- Show result on screen (Step. 17(F) or Step. 22(C)).

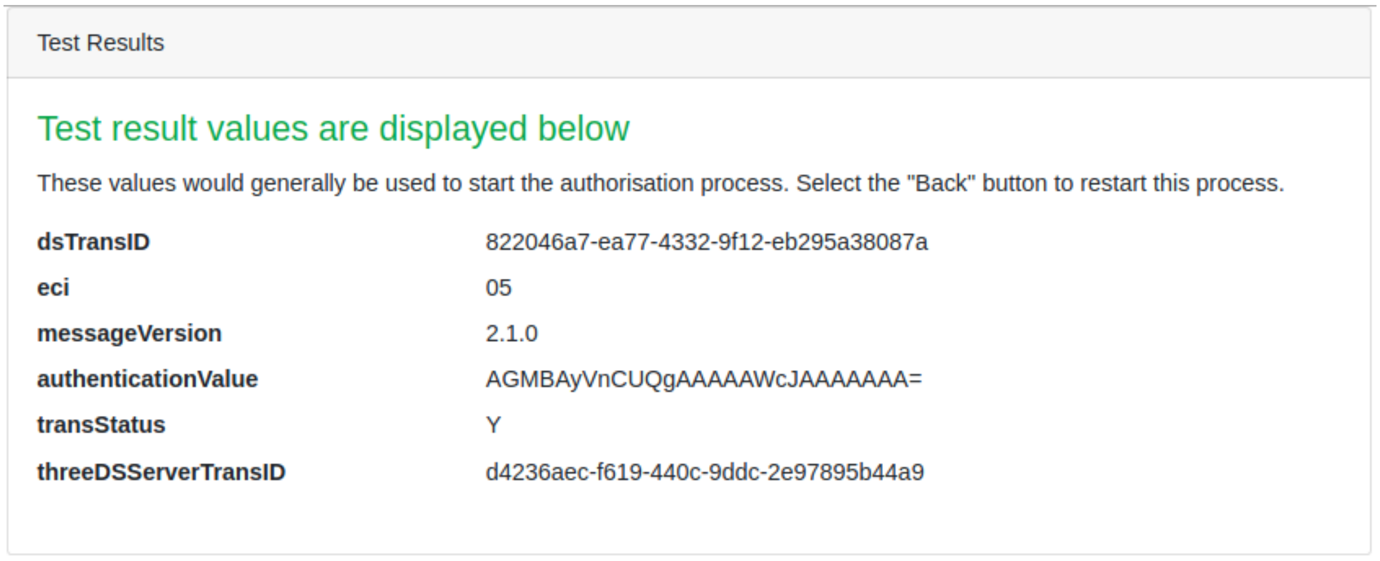
Firstly, in the `process.html` page, it will store the `serverTransId` into `sessionStorage` and go to `result.html` page.

```
//process.html
function openResultInNewWindow() {
if (serverTransId) {
  var url = '/v2/result';

  sessionStorage.setItem("serverTransId", serverTransId);
  window.open(url, 'newwindow', 'height=800,width=1000');
}
}
```

In the `v2/result.html` page, it call the `result()` method in **3ds-web-adapter** to get the authentication result. The `result()` method sends a `Request for authentication result` message to the backend. The backend will receive this request and call the **`result`** endpoint. To check how the **back-end** handles this request, refer here. The callback function `showData()` shows the results in the page.

```
//result.html
var serverTransId = sessionStorage.getItem("serverTransId");

//request for authentication result (Step 15(F) or Step 20(C))
result(serverTransId, showData);

//show result in separate page (Step 17(F) or Step 22(C))
function showData(type, data) {
    var toShow = "<dl class='row'>";
    Object.keys(data).forEach(function (k) {
      toShow = toShow + "<dt class='col-sm-4'>" + k + "</dt>" + "<dd class='col-
sm-8'>" + data[k]
          + "</dd>";
    });
    toShow += "</dl>";
    $('#showResult').empty().append(toShow);
    $("#resultCard").removeClass("d-none");
}
```

The new result screen will look like the screenshot below:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

✓ **Success**

This covers the front-end integration. After authentication is completed the checkout process can proceed with the authorisation process using the Transaction Status, ECI and CAVV to complete the transaction.

Whats next?

Select **Next** to learn about the **v2 Back-end implementation** for a 3DS Requestor.

Back-end implementation (v2)

In this section, we will show the implementation details for the back-end side of the merchant application using our sample code based on the **Auth API version 2**.

Important

GPayments strongly recommends all new implementations of **ActiveServer** to integrate with version 2 of the Auth API from the beginning, as version 1 will be deprecated in a future release. Refer to the API migration from v1 to v2 guide for a summary of the changes and migration process from **v1** to **v2**. To check the existing **API v1** implementation, refer here.

For the back-end, we need to implement a **3DS Requestor**. The **3DS Requestor** receives information from the **3ds-web-adapter** and sends the requests to **ActiveServer**. It also receives the authentication results from **ActiveServer** and forwards the results to the **3ds-web-adapter**.

The demo **3DS Requestor** code provides the backend implementation with the following server side languages:

- **Java**: The java version is implemented based on the Springboot framework. For details of Springboot, check out https://spring.io/projects/spring-boot
- **C#**: The C# version is implemented based on ASP.net.
- **PHP**: The PHP version is implemented based on PHP 7.2 with cURL (Client URL Library).
- **Go**: The Go version is implemented based on Go 1.12 with Go Module support. All dependencies are listed in `go.mod` file.

Why we need to implement a backend?

As defined by EMVCo's 3DSecure 2.0 specification, when the 3DS Server and the 3DS Requestor environment is separated, the communication between these two components must be mutual authenticated:

> [Req 300] If the 3DS Requestor and 3DS Server are separate components, ensure that data transferred between the components is protected at a level that satisfies Payment System security requirements with mutual authentication of both servers.

Before starting the authentication process with **ActiveServer**, the **3DS Requestor** needs to establish a mutual TLS connection with **ActiveServer**. Make sure you have the **client certificate** and configure it with the **3DS Requestor**. If not, follow the Get client certificate and Configure 3DS Requestor details instructions found on the Introduction page.

Tip

The implementation of TLS configuration for the HTTP Client can be found as follows:

- Java: The TLS configuration and client certificate loading can be found in class `RestClientConfig.java` .
- C#: The TLS configuration and client certificate loading can be found in class `RestClientHelper.cs` .
- PHP: The TLS configuration and client certificate loading can be found in file `RestClientConfig.php` .
- Go: The TLS configuration and client certificate loading can be found in file `https.go` .

Next, we will describe the details of the back-end implementation based on the authentication processes and authentication sequence.

Process 1: Initialise the Authentication

To initialise authentication, the **3DS Requestor** needs to:

- Receive the `Initialise authentication` request from the **3DS-web-adapter** (Step. 2).
- Forward the request to **ActiveServer** (Step. 3).
- Receive the response data from **ActiveServer** (Step. 4).
- Return the response data to the **3DS-web-adapter** (Step. 5).

Java C# PHP Go

```
//AuthControllerV2.java
/**
 * Receives the initialise authentication request from the 3DS-web-adapter
(Step 2)
 * Send data to ActiveServer to Initialise Authentication
 */
  @PostMapping("/v2/auth/init")
  public Message initAuth(@RequestBody Message request, HttpSession session) {

    //Generate requestor trans ID
    String transId = UUID.randomUUID().toString();
    request.put("threeDSRequestorTransID", transId);
    //Fill the event call back url with requestor url + /3ds-notify
    String callBackUrl = config.getBaseUrl() + "/3ds-notify";
    request.put("eventCallbackUrl", callBackUrl);

    //ActiveServer url for Initialise Authentication
    String initAuthUrl = config.getAsAuthUrl() + "/api/v2/auth/brw/init";
    logger.info("initAuthRequest on url: {}, body: \n{}", initAuthUrl, request);

    //Send data to ActiveServer to Initialise authentication (Step 3)
    //Get the response data from ActiveServer (Step 4)
    Message response =
        sendRequest(initAuthUrl, request, HttpMethod.POST);
    logger.info("initAuthResponseBRW: \n{}", response);

    if (response != null) {
      //save initAuth response into session storage
      session.setAttribute((String) response.get("threeDSServerTransID"),
          response);
    } else {
      logger.error("Error in initAuth response");
    }

    //Return data to 3ds-web-adapter (Step 5)
    return response;
  }
```

Note

We set the `eventCallbackUrl` to `{baseUrl}/3ds-notify`. This allows **ActiveServer** to make a notification when the browser information collection (Step. 7) is done. The `baseUrl` is set here.

The `initAuth` request will be sent to `{ActiveServer auth url}/api/v2/auth/brw/init`. To check the data structure of `initAuth` request, refer to the API document.

We save the **`initAuth`** response message to session for future use. To check the data structure of `initAuth` response, refer to the API document.

HTTP header for Master Auth API client certificate

If you are using a Master Auth API client certificate to authenticate a **Business Admin** user on behalf of a merchant, the back-end needs to add a HTTP Header in the HTTP Request with a field of `AS-Merchant-Token`, which should be set to the merchantToken from the merchants profile.

Java C# PHP Go

```java
//Note: the sendRequest() function will send the request to ActiveServer.
//If this is groupAuth, the request should include a HTTP Header with the field
of AS-Merchant-Token.
private Message sendRequest(String url, Message request, HttpMethod method) {

    HttpEntity<Message> req;
    HttpHeaders headers = null;

    if (config.isGroupAuth()) {
      //the certificate is for groupAuth, work out the header.
      headers = new HttpHeaders();
      headers.add("AS-Merchant-Token", config.getMerchantToken());
    }

    switch (method) {
      case POST:
        req = new HttpEntity<>(request, headers);
        return restTemplate.postForObject(url, req, Message.class);
      case GET:
        if (headers == null) {
          return restTemplate.getForObject(url, Message.class);
        } else {
          req = new HttpEntity<>(headers);
          return restTemplate.exchange(url, HttpMethod.GET, req,
Message.class).getBody();
        }
      default:
        return null;
    }
  }
}
```

Process 2: Execute Authentication

To execute authentication, the **3DS Requestor** needs to:

- Handle the `/3ds-notify` message from **ActiveServer** called through the iframe after Step. 7.
- Handle the `Execute authentication` request from the **3DS-web-adapter** (Step. 9 and Step. 10).

- Receive the authentication result and return it to the **3DS-web-adapter** (Step. 12 and Step. 13).

When the browser information collection (Step. 7) is done, **ActiveServer** makes a notification to the `eventCallbackUrl` which we set to `http://localhost:8082/3ds-notify`. The **3DS Requestor** handles this notification and passes the required parameters to the `notify-3ds-events.html` page.

Java C# PHP Go

```java
//MainController.java
@PostMapping("/3ds-notify")
public String notifyResult(
    @RequestParam("requestorTransId") String transId,
    @RequestParam("event") String callbackType,
    @RequestParam(name = "param", required = false) String param,
    Model model) {

  String callbackName;
  if ("3DSMethodFinished".equals(callbackType)) {

    callbackName = "_on3DSMethodFinished";

  } else if ("3DSMethodSkipped".equals(callbackType)) {

    callbackName = "_on3DSMethodSkipped";

  } else if ("AuthResultReady".equals(callbackType)) {
    callbackName = "_onAuthResult";
  } else {
    throw new IllegalArgumentException("invalid callback type");
  }

  model.addAttribute("transId", transId);
  model.addAttribute("callbackName", callbackName);
  model.addAttribute("callbackParam", param);

  return "notify_3ds_events";
}
```

Note

This handler is called by **ActiveServer**, parameters `requestorTransId`, `event` and an optional `param` will be provided by **ActiveServer**. The `event` can be `3DSMethodFinished`, `3DSMethodSkipped`, `InitAuthTimedOut` or `AuthResultReady`. For descriptions of each event refer to our API document for the field `eventCallbackUrl`. The handler method then sets the proper attribute in the page context and returns to the `notify_3ds_events.html` page. The page then renders the content with Mustache templating engine.

To check the front-end implementation of `notify_3ds_events.html`, refer here.

Then when the 3DS client is finished browser information collecting, it will call the `auth` endpoint to start the authentication. The **3DS Requestor** handles the `Execute authentication` requests (Step. 9, Step. 10, Step. 12, and Step. 13).

Java C# PHP Go

```java
//AuthControllerV2.java
/**
 * Receives the Execute authentication request from the 3DS-web-adapter (Step 9)
 * Send data to ActiveServer to Execute Authentication
 */
@PostMapping("/v2/auth")
public Message auth(@RequestBody Message request, HttpSession session) {
    //get authUrl from session storage
    Message initAuthResponse = (Message) session
        .getAttribute((String) request.get("threeDSServerTransID"));
    String authUrl = (String) initAuthResponse.get("authUrl");
    logger.info("requesting BRW Auth API {}, body: \n{}", authUrl, request);

    //Send data to ActiveServer to Execute Authentication (Step 10)
    //Get the response data from ActiveServer (Step 12)
    Message response =
        sendRequest(authUrl, request, HttpMethod.POST);
    logger.info("authResponseBRW: \n{}", response);

    //Return data to 3ds-web-adapter (Step 13)
    return response;
}
```

Note

The `authUrl` is the url to perform authentication which is defined in `initAuth` response message. The `initAuth` response message is stored in session when received in process 1.

The **3DS Requestor** returns a response to the front-end. The returned message contains a `transStatus` with value `Y` or `C`, which will trigger either frictionless flow or challenge flow. To check how the front-end handles the `transStatus`, refer here. To check the structure of the response data, refer to the API document.

Cancel challenge flow

When the `transStatus=C`, the 3DS client can choose to start the challenge or not. If the 3DS client chooses to cancel the challenge, it may call the `challengeStatus` endpoint to specify the Cancel Reason which the **3DS Requestor** will send to **ActiveServer**. To check how the **front-end** handles this request, refer here.

Java C# PHP Go

```java
//AuthControllerV2.java
@PostMapping("/v2/auth/challenge/status")
public Message challengeStatus(@RequestBody Message request) {

    String challengeStatusUrl = config.getAsAuthUrl() + "/api/v2/auth/challenge/
status";
    logger.info("request challenge status API {}, body: \n{}",
challengeStatusUrl, request);

    Message response =
        sendRequest(challengeStatusUrl, request, HttpMethod.POST);
    logger.info("challengeStatus response: \n{}", response);

    return response;
}
```

Process 3: Get Authentication Result

To get the authentication result, the **3DS Requestor** needs to:

- Handle the `Request for authentication result` request from the **3DS-web-adapter** (Step. 15(F) or Step. 20(C)).
- Send a request to **ActiveServer** to get the result and return the result to the front-end. (Step. 16(F), Step. 17(F) or Step. 21(C), Step. 22(C)).

Java C# PHP Go

```java
//AuthControllerV2.java
/**
 * Receives the Request for authentication result request (Step 15(F) and Step
20(C))
 * Send data to ActiveServer to Retrieve Authentication Results
 */
@GetMapping("/v2/auth/result")
public Message result(@RequestParam("txid") String serverTransId) {

    //ActiveServer url for Retrieve Results
    String resultUrl = config.getAsAuthUrl() +
        "/api/v2/auth/brw/result?threeDSServerTransID=" +
        serverTransId;

    //Get authentication result from ActiveServer (Step 16(F) and Step 21(C))
    Message response =
        sendRequest(resultUrl, null, HttpMethod.GET);
    logger.info("authResponse: \n{}", response);
    //Show authentication results on result.html (Step 17(F) and Step 22(C))
    return response;
}
```

✓ **Success**

This covers the backend implementation. After authentication is completed the checkout process can move on to performing authorisation using the Transaction Status, ECI and CAVV to complete the transaction.

Whats next?

If implementing v2 of the Auth API, skip to the **Sample code features** page to view all the features of our **Sample Code** for the **GPayments 3DS Requestor**. Otherwise, select **Next** to learn about the **v1 Front-end implementation** for a 3DS Requestor.

Front-end implementation (v1)

In this section, we will show the integration implementation for the front-end of the merchant application using our sample code based on the **Auth API version 1**.

Important

GPayments strongly recommends all new implementations of **ActiveServer** to integrate with version 2 of the Auth API from the beginning, as version 1 will be deprecated in a future release. Refer to the API migration from v1 to v2 guide for a summary of the changes and migration process from **v1** to **v2**. To check the new **API v2** implementation, refer here.

The following files in the sample code package are essential for 3DS2 authentication in the front-end. Check the directory tree for details if required.

- `v1/process.html` : Implements all the authentication sequences.
- `v1/3ds-web-adapter.js` : The core component of the **3DS Client** that passes 3DS2 data from the front-end to the back-end and establishes the required iframes for callback URLs.
- `notify_3ds_events.html` : Call back page used for **ActiveServer** to trigger the next step in authentication (Step. 7 and Step. 18(C)). This page delivers authentication events to the checkout process.

The following sections detail the front-end implementation based on the authentication processes and authentication sequence.

Process 1: Initialise the Authentication

To initialise an authentication, the front-end needs to:

- Send an `Initialise authentication` message to the **3DS Requestor** (Step. 1 and Step. 2).
- Receive the response message and setup a callback `iframe` (Step. 5 and Step. 6).

When a user clicks the "**Checkout (v1)**" button in the checkout page, the browser stores the necessary data to session storage and goes to the `process.html` page.

```
//checkout.html
  function checkout() {
    var apiVersion = getApiVersion();
    switch (apiVersion) {
      case "v1":
        goApiV1();
        break;
       ...
    }
  }

  function goApiV1() {
    var sessionData = genSessionData();
    sessionData.messageCategory = "pa";
    sessionStorage.setItem("sessionData", JSON.stringify(sessionData));
    window.location.href = "/v1/process";
  }

  function genSessionData() {
    var sessionData = {};
    sessionData.channel = "brw";
    sessionData.authData = genAuthData();
    sessionData.backButtonType = "toShop";
    return sessionData;
  }
```

Note

The `sessionData` contains:

- **`channel`** : `brw` or `3ri` . Here we used the `brw` channel for a browser-based authentication.
- **`messageCategory`** : either `pa` - payment authentication or `npa` - non-payment authentication. Here we use `pa` as an example.
- **`authData`** : all the necessary data in JSON format. For the data structure, refer to the API document.
- **`backButtonType`** : indicates the `back` button type in the process page. Here we make the `back` button as `Back to shop` .

NOTE: The use of `sessionStorage` for inter-page communications is just for demo purposes. The actual implementation of 3DS Requestor should choose the best approach for transferring parameters between pages for the integration with the existing checkout process.

In the `process.html` page, it receives the `sessionData` and starts the 3DS2 processes. Firstly, it sends information for initialising an authentication to the **3ds-web-adapter** (Step. 1).

```
//process.html
var sessionData = JSON.parse(sessionStorage.getItem("sessionData"));
...
switch (sessionData.channel) {
  case "brw":
    var container = $('#iframeDiv');
    brw(sessionData.authData, container, _callbackFn,
sessionData.messageCategory,
      sessionData.options);
    break;
  ...
```

Note

The `brw()` function in **3ds-web-adapter** takes the following parameters:

- `authData` : all the necessary data in JSON format. For data structure, refer to the API document.
- `container` : pre-defined container for the **3ds-web-adapter** to generate `iframes` .
- `callbackFn` : call back function to deal with the authentication results.
- `messageCategory` : either `pa` - payment authentication or `npa` - non-payment authentication.
- `options` : optional parameters for 3DS Requestor. For example, the 3DS Requestor can choose to cancel the challenge by setting `options.cancelChallenge=true` .

Next, in the **3ds-web-adapter**, method `brw()` sends information to the 3DS Requestor back-end to initialise authentication (Step. 2).

```
//3ds-web-adapter.js
function brw(authData, container, callbackFn, messageCategory, options) {

  _callbackFn = callbackFn;
  iframeContainer = container;
  if (options) {
    _options = options;
  }
  //generate an random number for iframe Id
  iframeId = String(Math.floor(100000 + Math.random() * 900000));

  //3DS Requestor url for Initialise Authentication
  var initAuthUrl;
  if (messageCategory) {
    if (messageCategory === "pa" || messageCategory === "npa") {
      initAuthUrl = "/v1/auth/init/" + messageCategory;
    } else {
      _onError({"Error": "Invalid messageCategory"});
    }
  } else {
    initAuthUrl = "/v1/auth/init/" + "pa";
  }

  console.log('init authentication', authData);

  //Send data to /auth/init/{messageCategory} to do Initialise authentication
(Step 2)
  doPost(initAuthUrl, authData, _onInitAuthSuccess, _onError);
}
```

The `brw()` function makes a POST request to the **back-end** with the `initAuth` API message. The object is posted in JSON format. To check how the **back-end** handles this request, refer here.

The **3ds-web-adapter** uses the `_onInitAuthSuccess()` function to handle the successful response (Step. 5).

```
//3ds-web-adapter.js
function _onInitAuthSuccess(data) {
  console.log('init auth returns:', data);

  if (data.threeDSServerCallbackUrl) {

    serverTransId = data.threeDSServerTransID;
    $('<iframe id="' + "3ds_" + iframeId
        + '" width="0" height="0" style="visibility: hidden;" src="'
        + data.threeDSServerCallbackUrl + '"></iframe>')
    .appendTo(iframeContainer);

    if (data.monUrl) {
      // optionally append the monitoring iframe
      $('<iframe id="' + "mon_" + iframeId
          + '" width="0" height="0" style="visibility: hidden;" src="'
          + data.monUrl + '"></iframe>')
      .appendTo(iframeContainer);
    }
  } else {
    _onError(data);
  }
}
```

The **3ds-web-adapter** inserts two hidden `iframes` into the checkout page (Step. 6). The first one is for (Step. 7) so that the browser information can be collected by the ACS and ActiveServer using the `threeDSServerCallbackUrl`.

The second one is an optional monitoring iframe to guarantee that an `InitAuthTimedOut` event will be returned by **ActiveServer** when any errors occur during browser info collecting or 3DS Method processing. This event has a timeout of 15 seconds.

Info

Step. 1 to Step. 7 of the sequence diagram are implemented by the end of this process.

Process 2: Execute Authentication

To execute authentication, the front-end needs to first finish the browser information collection and (if available), 3DS Method data collecting. After these 2 processes are done, `notify_3ds_event.html` will notify the `3ds-web-adapter` to continue with the authentication. In this process, if for any reason that the data collecting failed or was not able to finish, the separate monitoring iframe (setup in the InitAuth process) will notify the `3ds-web-adapter` with an event `InitAuthTimedOut` , and then the authentication process will be terminated.

The following are the steps to execute the authentication:

- Implement or re-use the provided `notify_3ds_events.html` to receive events about data collecting.
- Send an `Execute authentication` message to the **3DS Requestor** (Step. 8 and Step. 9) once events `_on3DSMethodSkipped` or `_on3DSMethodFinished` are notified.
- Handle the authentication result with frictionless flow or challenge flow (Step. 13, Step. 14(F) or Step. 14(C)).

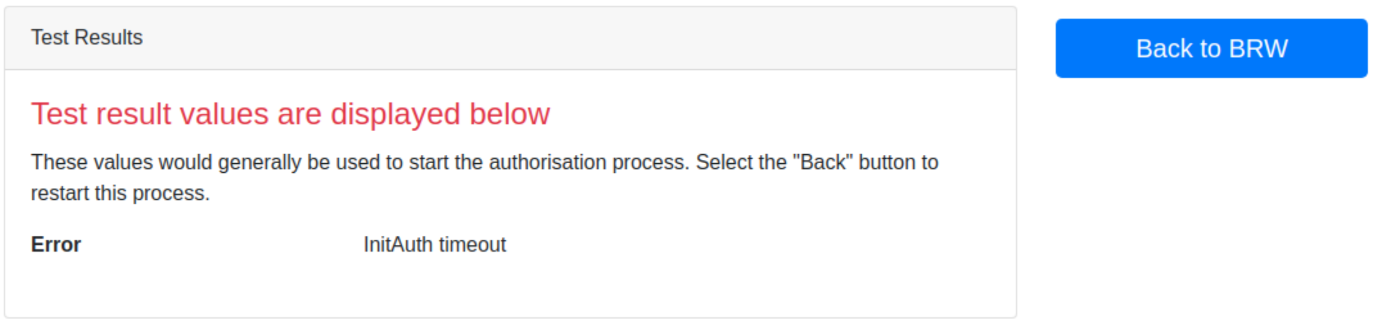
The `notify_3ds_events.html` is used to trigger the authentication process (Step. 8). The **3DS Requestor** will render `notify_3ds_events.html` with the variables `transId` , `callbackName` and an optional `param` . To check the back-end implementation, refer here.

```
<!--notify_3ds_events.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"/>
  <title>3DSecure 2.0 Authentication</title>
</head>
<body>

<form>
  <input type="hidden" id="notifyCallback" name="notifyCallback"
value={{callbackName}}>
  <input type="hidden" id="transId" name="transId" value={{transId}}>
  <input type="hidden" id="param" name="param" value={{callbackParam}}>
</form>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script>
  //notify parent check out page to proceed with rest procedures.

  var callbackFn = parent[$('#notifyCallback').val()];
  //callbackFn is defined in 3ds-notify handler method
  if (typeof callbackFn === 'function') {
    callbackFn($('#transId').val(), $('#param').val());
  }
</script>

</body>
</html>
```

You can see that depending on the `callbackName`, it will call different methods in the **3ds-web-adapter** (Step. 8). The value of `callbackName` should be `_onThreeDSMethodFinished`, `_onThreeDSMethodSkipped`, `_onAuthResult` or `_onInitAuthTimedOut`. An explanation of each method is below:

Event	Description
`_onThreeDSMethodFinished`	Notifies that the 3DS method is finished by ACS and it's time to call `_doAuth()`

Event	Description
`_onThreeDSMethodSkipped`	Notifies that the 3DS method has been skipped (Not available or other reasons) and it's time to call `_doAuth()` , Note regardless of 3DS Method is skipped or not, the 3DS Server browser information collecting is always performed prior to 3DS Method.
`_onAuthResult`	This event notifies that the authentication result is available to fetch. Used in frictionless and challenge flow (Step. 17(F) & 19(C)).
`_onInitAuthTimedOut`	Notifies that errors occurred during 3DS Method or browser info collecting. In this demo, the authentication process will be terminated when this event occurs. The screenshot below shows the error result page:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

Error InitAuth timeout

Here, at Step. 8, the `callbackName` returned by **ActiveServer** will be either `_onThreeDSMethodFinished` or `_onThreeDSMethodSkipped` . The **3ds-web-adapter** will call `_doAuth()` to make a POST request to execute authentication (Step. 9) once these events are received.

The backend then makes a call to the `auth` endpoint. To check how the **back-end** handles the request, refer here.

```
//3ds-web-adapter.js
function _doAuth(transId) {

  console.log('Do Authentication for transId', transId);

  //first remove any 3dsmethod iframe
  $("#3ds_" + iframeId).remove();
  var authData = {};
  authData.threeDSRequestorTransID = transId;
  authData.threeDSServerTransID = serverTransId;

  doPost("/v1/auth", authData, _onDoAuthSuccess, _onError);
}
```

The **3ds-web-adapter** uses `_onDoAuthSuccess()` function to handle the successful response (Step. 13).

```
//3ds-web-adapter.js
function _onDoAuthSuccess(data) {
  console.log('auth returns:', data);

  if (data.transStatus) {
    if (data.transStatus === "C") {
      // 3D requestor can decide whether to proceed the challenge here
      if (_options.cancelChallenge) {
        if (_options.cancelReason) {
          var sendData = {};
          sendData.threeDSServerTransID = serverTransId;
          sendData.status = _options.cancelReason;
          doPost("/v1/auth/challenge/status", sendData, _onCancelSuccess,
              _onCancelError)
        } else {
          var returnData = _cancelMessage();
          _callbackFn("onAuthResult", returnData);
        }
      } else {
        data.challengeUrl ? startChallenge(data.challengeUrl) : _onError(
            {"Error": "Invalid Challenge Callback Url"});
      }

    } else {
      _callbackFn("onAuthResult", data);
    }
  } else {
    _onError(data);
  }
}
```

It performs different flows based on the returned `transStatus` .

Note

The `transStatus` can be `Y`, `C`, `N`, `U`, `A`, and `R`

- `Y`: Authentication/Account Verification Successful
- `C`: Challenge/Additional authentication is required
- `N`: Not Authenticated/Account Not Verified
- `U`: Authentication/Account Verification Could Not Be Performed
- `A`: Attempts Processing Performed
- `R`: Authentication/Account Verification Rejected

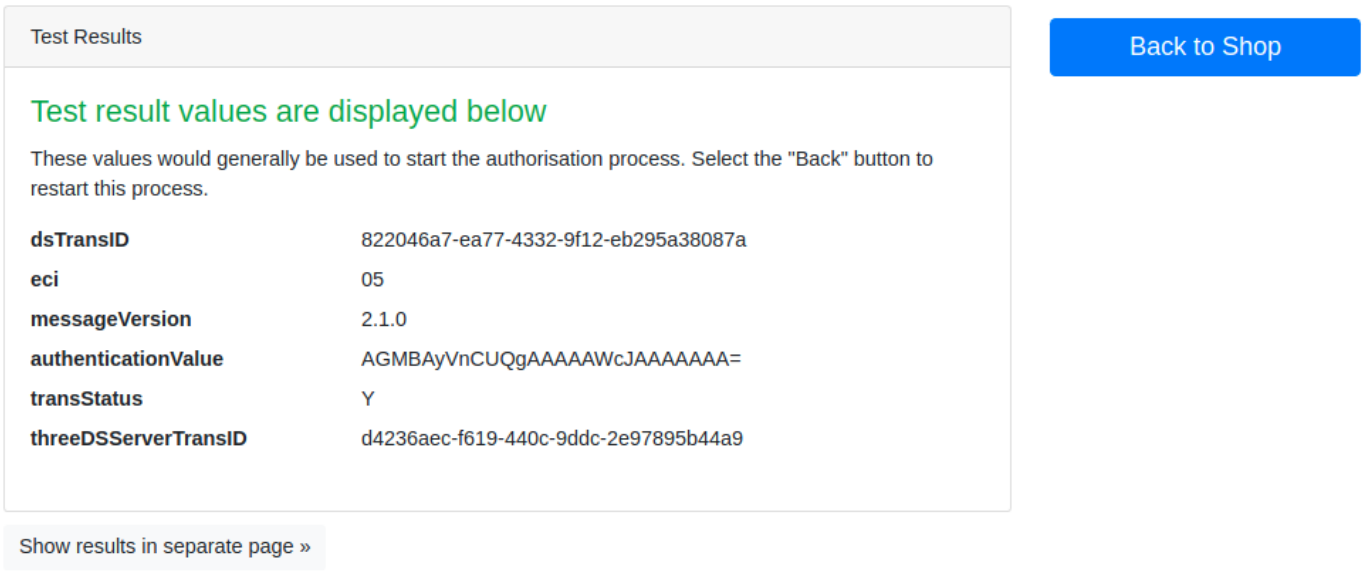
For more information related to `transStatus`, refer to the API document.

Frictionless authentication result

If the `transStatus` is not `C` (i.e. frictionless), the **3ds-web-adapter** will go to the frictionless flow (Step. 14(F)) and show the results using the `_callbackFn("onAuthResult", data)` (line 11).

```
//process.html
function _callbackFn(type, data) {

  switch (type) {
    case "onAuthResult":
      //display "Show results in separate page"
      $("#sepButton").removeClass("d-none");
      showResult(data);
      break;
```

The `showResult(data)` function will display the results in the `process.html` page:



Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

The frictionless authentication process stops at this point. The merchant checkout process can now move to the authorisation process as normal using the authentication result information.

Info

The authentication result can also be shown in a separate page. This is covered in process 3.

Continue challenge process

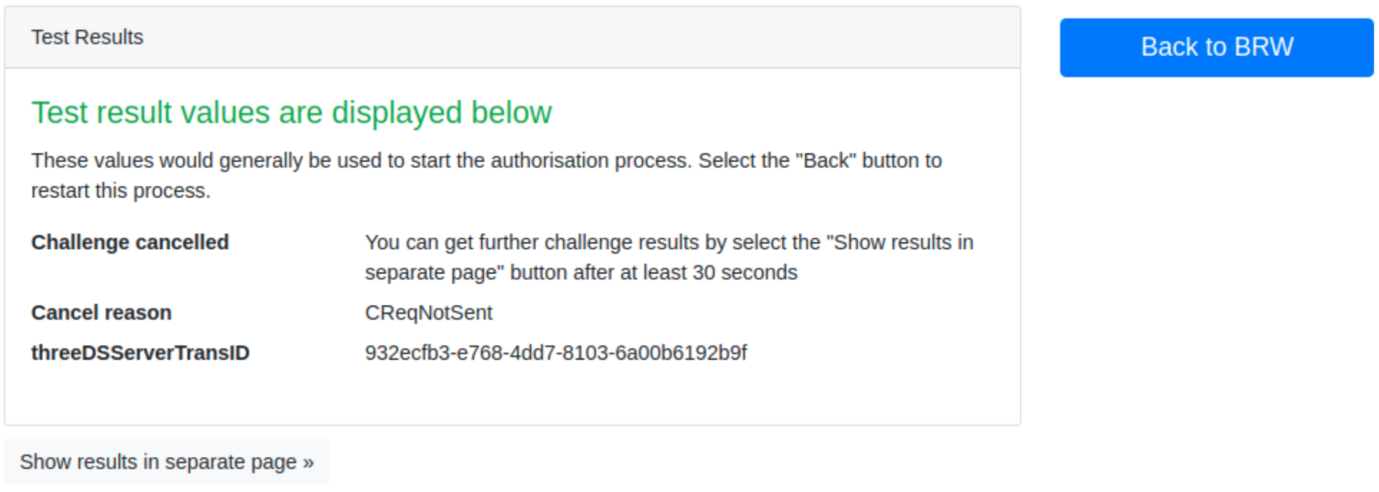
If the `transStatus` is `C`, a challenge is required from the ACS. It is up to the 3DS Requestor to decide whether it wants to continue the 3DS2 authentication and proceed to the challenge process or terminate the authentication process if a challenge is not desired. In this demo, the `options.cancelChallenge` parameter is used to indicate the 3DS Requestors decision on the challenge flow. This feature is outlined on the BRW Test Page.

Note

For demo purposes, the cancel challenge process is implemented in the front end javascript (`3ds-web-adapter`). For security reasons this process may need to be implemented in the back end on the production environment.

The **3ds-web-adapter** will check the `options.cancelChallenge` parameter. If `options.cancelChallenge=true`, the **3ds-web-adapter** will cancel the challenge. Optionally, `options.cancelReason` can also be set and sent to **ActiveServer** depending on the specified Cancel Reason. The specified reason will be sent by the **3ds-web-adapter** to the `challengeStatus` endpoint in the **back-end**. To check how the **back-end** handles this request, refer here.

The cancel challenge results screen should look similar to the screenshot below:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

Challenge cancelled	You can get further challenge results by select the "Show results in separate page" button after at least 30 seconds
Cancel reason	CReqNotSent
threeDSServerTransID	932ecfb3-e768-4dd7-8103-6a00b6192b9f

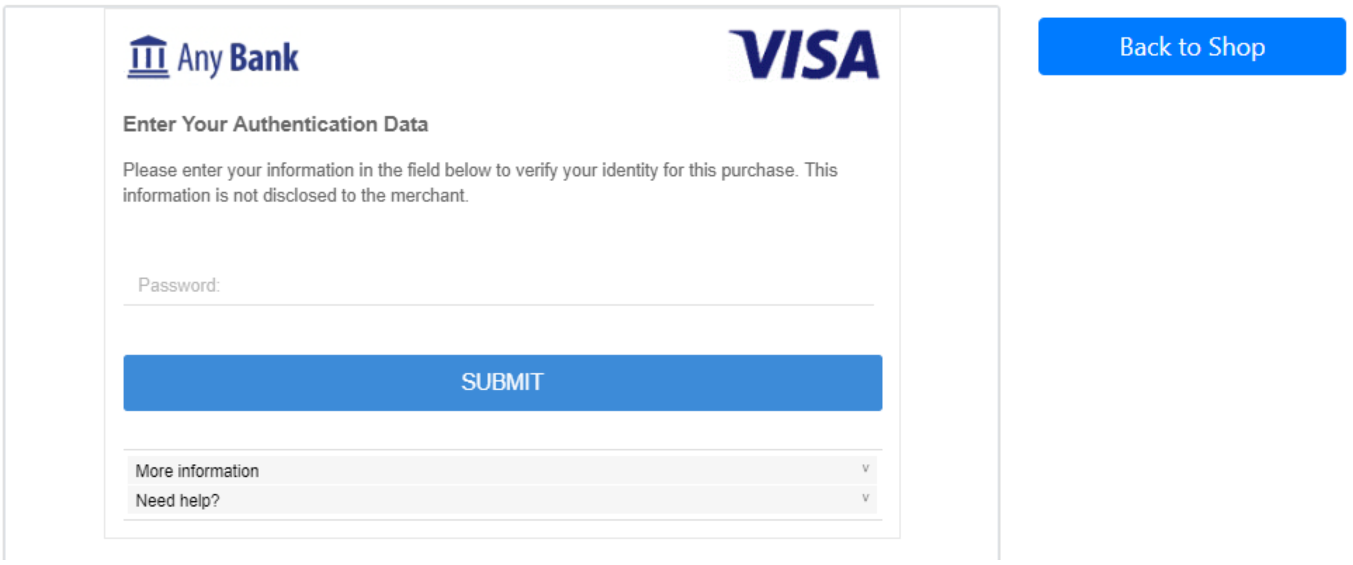
If `options.cancelChallenge=true` is not present (or the value is set to false), the **3ds-web-adapter** will call `startChallenge()`, which inserts an `iframe` for the challenge window with `src` set to `challengeUrl` (Step. 14(C)) in order to show the challenge screen to the card holder.

```
//3ds-web-adapter.js
function startChallenge(url) {

  _callbackFn("onChallengeStart");
  //create the iframe
  $('<iframe src="' + url
      + '" width="100%" height="100%" style="border:0" id="' + "cha_" + iframeId
      + '"></iframe>')
  .appendTo(iframeContainer);

}
```

Note

If you want to test the challenge scenario, follow the guide here.

You can see that the `iframe` class has been set to `width="100%"` and `height="100%"`. This is required because the `iframe` needs to resize according to the content provided by the ACS. The challenge screen should look similar to the screenshot below:


Any Bank

VISA

Back to Shop

Enter Your Authentication Data

Please enter your information in the field below to verify your identity for this purchase. This information is not disclosed to the merchant.

Password:

SUBMIT

More information

Need help?

The cardholder can now input the authentication data such as an OTP password and submit the challenge form. The ACS will authenticate the transaction and the card holder and return the challenge result.

After the challenge flow is finished, **ActiveServer** will call the `3ds-notify` entry point in the **3DS Requestor** backend with event `_onAuthResult` and then the **3DS Requestor** will render the `notify_3ds_events.html` page, similar to before (Step. 19(C)). The **3ds-web-adapter** calls the `_onAuthResult()` function to get the authentication results and displays them on the `process.html` page.

Info

In a production environment the ACS will perform complex risk-based authentication from the information obtained about the cardholder. Similarly, the authentication method (e.g. OTP, biometrics etc) will be determined and implemented by the cardholder's issuing bank.

Process 3: Get Authentication Result

After the challenge process is finished (or if the frictionless process results need to be shown in a separate page), we now need to request for the authentication result so that it can be used for the authorisation process.

Why a separate result request is necessary?

You may wonder why you need to request the result again since you already have the result of authentication available after process 2.

This is because the authentication result is returned to the authentication page in Step. 13 by the 3DS Requestor. It is common that the authentication result page is shown as a separate page from the checkout page. In this case, the **3ds-web-adapter** could transfer the result back to the 3DS Requestor or the result page, however, it is not a recommended data flow as the authentication result is re-transferred by the client side code and is in general considered as insecure.

The server-side of the 3DS Requestor should always have its own mechanism to get the result back from the origin source: **ActiveServer**. Instead of transferring the result to the new result page, the 3DS Requestor server-side can provide a result page that shows the authentication result. Therefore, you need to request for a result receipt in this step.

Here in this demo, you can select `Show results in separate page >>` at the end of process 2 in the `process.html` page to show the result in a separate page.

To get the authentication result and show it in a separate page, the front-end needs to:

- Send a `Request for authentication result` message to the **3DS Requestor** (Step. 15(F) or Step. 20(C)).
- Show result on screen (Step. 17(F) or Step. 22(C)).

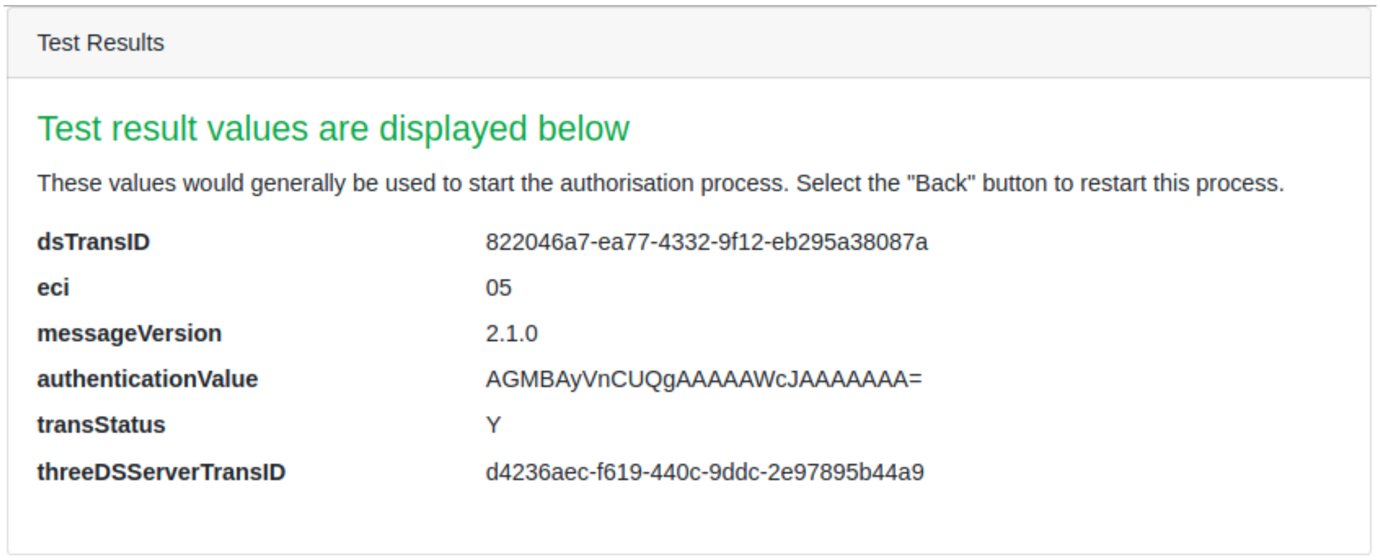
Firstly, in the `process.html` page, it will store the `serverTransId` into `sessionStorage` and go to `result.html` page.

```
//process.html
function openResultInNewWindow() {
if (serverTransId) {
  var url = '/v1/result';

  sessionStorage.setItem("serverTransId", serverTransId);
  window.open(url, 'newwindow', 'height=800,width=1000');
}
}
```

In the `v1/result.html` page, it call the `result()` method in **3ds-web-adapter** to get the authentication result. The `result()` method sends a `Request for authentication result` message to the backend. The backend will receive this request and call the `result` endpoint. To check how the **back-end** handles this request, refer here. The callback function `showData()` shows the results in the page.

```
//result.html
var serverTransId = sessionStorage.getItem("serverTransId");

//request for authentication result (Step 15(F) or Step 20(C))
result(serverTransId, showData);

//show result in separate page (Step 17(F) or Step 22(C))
function showData(type, data) {
    var toShow = "<dl class='row'>";
    Object.keys(data).forEach(function (k) {
      toShow = toShow + "<dt class='col-sm-4'>" + k + "</dt>" + "<dd class='col-
sm-8'>" + data[k]
          + "</dd>";
    });
    toShow += "</dl>";
    $('#showResult').empty().append(toShow);
    $("#resultCard").removeClass("d-none");
}
```

The new result screen will look like the screenshot below:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

✓ **Success**

This covers the front-end integration. After authentication is completed the checkout process can proceed with the authorisation process using the Transaction Status, ECI and CAVV to complete the transaction.

Whats next?

Select **Next** to learn about the **v1 Back-end implementation** for a 3DS Requestor.

Back-end implementation (v1)

In this section, we will show the implementation details for the back-end side of the merchant application using our sample code based on the **Auth API version 1**.

Important

GPayments strongly recommends all new implementations of **ActiveServer** to integrate with version 2 of the Auth API from the beginning, as version 1 will be deprecated in a future release. Refer to the API migration from v1 to v2 guide for a summary of the changes and migration process from **v1** to **v2**. To check the new **API v2** implementation, refer here.

For the back-end, we need to implement a **3DS Requestor**. The **3DS Requestor** receives information from the **3ds-web-adapter** and sends the requests to **ActiveServer**. It also receives the authentication results from **ActiveServer** and forwards the results to the **3ds-web-adapter**.

The demo **3DS Requestor** code provides the backend implementation with the following server side languages:

- **Java**: The java version is implemented based on the Springboot framework. For details of Springboot, check out https://spring.io/projects/spring-boot
- **C#**: The C# version is implemented based on ASP.net.
- **PHP**: The PHP version is implemented based on PHP 7.2 with cURL (Client URL Library).
- **Go**: The Go version is implemented based on Go 1.12 with Go Module support. All dependencies are listed in `go.mod` file.

Why we need to implement a backend?

As defined by EMVCo's 3DSecure 2.0 specification, when the 3DS Server and the 3DS Requestor environment is separated, the communication between these two components must be mutual authenticated:

> [Req 300] If the 3DS Requestor and 3DS Server are separate components, ensure that data transferred between the components is protected at a level that satisfies Payment System security requirements with mutual authentication of both servers.

Before starting the authentication process with **ActiveServer**, the **3DS Requestor** needs to establish a mutual TLS connection with **ActiveServer**. Make sure you have the **client certificate** and configure it with the **3DS Requestor**. If not, follow the Get client certificate and Configure 3DS Requestor details instructions found on the Introduction page.

Tip

The implementation of TLS configuration for the HTTP Client can be found as follows:

- Java: The TLS configuration and client certificate loading can be found in class `RestClientConfig.java` .
- C#: The TLS configuration and client certificate loading can be found in class `RestClientHelper.cs` .
- PHP: The TLS configuration and client certificate loading can be found in file `RestClientConfig.php` .
- Go: The TLS configuration and client certificate loading can be found in file `https.go` .

Next, we will describe the details of the back-end implementation based on the authentication processes and authentication sequence.

Process 1: Initialise the Authentication

To initialise authentication, the **3DS Requestor** needs to:

- Receive the `Initialise authentication` request from the **3DS-web-adapter** (Step. 2).
- Forward the request to **ActiveServer** (Step. 3).
- Receive the response data from **ActiveServer** (Step. 4).
- Return the response data to the **3DS-web-adapter** (Step. 5).

Java C# PHP Go

```java
//AuthControllerV1.java
/**
 * Receives the initialise authentication request from the 3DS-web-adapter
(Step 2)
 * Send data to ActiveServer to Initialise Authentication
 */
@PostMapping("/v1/auth/init/{messageCategory}")
public Message initAuth(@RequestBody Message request,
      @PathVariable(value = "messageCategory") String messageCategory) {

    //Generate requestor trans ID
    String transId = UUID.randomUUID().toString();
    request.put("threeDSRequestorTransID", transId);
    //Fill the event call back url with requestor url + /3ds-notify
    String callBackUrl = config.getBaseUrl() + "/3ds-notify";
    request.put("eventCallbackUrl", callBackUrl);

    //ActiveServer url for Initialise Authentication
    String initAuthUrl = config.getAsAuthUrl() + "/api/v1/auth/brw/init/" +
messageCategory;
    logger.info("initAuthRequest on url: {}, body: \n{}", initAuthUrl, request);

    //Send data to ActiveServer to Initialise authentication (Step 3)
    //Get the response data from ActiveServer (Step 4)
    Message response =
        sendRequest(initAuthUrl, request, HttpMethod.POST);
    logger.info("initAuthResponseBRW: \n{}", response);

    //Return data to 3ds-web-adapter (Step 5)
    return response;
}
```

Note

We set the `eventCallbackUrl` to `{baseUrl}/3ds-notify`. This allows **ActiveServer** to make a notification when the browser information collection (Step. 7) is done. The `baseUrl` is set here.

The `initAuth` request will be sent to `{ActiveServer auth url}/api/v1/auth/brw/init/{messageCategory}`. To check the data structure of `initAuth` request, refer to the API document.

HTTP header for Master Auth API client certificate

If you are using a Master Auth API client certificate to authenticate a **Business Admin** user on behalf of a merchant, the back-end needs to add a HTTP Header in the HTTP Request with a field of `AS-Merchant-Token`, which should be set to the merchantToken from the merchants profile.

Java C# PHP Go

```
//Note: the sendRequest() function will send the request to ActiveServer.
//If this is groupAuth, the request should include a HTTP Header with the field
of AS-Merchant-Token.
private Message sendRequest(String url, Message request, HttpMethod method) {

    HttpEntity<Message> req;
    HttpHeaders headers = null;

    if (config.isGroupAuth()) {
      //the certificate is for groupAuth, work out the header.
      headers = new HttpHeaders();
      headers.add("AS-Merchant-Token", config.getMerchantToken());
    }

    switch (method) {
      case POST:
        req = new HttpEntity<>(request, headers);
        return restTemplate.postForObject(url, req, Message.class);
      case GET:
        if (headers == null) {
          return restTemplate.getForObject(url, Message.class);
        } else {
          req = new HttpEntity<>(headers);
          return restTemplate.exchange(url, HttpMethod.GET, req,
Message.class).getBody();
        }
      default:
        return null;
    }
  }
}
```

Process 2: Execute Authentication

To execute authentication, the **3DS Requestor** needs to:

- Handle the `/3ds-notify` message from **ActiveServer** called through the iframe after Step. 7.
- Handle the `Execute authentication` request from the **3DS-web-adapter** (Step. 9 and Step. 10).
- Receive the authentication result and return it to the **3DS-web-adapter** (Step. 12 and Step. 13).

When the browser information collection (Step. 7) is done, **ActiveServer** makes a notification to the `eventCallbackUrl` which we set to `http://localhost:8082/3ds-notify` . The **3DS Requestor** handles this notification and passes the required parameters to the `notify-3ds-events.html` page.

Java C# PHP Go

```java
//MainController.java
@PostMapping("/3ds-notify")
public String notifyResult(
    @RequestParam("requestorTransId") String transId,
    @RequestParam("event") String callbackType,
    @RequestParam(name = "param", required = false) String param,
    Model model) {

  String callbackName;
  if ("3DSMethodFinished".equals(callbackType)) {

    callbackName = "_on3DSMethodFinished";

  } else if ("3DSMethodSkipped".equals(callbackType)) {

    callbackName = "_on3DSMethodSkipped";

  } else if ("AuthResultReady".equals(callbackType)) {
    callbackName = "_onAuthResult";
  } else {
    throw new IllegalArgumentException("invalid callback type");
  }

  model.addAttribute("transId", transId);
  model.addAttribute("callbackName", callbackName);
  model.addAttribute("callbackParam", param);

  return "notify_3ds_events";
}
```

Note

This handler is called by **ActiveServer**, parameters `requestorTransId`, `event` and an optional `param` will be provided by **ActiveServer**. The `event` can be `3DSMethodFinished`, `3DSMethodSkipped`, `InitAuthTimedOut` or `AuthResultReady`. For descriptions of each event refer to our API document for the field `eventCallbackUrl`. The handler method then sets the proper attribute in the page context and returns to the `notify_3ds_events.html` page. The page then renders the content with Mustache templating engine.

To check the front-end implementation of `notify_3ds_events.html`, refer here.

Then when the 3DS client is finished browser information collecting, it will call the `auth` endpoint to start the authentication. The **3DS Requestor** handles the `Execute authentication` requests (Step. 9, Step. 10, Step. 12, and Step. 13).

Java C# PHP Go

```java
//AuthControllerV1.java
/**
 * Receives the Execute authentication request from the 3DS-web-adapter (Step 9)
 * Send data to ActiveServer to Execute Authentication
 */
@PostMapping("/v1/auth")
public Message auth(@RequestBody Message request) {

    //ActiveServer url for Execute Authentication
    String authUrl = config.getAsAuthUrl() + "/api/v1/auth/brw";
    logger.info("requesting BRW Auth API {}, body: \n{}", authUrl, request);

    //Send data to ActiveServer to Execute Authentication (Step 10)
    //Get the response data from ActiveServer (Step 12)
    Message response =
        sendRequest(authUrl, request, HttpMethod.POST);
    logger.info("authResponseBRW: \n{}", response);

    //Return data to 3ds-web-adapter (Step 13)
    return response;
}
```

Note

The **3DS Requestor** returns a response to the front-end. The returned message contains a `transStatus` with value `Y` or `C`, which will trigger either frictionless flow or challenge flow. To check how the front-end handles the `transStatus`, refer here. To check the structure of the response data, refer to the API document.

Cancel challenge flow

When the `transStatus=C`, the 3DS client can choose to start the challenge or not. If the 3DS client chooses to cancel the challenge, it may call the `challengeStatus` endpoint to specify the Cancel Reason which the **3DS Requestor** will send to **ActiveServer**. To check how the **front-end** handles this request, refer here.

Java C# PHP Go

```java
//AuthControllerV1.java
@PostMapping("/v1/auth/challenge/status")
public Message challengeStatus(@RequestBody Message request) {

    String challengeStatusUrl = config.getAsAuthUrl() + "/api/v1/auth/challenge/
status";
    logger.info("request challenge status API {}, body: \n{}",
challengeStatusUrl, request);

    Message response =
        sendRequest(challengeStatusUrl, request, HttpMethod.POST);
    logger.info("challengeStatus response: \n{}", response);

    return response;
}
```

Process 3: Get Authentication Result

To get the authentication result, the **3DS Requestor** needs to:

- Handle the `Request for authentication result` request from the **3DS-web-adapter** (Step. 15(F) or Step. 20(C)).
- Send a request to **ActiveServer** to get the result and return the result to the front-end. (Step. 16(F), Step. 17(F) or Step. 21(C), Step. 22(C)).

Java C# PHP Go

```java
//AuthControllerV1.java
/**
 * Receives the Request for authentication result request (Step 15(F) and Step 20(C))
 * Send data to ActiveServer to Retrieve Authentication Results
 */
@GetMapping("/v1/auth/result")
public Message result(@RequestParam("txid") String serverTransId) {

    //ActiveServer url for Retrieve Results
    String resultUrl = config.getAsAuthUrl() +
        "/api/v1/auth/brw/result?threeDSServerTransID=" +
        serverTransId;

    //Get authentication result from ActiveServer (Step 16(F) and Step 21(C))
    Message response =
        sendRequest(resultUrl, null, HttpMethod.GET);
    logger.info("authResponse: \n{}", response);
    //Show authentication results on result.html (Step 17(F) and Step 22(C))
    return response;
}
```

✓ **Success**

This covers the backend implementation. After authentication is completed the checkout process can move on to performing authorisation using the Transaction Status, ECI and CAVV to complete the transaction.

Whats next?

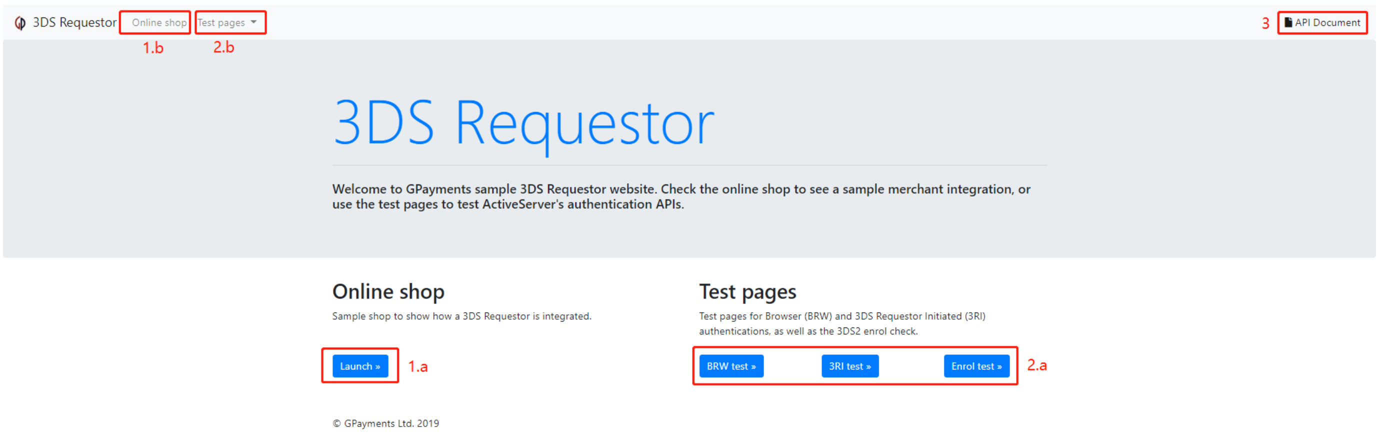
Select **Next** to go through all the features of our **Sample Code** for the **GPayments 3DS Requestor**.

Sample code features

In this section, we show all the features of the **3DS Requestor Demo** project. You can access the 3DS Requestor demo by:

- Downloading and running the 3DS Requestor demo code here, or
- Using our online TestLabs Requestor

Shown below is the index page of the 3DS Requestor site. There are two parts, **Online shop** and **Test pages**. The online shop can be accessed by selecting the `Launch` button (1.a) or selecting the `Online shop` button (1.b) at the top left. Each test page can be accessed by selecting the associated buttons (2.a) or selecting the `Test pages` dropdown (2.b) at the top left. Additionally, at the top right there is a `API Document` link (3) which will direct you to the **ActiveServer Authentication API Reference** document.



Welcome to GPayments sample 3DS Requestor website. Check the online shop to see a sample merchant integration, or use the test pages to test ActiveServer's authentication APIs.

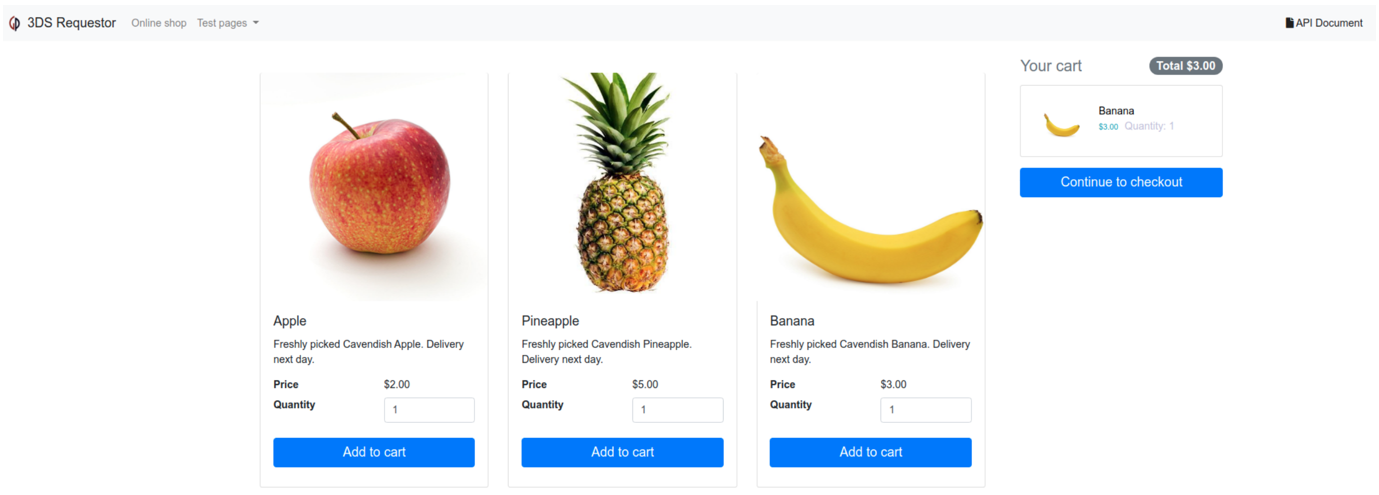
Sample shop to show how a 3DS Requestor is integrated.

Test pages for Browser (BRW) and 3DS Requestor Initiated (3RI) authentications, as well as the 3DS2 enrol check.

Online Shop

The online shop is a demo merchant checkout page that gives a merchant side view of how to integrate 3DS2 into an eCommerce site. For the implementation of the integration, follow the Integration Guide. To make a test transaction:

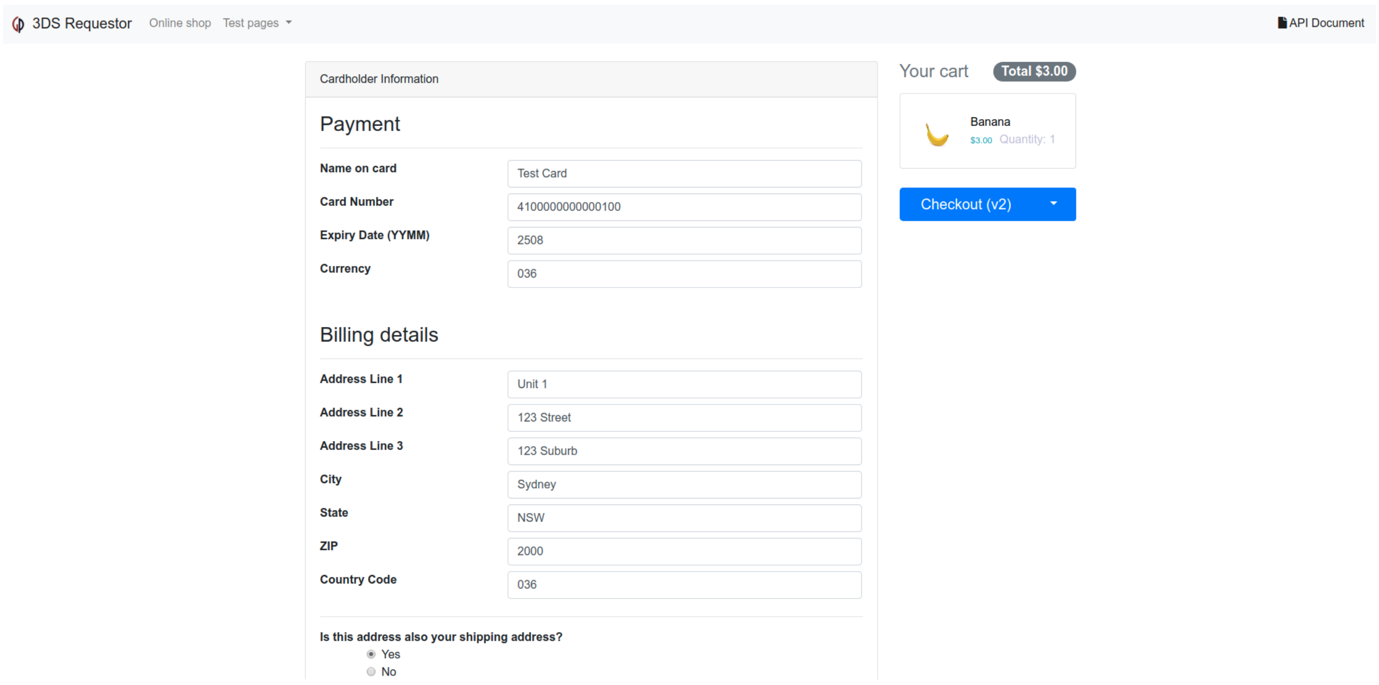
Add items to the cart in the shop page and select the ***Continue to Checkout*** button to move to the checkout page:



Default payment and billing information is pre-filled, including a card number, which can be used to complete a frictionless transaction. Select the ***Checkout*** button to trigger the 3DS2 authentication process:

Note

You can select the API version to perform the 3DS2 authentication process by clicking the arrow at the right of the ***Checkout*** button.



Cardholder Information

Payment

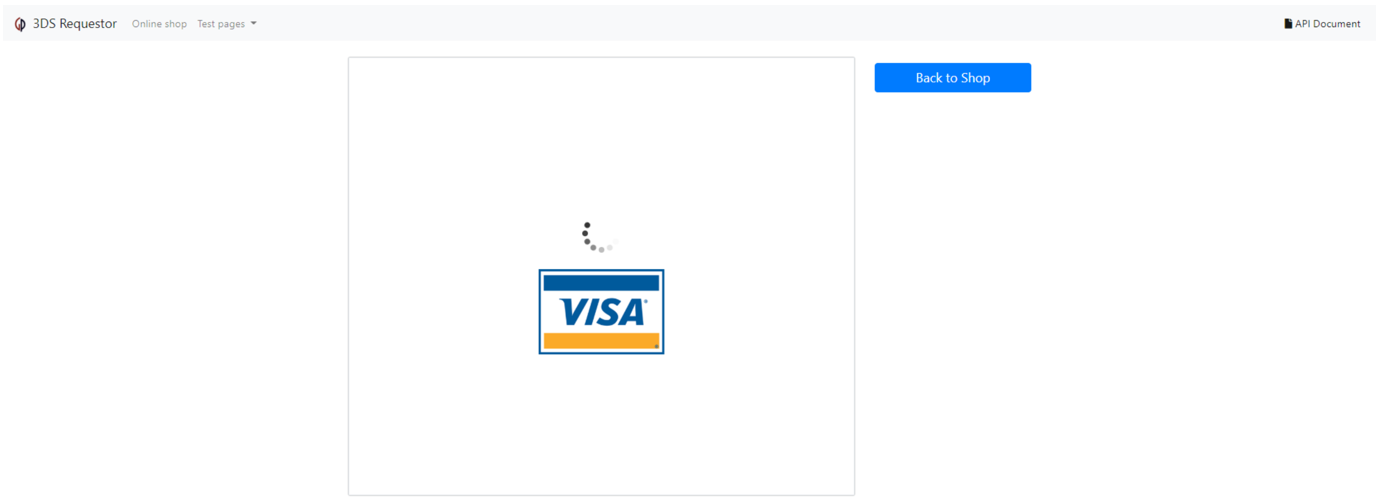
Name on card	Test Card
Card Number	4100000000000100
Expiry Date (YYMM)	2508
Currency	036

Billing details

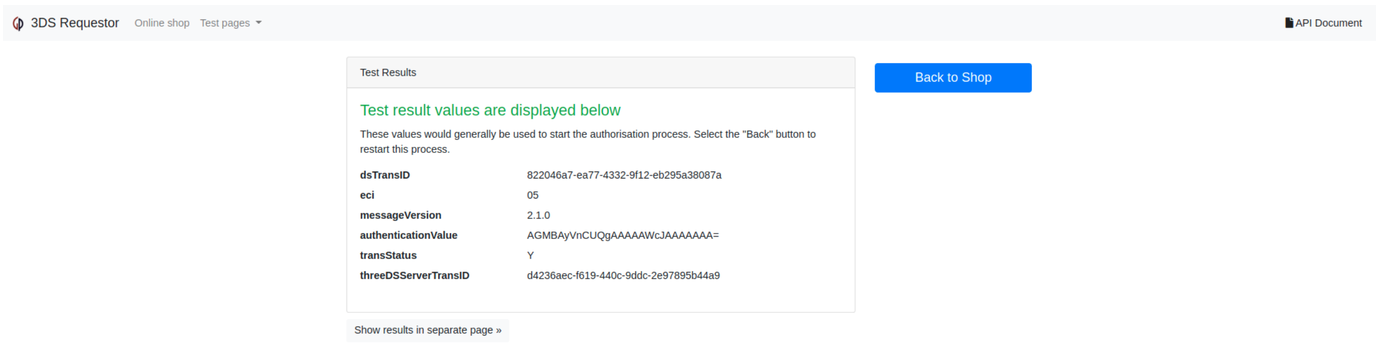
Address Line 1	Unit 1
Address Line 2	123 Street
Address Line 3	123 Suburb
City	Sydney
State	NSW
ZIP	2000
Country Code	036

Is this address also your shipping address?
[✓ Yes]
No

There is a spinner on the processing page while the 3DS2 authentication is in progress:



After the 3DS2 authentication is finished the result will be shown:



Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

Show results in separate page »

Selecting the `Show results in separate pages` will open a new page to show the results.

Challenge flow

The above demonstrates a frictionless authentication. If the authentication requires a cardholder challenge, this will be shown after the first spinner processing page.

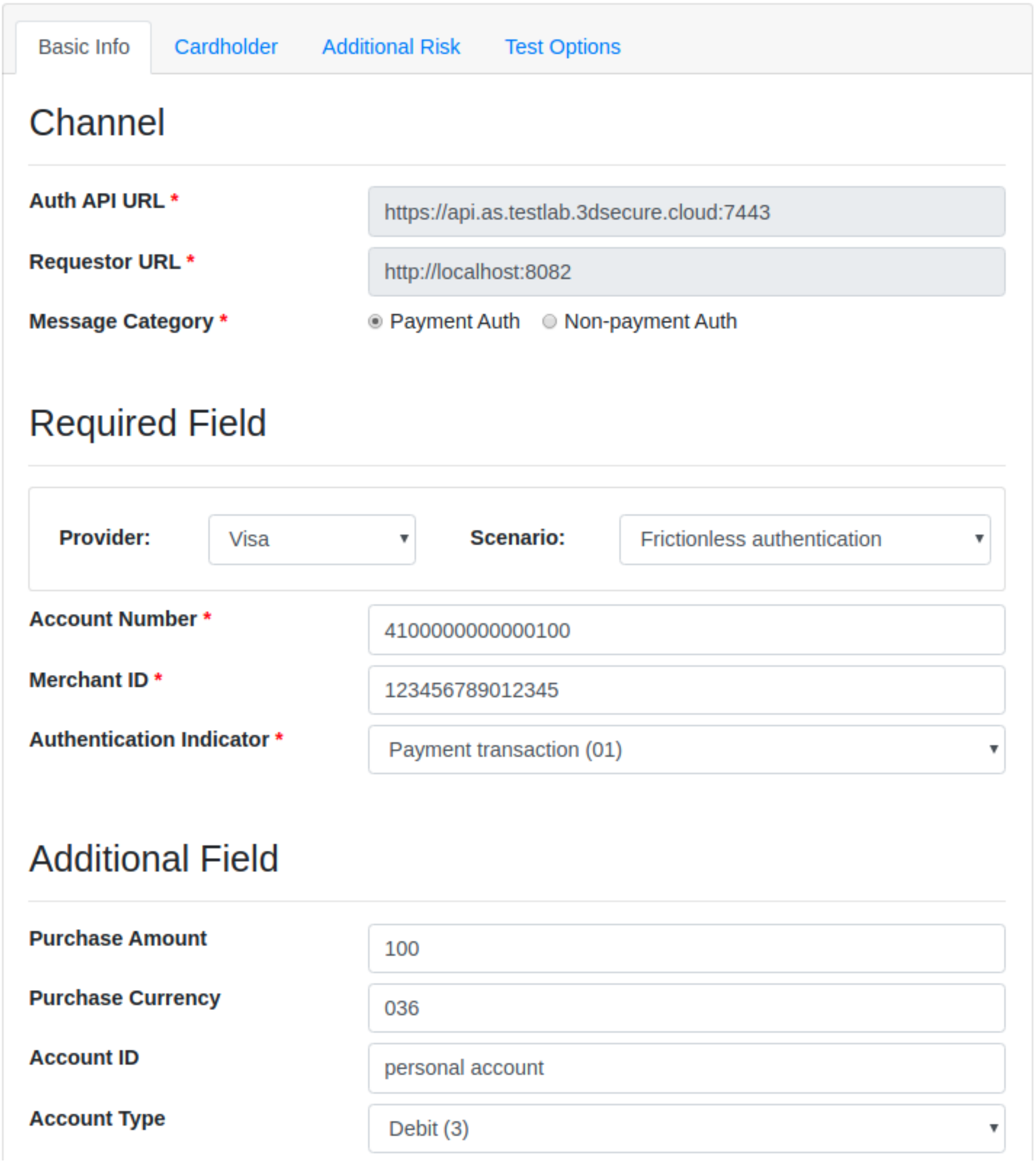
Test Pages

The test pages allow you to test the **Browser-based Authentication (BRW)**, **3DS Requestor Initiated (3RI)** and **Enrol** channels with all the parameters defined in the API document.

BRW Test Page

The BRW test page has four tabs, `Basic Info`, `Cardholder`, `Additional Risk` and `Test Options`.

Basic Info



Channel

Auth API URL *	https://api.as.testlab.3dsecure.cloud:7443
Requestor URL *	http://localhost:8082
Message Category *	[✓ Payment Auth] Non-payment Auth

Required Field

Provider:	Visa	Scenario:	Frictionless authentication

Account Number *	4100000000000100
Merchant ID *	123456789012345
Authentication Indicator *	Payment transaction (01)

Additional Field

Purchase Amount	100
Purchase Currency	036
Account ID	personal account
Account Type	Debit (3)

The `Basic Info` tab has three sections, `Channel`, `Required Field`, and `Additional Field`.

In `Channel`, you can check the **3DS Server URL** and the **Requestor URL** that have been loaded into the application. You can also select which **Message Category** is used for the test, either `pa` or `npa`.

In `Required Field`, you need to fill in the **Account Number**, **Merchant ID** and **Authentication Indicator**. The **Merchant ID** must match the one from the merchant profile you are using, which also corresponds to the **client certificate** being used.

Using GPayments TestLabs scenarios

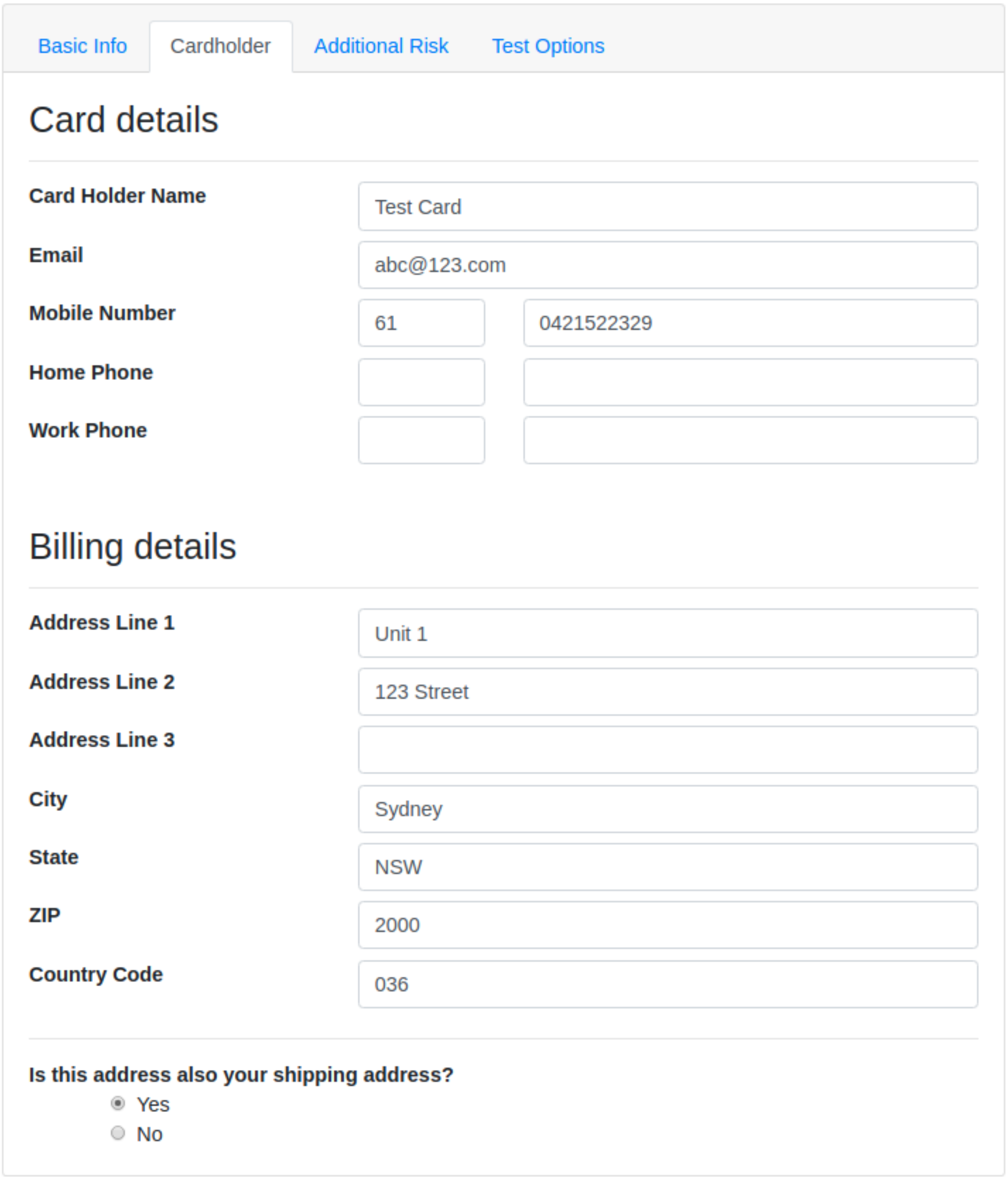
For the **Account Number**, you can fill in an account number manually, or you can choose from our predefined scenarios. For example, you can choose `Visa` with `Frictionless authentication` and the the account number `4100000000000100` will automatically be filled in the Account Number field.

The demo 3DS Requestor allows you to trigger various 3DS2 authentication scenarios by specifying a card number before the authentication. To check the card list for different authentication scenarios, refer to the TestLabs guide for details.

In `Additional Field`, you can fill in additional information such as the **Purchase amount**, **Purchase currency**, **Expiry date**, etc.

Cardholder

On the `Cardholder` tab, you can fill in cardholder information, including the **Card holder name**, **Email**, phone numbers and address details.



Basic Info | Cardholder | Additional Risk | Test Options

Card details

Card Holder Name	Test Card	
Email	abc@123.com	
Mobile Number	61	0421522329
Home Phone		
Work Phone		

Billing details

Address Line 1	Unit 1
Address Line 2	123 Street
Address Line 3	
City	Sydney
State	NSW
ZIP	2000
Country Code	036

Is this address also your shipping address?

[✓ Yes]

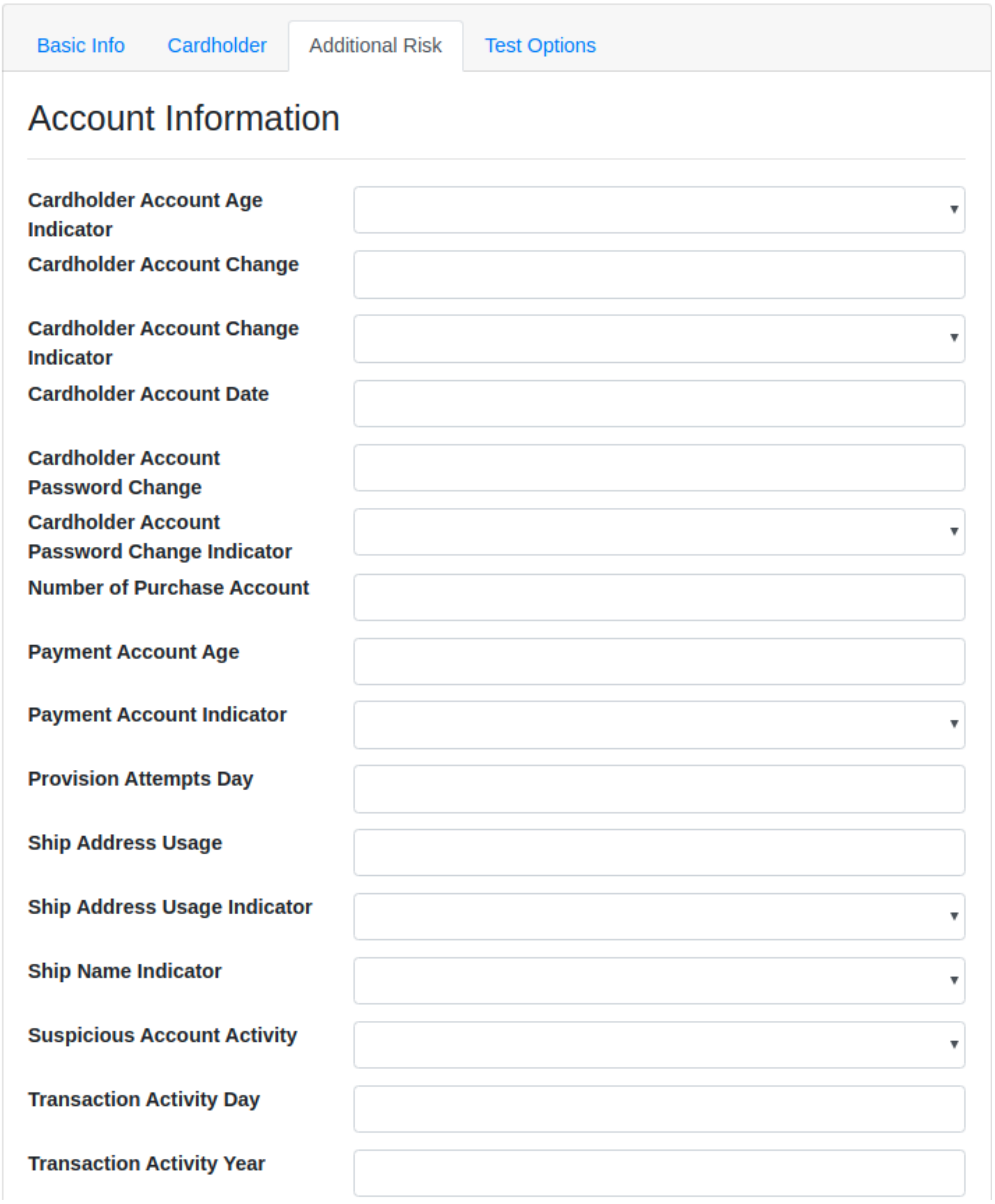
No

Additional Risk

On the `Additional Risk` tab, you can fill in other additional risk information, such as the account information, the requestor authentication information, and the merchant risk indicator.

Merchant risk information

This information is optional to provide. However, it is very beneficial for the merchant to provide this information if it is available, as it will assist the issuers ACS with risk-based authentication, and potentially lead to higher rates of frictionless authentications.



Account Information

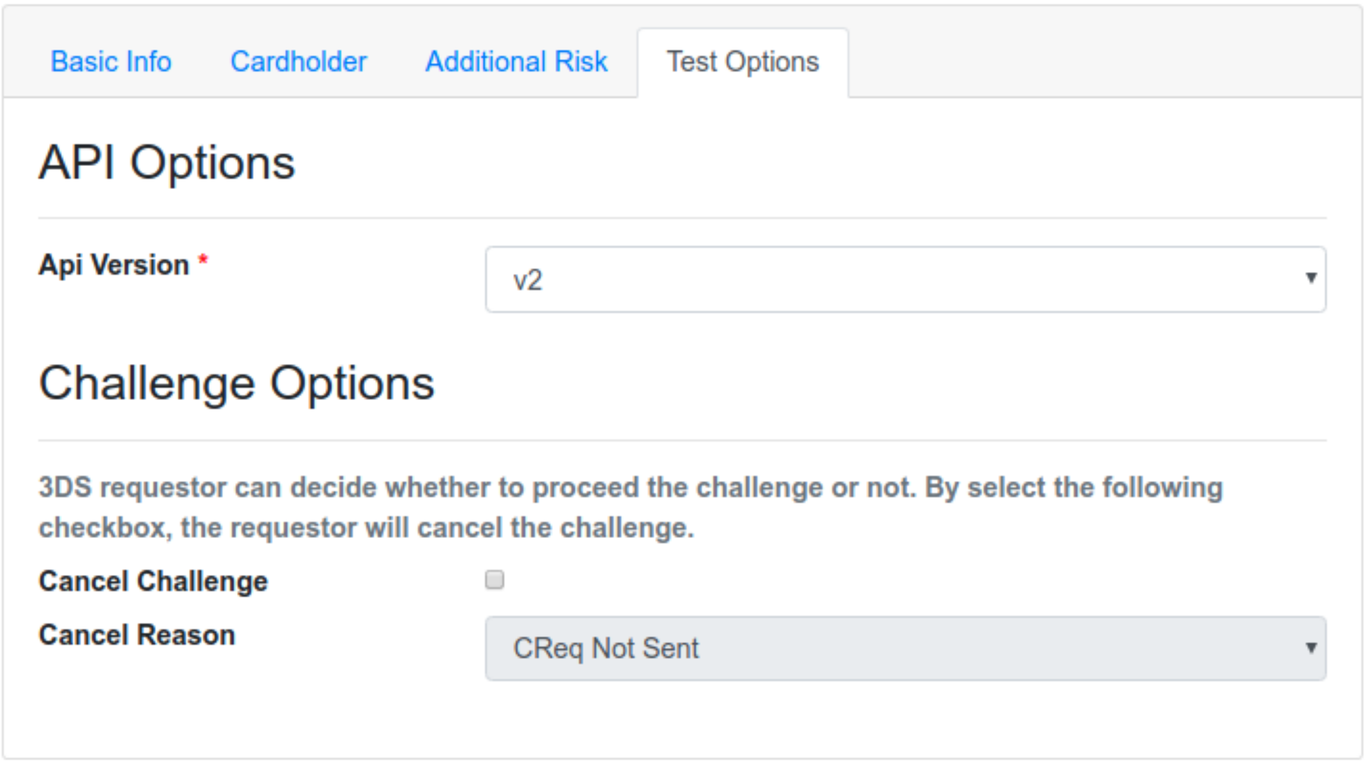
Cardholder Account Age Indicator	▾
Cardholder Account Change	
Cardholder Account Change Indicator	▾
Cardholder Account Date	
Cardholder Account Password Change	
Cardholder Account Password Change Indicator	▾
Number of Purchase Account	
Payment Account Age	
Payment Account Indicator	▾
Provision Attempts Day	
Ship Address Usage	
Ship Address Usage Indicator	▾
Ship Name Indicator	▾
Suspicious Account Activity	▾
Transaction Activity Day	
Transaction Activity Year	

Test Options

On the `Test Options` tab, firstly, you can choose which API version to use to perform the 3DS authentication process.

Secondly, you can choose to cancel a challenge if the ACS requests one for the transaction by selecting the `Cancel Challenge` checkbox. When cancelling the challenge, the `3ds-web-adapter` will not execute the CReq callback page in the iframe. Optionally the `Cancel Reason` can then be specified using the `challengeStatus` endpoint to let **ActiveServer** know why the challenge is being cancelled:

- **CReq Not Sent**: indicates that the challenge request was not initiated due to the 3DS Requestor choosing to opt out of the challenge. Will send a status of `CReqNotSent` to the `challengeStatus` endpoint.
- **Auth Result Not Delivered**: indicates that the challenge request was not able to be delivered to the 3DS Requestor due to a technical error, such as **ActiveServer** not responding when the 3DS Requestor executes an authentication request. Will send a status of `AuthResultNotDelivered` to the `challengeStatus` endpoint.
- **No Reason Sent**: simulates a scenario where the 3DS Requestor does not initiate the challenge, but also does not call the `challengeStatus` endpoint, meaning no reason will be sent to **ActiveServer**.



API Options

Api Version *	v2

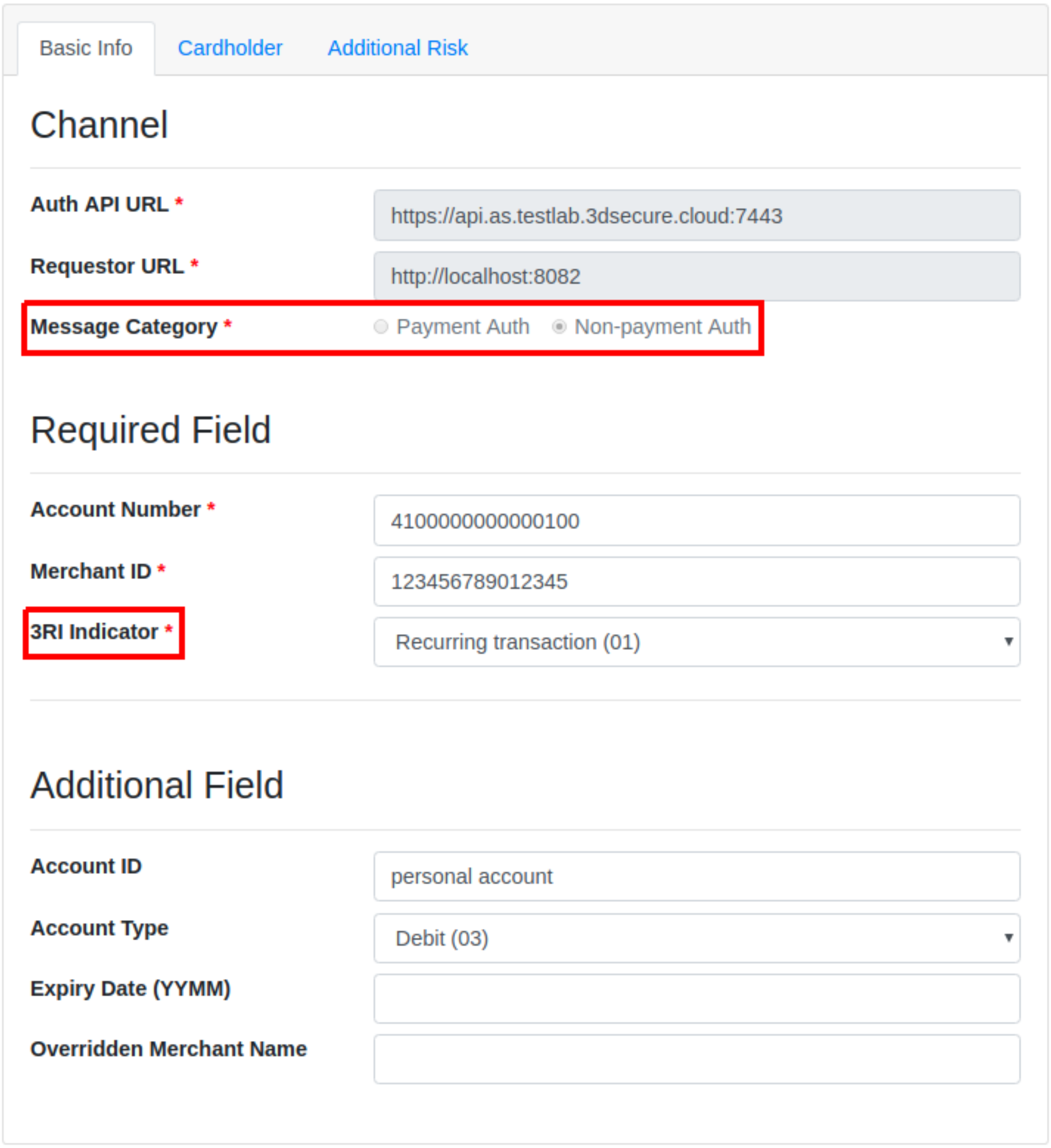
Challenge Options

3DS requestor can decide whether to proceed the challenge or not. By select the following checkbox, the requestor will cancel the challenge.

Cancel Challenge	☐
Cancel Reason	CReq Not Sent

3RI Test Page

The 3RI test page initiates an authentication using the 3RI channel. Similar to the BRW test page, the 3RI test page has three tabs, `Basic Info`, `Cardholder`, and `Additional Risk`. One difference is that there is a `3RI Indicator` parameter in the `Required Field` section and there are also less parameters in the `Additional Field` due to the nature of 3RI. Lastly, you are not able to choose a `Message Category` in 3RI tests because in version 2.1.0 of 3DS2 the 3RI channel can only used for Non-payment authentication.



Channel

Auth API URL *	https://api.as.testlab.3dsecure.cloud:7443
Requestor URL *	http://localhost:8082
Message Category *	Payment Auth [✓ Non-payment Auth]

Required Field

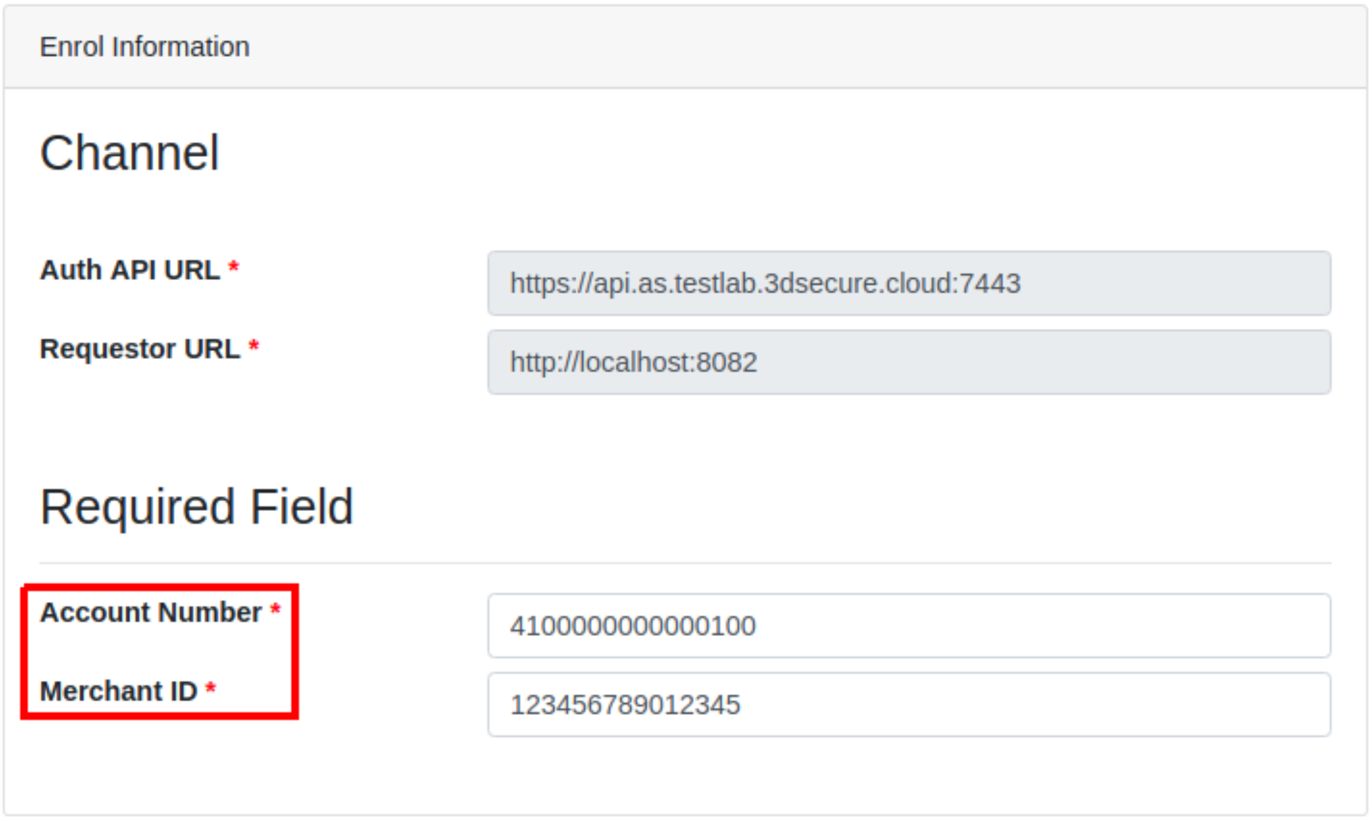
Account Number *	4100000000000100
Merchant ID *	123456789012345
3RI Indicator *	Recurring transaction (01)

Additional Field

Account ID	personal account
Account Type	Debit (03)
Expiry Date (YYMM)	
Overridden Merchant Name	

Enrol Test Page

The enrol test page initiates an `/enrol` check with **ActiveServer**, which is used to see if a card is in a card range that supports 3DS2. Only an **Account Number** and a **Merchant ID** are required for the enrol test.


Enrol Information

Channel

Auth API URL *	https://api.as.testlab.3dsecure.cloud:7443
Requestor URL *	http://localhost:8082

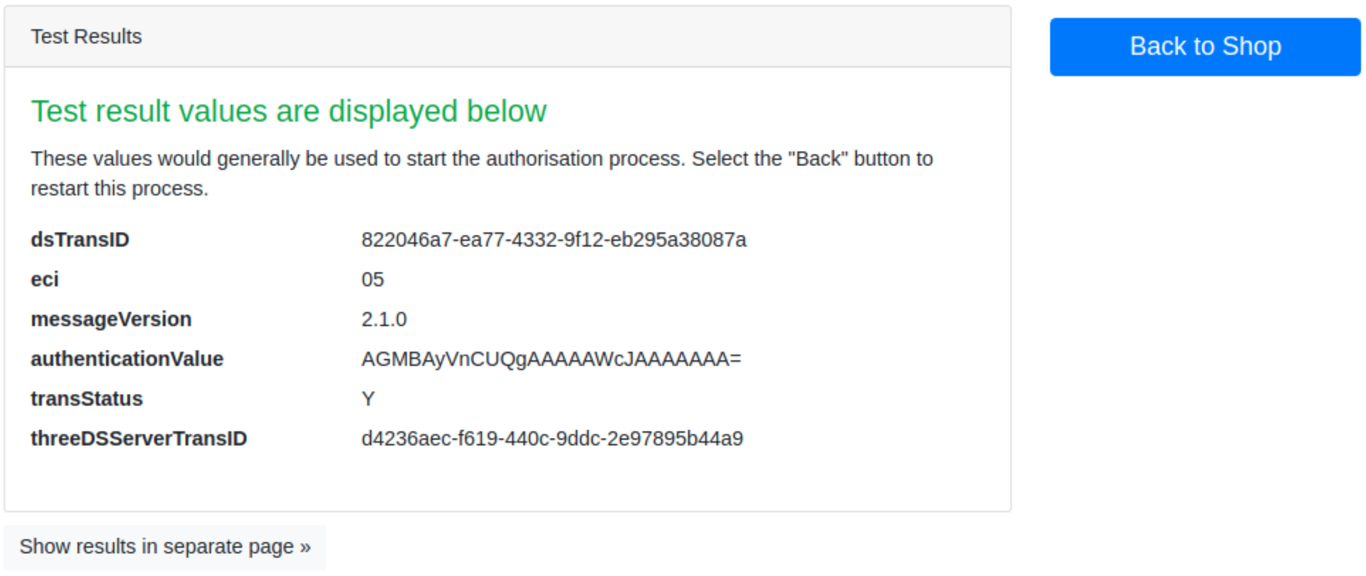
Required Field

Account Number *	4100000000000100
Merchant ID *	123456789012345

Test Results

After selecting the ***Test*** button on one of the test pages, you will be directed to `process.html`. All the 3DS2 processes will be handled by a `process.html` page and there will be a spinner while the 3DS2 process is running. When the 3DS2 process is done, the results will be displayed on the same page.

For example, if you select `BRW test` page then select `Test BRW` button with pre-filled information, it will get a successful response.



Test Results

Back to Shop

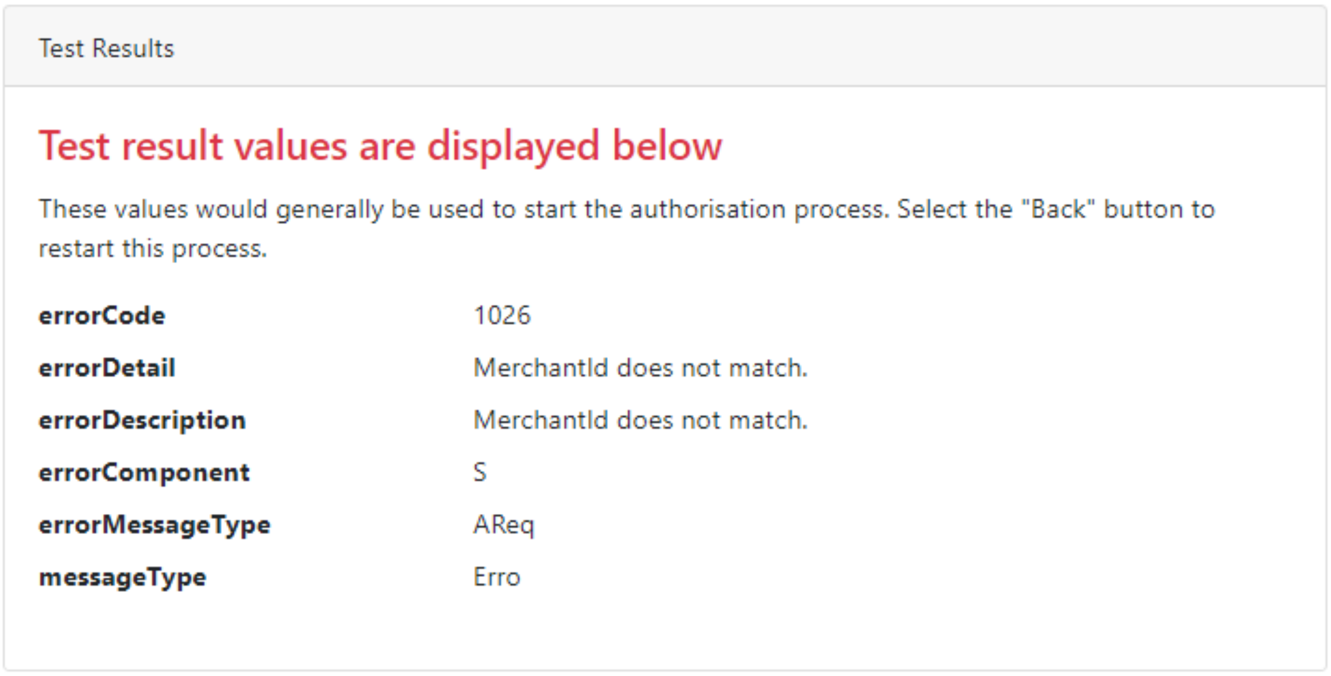
Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

dsTransID	822046a7-ea77-4332-9f12-eb295a38087a
eci	05
messageVersion	2.1.0
authenticationValue	AGMBAyVnCUQgAAAAAWcJAAAAAAA=
transStatus	Y
threeDSServerTransID	d4236aec-f619-440c-9ddc-2e97895b44a9

Show results in separate page »

Or, you can fill in an invalid parameter in `BRW test` page (e.g. `000` for `Merchant ID`), and it will return an error information.



Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

errorCode	1026
errorDetail	MerchantId does not match.
errorDescription	MerchantId does not match.
errorComponent	S
errorMessageType	AReq
messageType	Erro

Directory tree

Below are the directory trees for Java, C#, PHP, and Go.

Java

21 directories, 52 files

```
.
├── mvnw
├── mvnw.cmd
├── pom.xml
└── src
    └── main
        ├── java
        │   └── com
        │       └── gpayments
        │           └── requestor
        │               └── testlab
        │                   ├── AuthControllerV1.java
        │                   ├── AuthControllerV2.java
        │                   ├── config
        │                   │   ├── Config.java
        │                   │   └── RestClientConfig.java
        │                   ├── dto
        │                   │   └── Message.java
        │                   ├── exception
        │                   │   └── GlobalExceptionHandler.java
        │                   ├── MainController.java
        │                   └── RequestorApplication.java
        └── resources
            ├── application.yml
            ├── certs
            │   └── cacerts.pem
            ├── static
            │   ├── css
            │   │   ├── cart.css
            │   │   ├── spinner.css
            │   │   └── style.css
            │   ├── favicon.ico
            │   ├── images
            │   │   ├── amex-logo.png
            │   │   ├── apple.jpeg
            │   │   ├── banana.jpg
            │   │   ├── discover-logo.png
            │   │   ├── jcb-logo.png
            │   │   ├── left-icon.ico
            │   │   ├── mastercard-logo.png
            │   │   ├── pineapple.jpeg
            │   │   └── visa-logo.png
            │   └── js
            │       ├── cart.js
            │       ├── check-credit-card-type.js
            │       ├── common.js
            │       ├── test-lab-scenarios.js
            │       ├── v1
            │       |   └── 3ds-web-adapter.js
            │       └── v2
            │           └── 3ds-web-adapter.js
            └── templates
                ├── 3ri.html
                ├── brw.html
                ├── checkout.html
                ├── contents
                │   ├── acct_info.html
                │   ├── authentication_info.html
                │   ├── cardholder_info.html
                │   ├── deps.html
                │   ├── merchant_risk_indicator.html
                │   ├── nav_bar.html
                │   ├── process_head.html
                │   └── process_main_body.html
                ├── enrol.html
                ├── error.html
                ├── index.html
                ├── notify_3ds_events.html
                ├── shop.html
                ├── v1
                │   ├── process.html
                │   └── result.html
                └── v2
                    ├── process.html
                    └── result.html
```

C#

17 directories, 61 files

```
.
├── 3ds2RequestorDemo.csproj
├── 3ds2RequestorDemo.sln
├── App_Start
│   ├── FilterConfig.cs
│   ├── RouteConfig.cs
│   └── WebApiConfig.cs
├── Certs
│   └── cacerts.pem
├── Controllers
│   ├── AuthV1Controller.cs
│   ├── AuthV2Controller.cs
│   ├── EnrollController.cs
│   └── MainController.cs
├── css
│   ├── cart.css
│   ├── spinner.css
│   └── style.css
├── favicon.ico
├── Global.asax
├── Global.asax.cs
├── Helpers
│   ├── Config.cs
│   └── RestClientHelper.cs
├── images
│   ├── amex-logo.png
│   ├── apple.jpeg
│   ├── banana.jpg
│   ├── cart.png
│   ├── discover-logo.png
│   ├── jcb-logo.png
│   ├── left-icon.ico
│   ├── mastercard-logo.png
│   ├── pineapple.jpeg
│   └── visa-logo.png
├── js
│   ├── cart.js
│   ├── check-credit-card-type.js
│   ├── common.js
│   ├── test-lab-scenarios.js
│   ├── v1
│   │   └── 3ds-web-adapter.js
│   └── v2
│       └── 3ds-web-adapter.js
├── Models
│   └── dto
│       └── Message.cs
├── packages.config
├── Properties
│   └── AssemblyInfo.cs
├── startup.bat
├── Views
│   ├── Main
│   │   ├── 3ri.html
│   │   ├── brw.html
│   │   ├── checkout.html
│   │   ├── contents
│   │   │   ├── acct_info.html
│   │   │   ├── authentication_info.html
│   │   │   ├── cardholder_info.html
│   │   │   ├── deps.html
│   │   │   ├── merchant_risk_indicator.html
│   │   │   ├── nav_bar.html
│   │   │   ├── process_head.html
│   │   │   └── process_main_body.html
│   │   ├── enrol.html
│   │   ├── error.html
│   │   ├── index.html
│   │   ├── notify_3ds_events.html
│   │   ├── v1
│   │   │   ├── process.html
│   │   │   └── result.html
│   │   └── v2
│   │       ├── process.html
│   │       └── result.html
│   ├── Shared
│   │   └── Error.cshtml
│   ├── _ViewStart.cshtml
│   └── Web.config
└── Web.config
```

PHP

14 directories, 53 files

```
.
├── composer.json
├── composer.lock
├── config
│   ├── Config.php
│   ├── RestClientConfig.php
│   ├── Router.php
│   └── Utils.php
├── controllers
│   ├── AuthControllerV1.php
│   ├── AuthControllerV2.php
│   └── MainController.php
├── css
│   ├── cart.css
│   ├── spinner.css
│   └── style.css
├── favicon.ico
├── images
│   ├── amex-logo.png
│   ├── apple.jpeg
│   ├── banana.jpg
│   ├── cart.png
│   ├── discover-logo.png
│   ├── jcb-logo.png
│   ├── left-icon.ico
│   ├── mastercard-logo.png
│   ├── pineapple.jpeg
│   └── visa-logo.png
├── index.php
├── js
│   ├── cart.js
│   ├── check-credit-card-type.js
│   ├── common.js
│   ├── test-lab-scenarios.js
│   ├── v1
│   |   └── 3ds-web-adapter.js
│   └── v2
│       └── 3ds-web-adapter.js
├── README.md
└── resources
    ├── application.ini
    ├── certs
    │   └── cacerts.pem
    └── templates
        ├── 3ri.html
        ├── brw.html
        ├── checkout.html
        ├── contents
        │   ├── acct_info.html
        │   ├── authentication_info.html
        │   ├── cardholder_info.html
        │   ├── deps.html
        │   ├── merchant_risk_indicator.html
        │   ├── nav_bar.html
        │   ├── process_head.html
        │   └── process_main_body.html
        ├── enrol.html
        ├── error.html
        ├── index.html
        ├── notify_3ds_events.html
        ├── shop.html
        ├── v1
        │   ├── process.html
        │   └── result.html
        └── v2
            ├── process.html
            └── result.html
```

Go

11 directories, 47 files

```
.
├── conf
│   ├── application.yaml
│   └── cacerts.pem
├── conf.go
├── go.mod
├── go.sum
├── https.go
├── main.go
└── web
    ├── 3ri.html
    ├── brw.html
    ├── checkout.html
    ├── contents
    │   ├── acct_info.html
    │   ├── authentication_info.html
    │   ├── cardholder_info.html
    │   ├── deps.html
    │   ├── merchant_risk_indicator.html
    │   ├── nav_bar.html
    │   ├── process_head.html
    │   └── process_main_body.html
    ├── css
    │   ├── cart.css
    │   ├── spinner.css
    │   └── style.css
    ├── enrol.html
    ├── error.html
    ├── favicon.ico
    ├── images
    │   ├── amex-logo.png
    │   ├── apple.jpeg
    │   ├── banana.jpg
    │   ├── cart.png
    │   ├── discover-logo.png
    │   ├── jcb-logo.png
    │   ├── left-icon.ico
    │   ├── mastercard-logo.png
    │   ├── pineapple.jpeg
    │   └── visa-logo.png
    ├── index.html
    ├── js
    │   ├── cart.js
    │   ├── check-credit-card-type.js
    │   ├── common.js
    │   ├── test-lab-scenarios.js
    │   ├── v1
    │   │   └── 3ds-web-adapter.js
    │    └── v2
    │        └── 3ds-web-adapter.js
    ├── notify_3ds_events.html
    ├── shop.html
    ├── v1
    │   ├── process.html
    │   └── result.html
    └── v2
        ├── process.html
        └── result.html
```

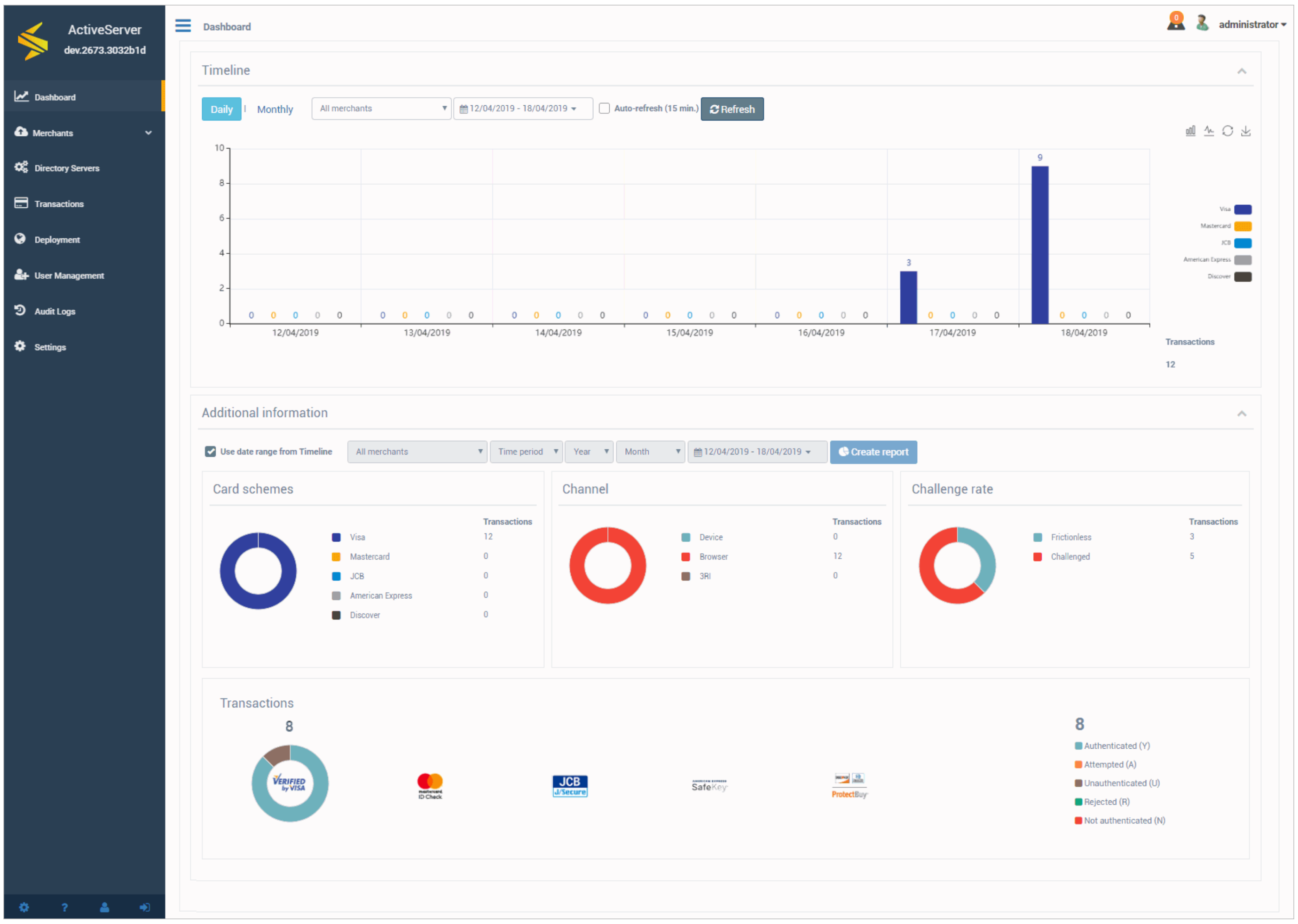
Using the dashboard

The **Dashboard** has two sections:

- Timeline
- Additional information

Tip

Each dashboard section can be collapsed/expanded by clicking the section header.



Card schemes

	Transactions
Visa	12
Mastercard	0
JCB	0
American Express	0
Discover	0

Channel

	Transactions
Device	0
Browser	12
3RI	0

Challenge rate

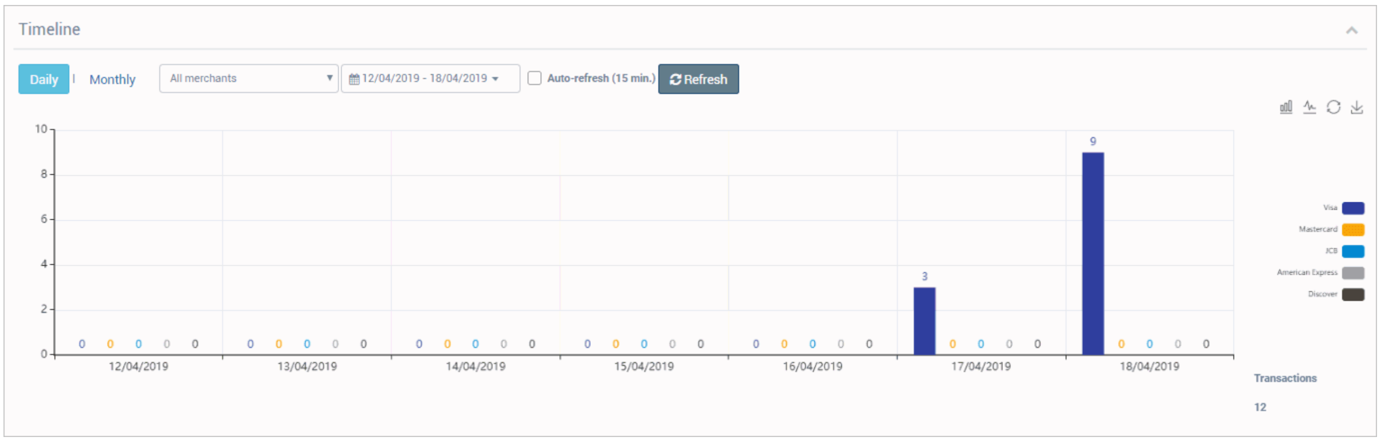
	Transactions
Frictionless	3
Challenged	5

Timeline

The **Timeline** provides an historical, graphical breakdown of the number of transactions performed during a specified time period, and is separated by card scheme. Each card scheme view can be toggled off by clicking its icon on the right hand side of the graph. The graph is available in both line and bar chart format, which can be toggled between using the relevant buttons in the top right hand corner of the chart.

The interface can be set to update every 15 minutes by choosing the **Auto-refresh** option. The ***Refresh*** button can be used to refresh the data on the screen to the most recently collected data.

The data can be shown as either a **Daily** or **Monthly** view, indicating what the column values represent.



Daily

When the **Daily** view is selected, each column on the graph represents one calendar day in **DD/MM/YYYY** format. The **Daily** view comes with pre-selectable date ranges for quick viewing:

- `Last 7 days` - previous 7 calendar days, including current day.
- `Last 30 days` - last 30 calendar days, including current day.
- `Current month` - all days in the current calendar month, including current day.
- `Last month` - all days in the last calendar month.

You can also select a **custom date range** using the date range picker. This custom date range can be a minimum of 1 day and maximum of 31 days.

Monthly

When the **Monthly** view is selected, each column on the graph represents one calendar month in **MM/YYYY** format. The **Monthly** view comes with pre-selectable date ranges for quick viewing:

- `Current month` - current calendar month, including current day.
- `Current year` - current calendar year, including current day.
- `Last month` - previous calendar month.
- `Last year` - previous calendar year.

You can also select a **custom date range** using the date range picker. This custom date range can be a minimum of 1 month and maximum of 12 months.

Additional information

The **Additional information** section of the dashboard gives a deeper level of insight into the final status of each transaction performed in **ActiveServer**. This information can be used to supplement the information shown in the **Timeline** section or to view transaction statistics for any historical date range for the system.

Time periods

If **Use date range from Timeline** is selected, date range and merchant selection options will be greyed out and the options chosen in the **Timeline** section will be used to show a synchronised view across both sections.

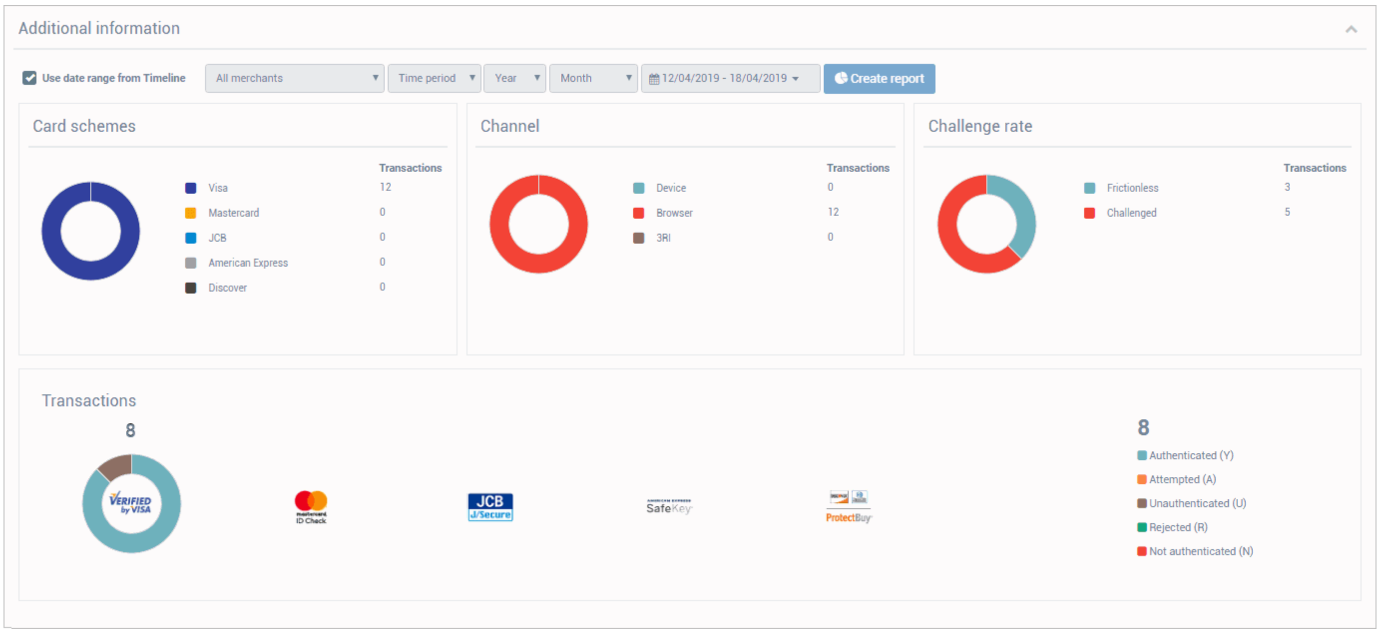
If **Use date range from Timeline** is not selected, the user will be able to specify which merchant statistics should be shown, along with the following time periods:

- `Year` - calendar year period.
- `Month` - calendar month period, in a specific year.
- `Custom` - specific date range for custom period, selected from pre-set options:
    - `Current month` - current calendar month, including current day.
    - `Current year` - current calendar year, including current day.
    - `Last month` - previous calendar month.

- `Last year` - previous calendar year.
- `Custom` - any custom period.

Graphs

Each graph shows a different subsection of data for a transaction.



Additional information

✓ Use date range from Timeline | All merchants | Time period | Year | Month | 12/04/2019 - 18/04/2019 | Create report

Card schemes

	Transactions
🟦 Visa	12
🟧 Mastercard	0
🟦 JCB	0
⬜ American Express	0
⬛ Discover	0

Channel

	Transactions
🟦 Device	0
🟥 Browser	12
🟫 3RI	0

Challenge rate

	Transactions
🟦 Frictionless	3
🟥 Challenged	5

Transactions

8

8
- 🟦 Authenticated (Y)
- 🟧 Attempted (A)
- 🟫 Unauthenticated (U)
- 🟩 Rejected (R)
- 🟥 Not authenticated (N)

Card schemes

The **Card schemes** section gives a numerical breakdown of total transactions per card scheme in the `Transaction` column.

Hover over the graph for a percentage breakdown of the total transactions per card scheme.

Channel

The **Channel** section gives a numerical breakdown of total transactions per channel used, which can give an idea on which platform a merchant is performing most of their transactions. A `Browser` entry indicates that the user was authenticated using a web-based checkout process. A `Device` entry indicates that the user was authenticated using a native mobile app during the checkout process. A `3RI` entry indicates that the merchant has performed `3DS Requestor Initiated` authentication (e.g confirming account validity or decoupled transaction).

The transactions totals are displayed in the **Transactions** column. Hover over the graph for a percentage breakdown of the channels used.

Challenge rate

The **Challenge rate** section gives a numerical breakdown of total transactions based on the challenge status of transactions. A `Frictionless` entry indicates that the cardholder was able to be authenticated by the issuer's ACS using Risk-Based Authentication (RBA) and a step up challenge was not required. A `Challenged` entry indicates that the ACS requested that the cardholder authenticate themselves. This is useful for monitoring the progress of frictionless flow capabilities over time.

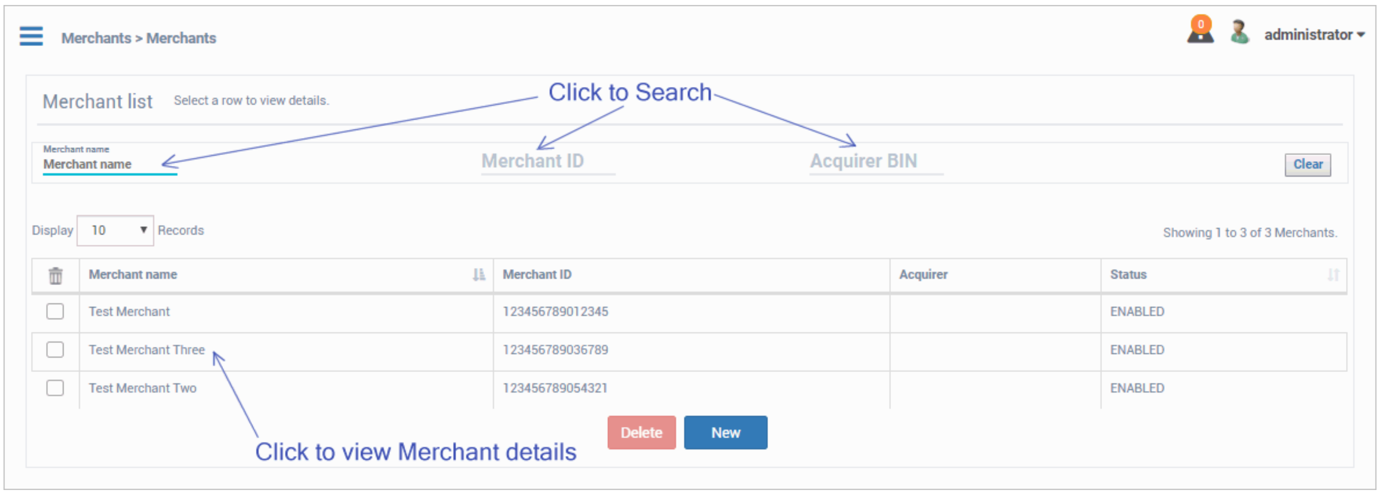
Transactions

The **Transactions** section gives a numerical breakdown of transactions and their authentication status. The column on the right hand side shows the total amount of transactions, divided into the final status response. The individual graphs show the percentage breakdown of transaction status's per card scheme. Authentication status's are described as the following: The **Transactions** section gives a numerical breakdown of transactions and their authentication status. The column on the right hand side shows the total amount of authentications, divided into the final status response. The individual graphs show the percentage breakdown of transaction status per card scheme. Authentication status is described as follows:

- **Authenticated (Y)** - Authentication verification successful.
- **Attempted (A)** - Attempts processing performed; Not authenticated/verified but a proof of attempted authentication/verification is provided.
- **Unauthenticated (U)** - Authentication/account verification could not be performed; technical or other problem.
- **Rejected (R)** - Authentication/account verification rejected; Issuer is rejecting transaction/ verification and requests that authorisation not be attempted.
- **Not Authenticated (N)** - Not authenticated/account not verified; transaction denied.

Search for merchants

A list of merchants can be accessed from the **Merchants** menu on the administration interface.



	Merchant name	Merchant ID	Acquirer	Status
☐	Test Merchant	123456789012345		ENABLED
☐	Test Merchant Three	123456789036789		ENABLED
☐	Test Merchant Two	123456789054321		ENABLED

All merchants that the user is assigned to are shown in the Merchant list by default. You can filter the list and search for specific merchants using the following parameters:

- **Merchant name** - Full or partial search on the **Merchant name** provided in the merchant profile.
- **Merchant ID** - Full or partial search on the acquirer assigned **Merchant ID** provided in the merchant profile.
- **Acquirer BIN** - Full or partial search on any of the **Acquirer BINs** provided in the merchant profile.

Search results are displayed in a table with **Merchant name**, **Merchant ID**, **Acquirer BIN** and **Enabled status** headings. Select a merchant to view or edit its details.

User Access

The **Merchants page** can only be accessed by a user that manages a merchant entity, e.g. a **Business Admin**, **Merchant Admin** or **Merchant** user.

Hint

If no merchant is displayed for a user that has either the **Merchant Admin** or **Merchant** role, double check that they have been assigned a merchant on the **User Management > User details** page.

Manage merchants

To allow merchants to make authentication requests via the authentication API, a merchant entity needs to be created and the client certificate for its 3DS Requestor downloaded. Details that are included in an authentication request, and do not change very often, are stored in the database to simplify the API functionality. The **create**, **view**, **edit** and **delete** processes for merchant entities are detailed below.

Create a merchant

To create a merchant, first head to the **Merchants** page on the administration interface and select the ***New*** button.

On the **New merchant** screen use the fields described below to create a new merchant.

User Access

A user requires the **Business admin** role to create merchants.

Important

When creating a merchant or editing a merchants details, note that the combination of **Merchant name** and **Merchant ID** must be unique.

Details

These are the general merchant details used for authentication requests:

- **Merchant name** - Merchant name assigned by the Acquirer. This should be the same name used in the authorisation message request. *Required, Maximum 40 characters*
- **Merchant ID** - Merchant identifier assigned by the Acquirer. This should be the same name used in the authorisation message request. *Required, Maximum 35 characters*

- **Country** - country that the merchant operates from. As part of an authentication request, ActiveServer will use this entry and convert it to the **Merchant Country Code** and it should match the value used in the authorisation message request. *Required*
- **Default currency** - default currency that will be used in an authentication request. This value can be overwritten in the browser based init API call by specifying the `purchaseCurrency` . *Required*
- **3DS Requestor URL** - fully qualified URL of the 3DS Requestor website or customer care site. This data element provides additional information to the receiving 3-D Secure system, if a problem arises, and should include contact information. *Required*
- **Status** - status to indicate whether the merchant is **enabled** or **disabled**. Disabling a merchant will not allow authentication API requests for that specific merchant. *Required*
- **Notes** - optional section to allow an admin user to access and edit notes for the merchant.

User Access

A user requires the **Business admin** role to view and edit the **Status** and **Notes** fields.

Card Schemes

These are the card scheme specific details used for authentication requests:

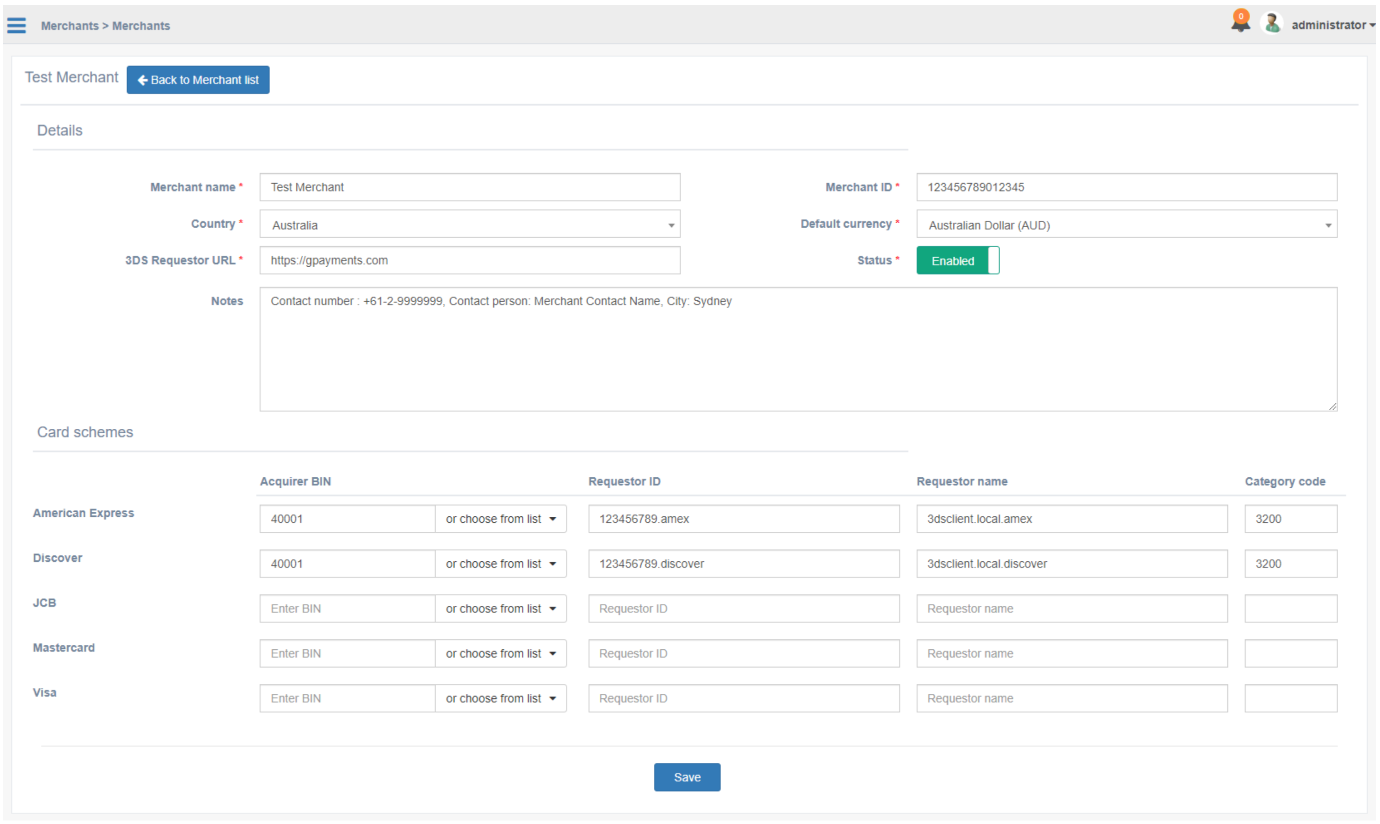
- **Acquirer BIN** - acquiring institution identification code as assigned by the DS that is receiving the AReq message. Can be entered manually or pre-filled by choosing an existing acquirer from the **drop down list**. *Maximum 11 characters*
- **Requestor ID** - DS assigned 3DS Requestor identifier. Each DS will provide a unique ID to each 3DS Requestor on an individual basis after 3DS2 merchant on boarding is complete. *Maximum 35 characters*
- **Requestor name** - DS assigned 3DS Requestor name. Each DS will provide a unique name to each 3DS Requestor on an individual basis after 3DS2 merchant on boarding is complete. *Maximum 40 characters*
- **Category code** - DS specific code describing the Merchant's type of business, product or service. *Maximum 4 characters*

Warning

All the above card scheme specific details are required to be supplied in an authentication request. If any of them are missing, the authentication request will fail.

View Merchant Details

To view merchant details, search for the merchant on the **Merchants** page of the administration interface and select the Merchant in the Merchant list. Merchant security is also managed on this page.



Test Merchant

Details

Merchant name *	Test Merchant	Merchant ID *	123456789012345
Country *	Australia	Default currency *	Australian Dollar (AUD)
3DS Requestor URL *	https://gpayments.com	Status *	Enabled
Notes	Contact number : +61-2-9999999, Contact person: Merchant Contact Name, City: Sydney		

Card schemes

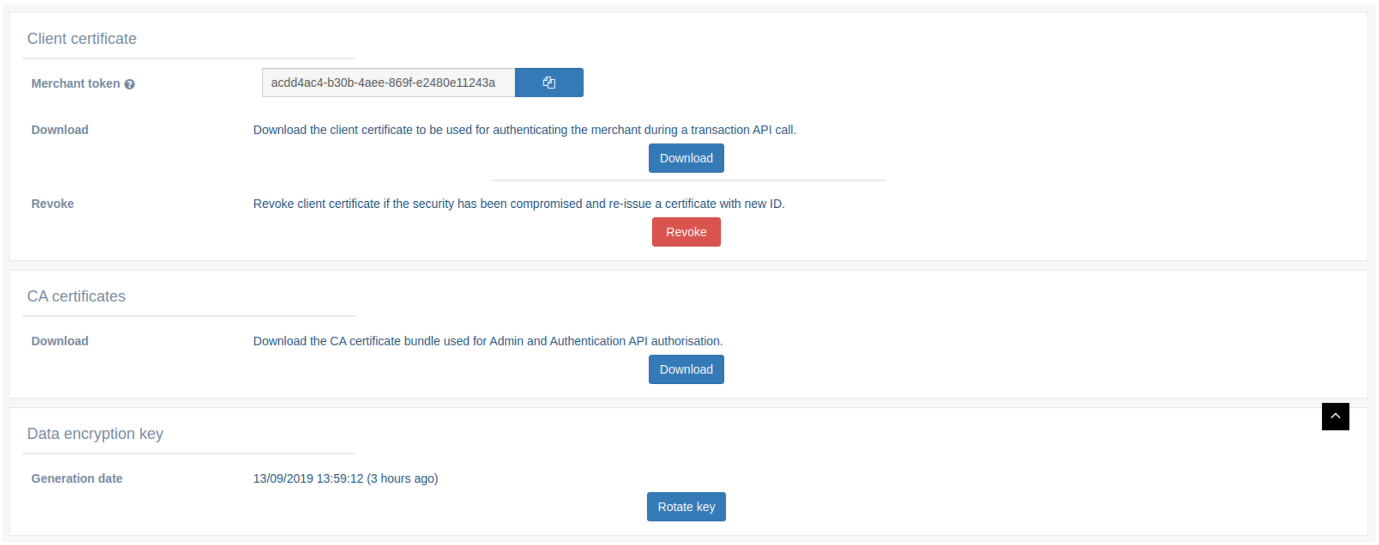
	Acquirer BIN		Requestor ID	Requestor name	Category code
American Express	40001	or choose from list	123456789.amex	3dsclient.local.amex	3200
Discover	40001	or choose from list	123456789.discover	3dsclient.local.discover	3200
JCB	Enter BIN	or choose from list	Requestor ID	Requestor name	
Mastercard	Enter BIN	or choose from list	Requestor ID	Requestor name	
Visa	Enter BIN	or choose from list	Requestor ID	Requestor name	

Merchant Security

The Merchant's **Client Certificate** and **Merchant token**, as well as the server CA certificates, can be accessed from this page. In addition, the user can manage the **Data Encryption Key** for security purposes.

Tip

Certificate management is only available once the instance has been activated.


Client certificate

Merchant token acdd4ac4-b30b-4aee-869f-e2480e11243a

Download Download the client certificate to be used for authenticating the merchant during a transaction API call.

Revoke Revoke client certificate if the security has been compromised and re-issue a certificate with new ID.

CA certificates

Download Download the CA certificate bundle used for Admin and Authentication API authorisation.

Data encryption key

Generation date 13/09/2019 13:59:12 (3 hours ago)

Client Certificate

The 3DS Requestor client certificate is required for a merchant to include in the authentication API requests for SSL authentication:

- **Merchant token** - token to be added in HTTP header of an authentication API request. Only required if using a Master Auth API client certificate to authenticate on behalf of a merchant. This field will only be visible to a user with a **Business admin** role, as this is the only role with access to the Master Auth API client certificate. For information on using the master certificate with the merchant token, refer here.
- **Download** - allows the user to download the 3DS Requestor client certificate in a **.p12** format after specifying a password. For more information on this functionality, see the API document overview.
- **Revoke** - disables the current 3DS Requestor client certificate if there has been a security breach or lost certificate, then re-issues a new certificate which can be downloaded and provided to the merchant.

Warning

Revoking a client certificate will invalidate all instances of the certificate, and the merchant will not be able to initiate API requests until the replacement certificate can be installed.

CA Certificates

- **Download** - allows the download of the servers CA certificates, to be used in authentication API requests. For more information on this functionality, see the API document overview.

> **Version 1.0.5**
>
> CA certificate download was added in the version 1.0.5 release.

> **User Access**
>
> A user requires the **Business admin**, **Merchant admin** or **Merchant** role to download a certificate.
> A user requires the **Business admin** or **Merchant admin** role to revoke a certificate.

Data encryption key

There is a key assigned for every merchant which **ActiveServer** uses to encrypt the requests and responses for all authentications prior to saving them in the database. This key is also used to decrypt the account number used for the transaction when searching for transactions.

- **Rotate key** - used for changing the current data encryption key, in use, if required, e.g. for internal or external policies requiring the rotation of encryption keys. Old key will still be available to be used for decrypting/encrypting the old transactions. New key will be used for transactions performed after the rotation.

> **User Access**
>
> A user requires the **Business admin** or **Merchant admin** role to rotate a key.

Edit merchant details

To edit a merchant, view its profile and edit its available fields.

The merchant profile details available are specific to user roles:

- **Status** - the enabled status is only available to users with the **Business admin** role.
- **Notes** - the notes section is only available to users with the **Business admin** role.

User Access

A user requires the **Business admin**, **Merchant admin** or **Merchant** role to view merchant details.
A user requires the **Business admin** or **Merchant admin** role to edit merchant details.

Delete a merchant

To delete a merchant, first head to the **Merchants** page on the administration interface, search for the merchant and select the **delete check box** adjacent to the Merchant name, in the search result table. Select the ***Delete*** button and confirm on the dialogue box.

Important

The default **Test Merchant** cannot be deleted, as it is used for testing purposes.

User Access

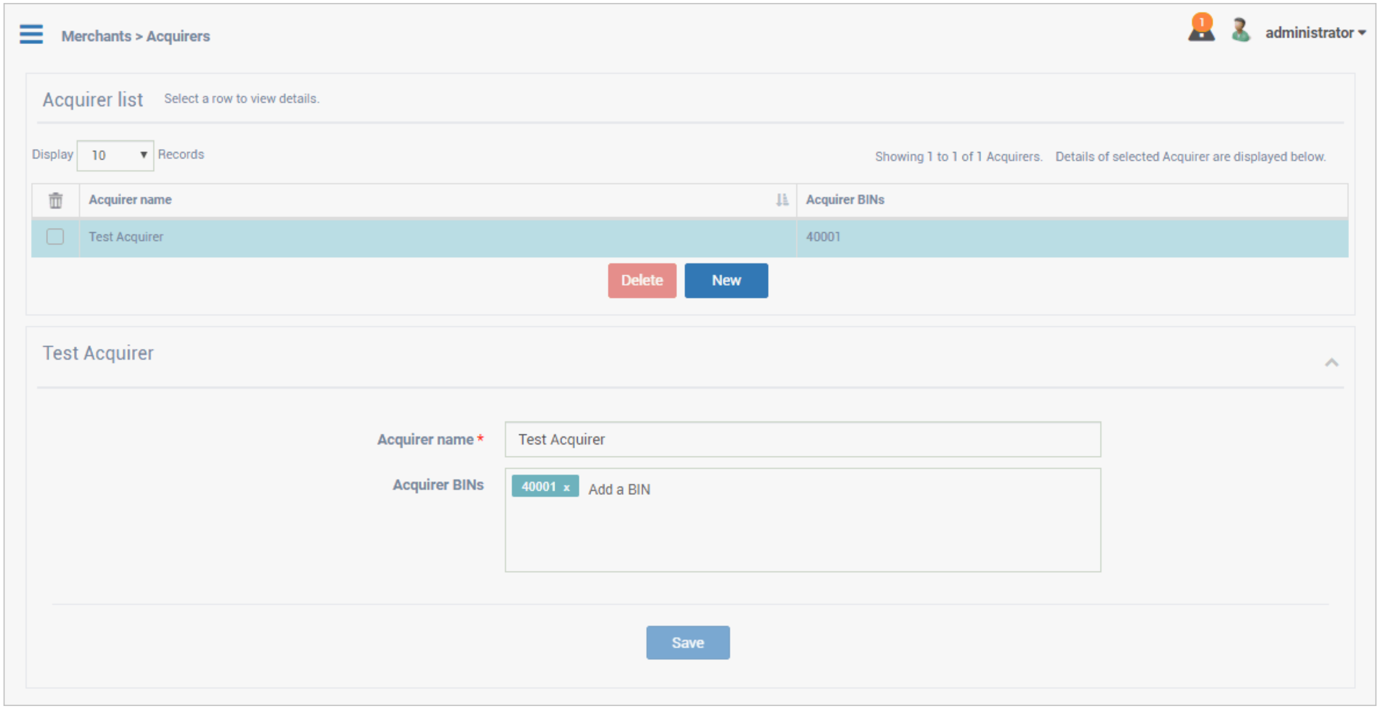
A user requires the **Business admin** role to delete merchants.

Manage acquirers

A list of all acquirers can be accessed from the **Merchants > Acquirers** menu on the administration interface. The list shows **Acquirer name** and all associated **Acquirer BINs**.

User Access

A user requires the **Business admin** role to create, view, edit and delete Acquirers.


Merchants > Acquirers
administrator
Acquirer list
Select a row to view details.
Display 10 Records
Showing 1 to 1 of 1 Acquirers. Details of selected Acquirer are displayed below.
Delete
New
Save

	Acquirer name	Acquirer BINs
☐	Test Acquirer	40001

Test Acquirer

Acquirer name *	Test Acquirer
Acquirer BINs	40001 x Add a BIN

Create an acquirer

To **create** an acquirer, select the ***New*** button and fill in the fields:

- **Acquirer name** - name used to identify the acquirer in ActiveServer. Not used for authentication messaging.
- **Acquirer BINs** - one or more BINs that can be assigned to the acquirer. This field is sent in authentication messaging and should be the same one used when enrolling a merchant to the payment system DS.

Select the ***Create*** button to create the acquirer.

View and edit acquirer details

Select an acquirer from the acquirer list to view and edit its details. Make changes to the details, as required, and select the ***Save*** button.

Delete an acquirer

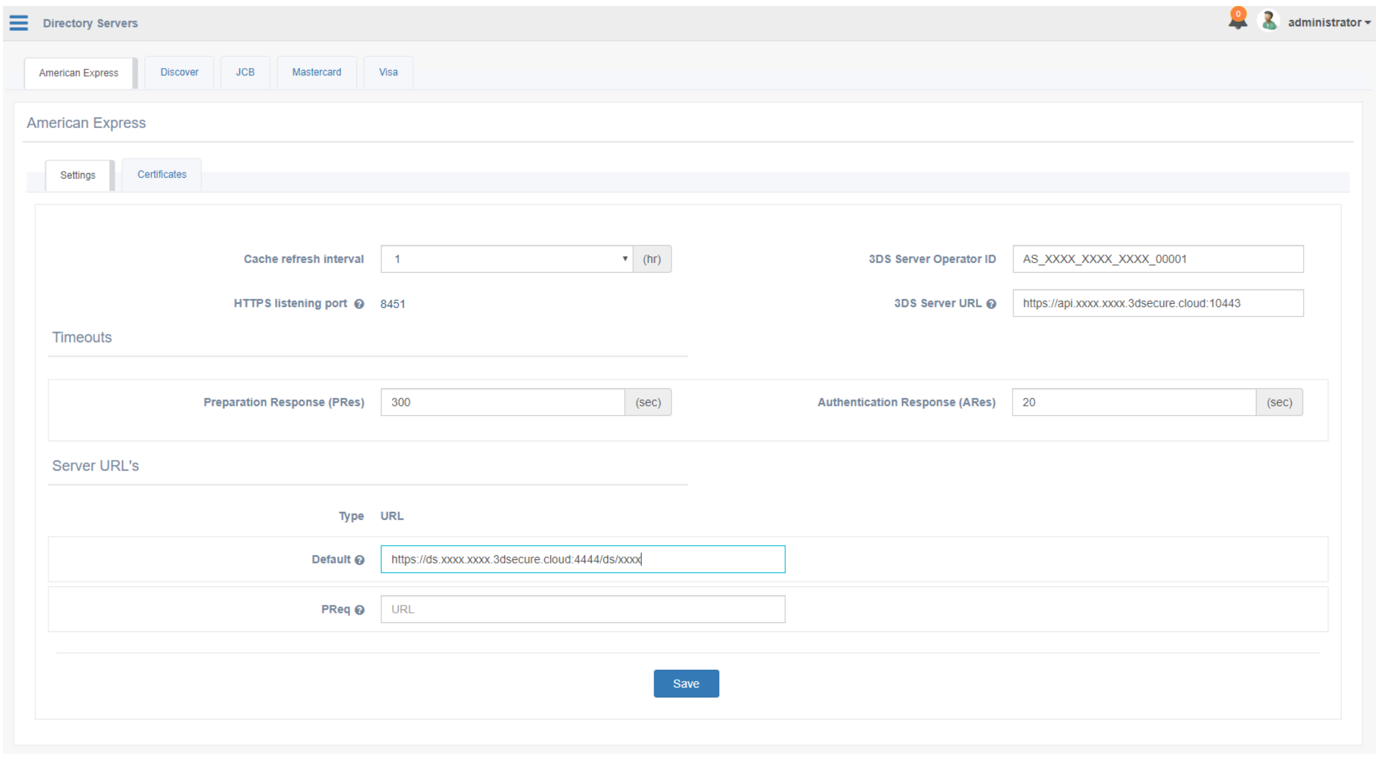
To delete an acquirer, select the adjacent **delete** check box in the acquirer list. Select the ***Delete*** button and confirm on the dialogue box.

Manage DS settings

All the settings for card schemes supported by ActiveServer can be managed from the **Directory Servers** page under the **Settings** tab.

User Access

A user requires the **System admin** role to manage DS settings.



American Express

Timeouts

Server URL's

Cache refresh interval	1 (hr)	3DS Server Operator ID	AS_XXXX_XXXX_XXXX_00001
HTTPS listening port	8451	3DS Server URL	https://api.xxxx.xxxx.3dsecure.cloud:10443
Preparation Response (PRes)	300 (sec)	Authentication Response (ARes)	20 (sec)

Type	URL
Default	https://ds.xxxx.xxxx.3dsecure.cloud:4444/ds/xxxx
PReq	URL

To edit a card scheme's settings:

Select the appropriate **card scheme** tab at the top of the page to display its details.

Settings

The following settings can be edited:

- **Cache refresh interval** - interval in which the PRes cache refreshes for all available card schemes. The PReq/PRes messages are utilised by **ActiveServer** to cache information about the Protocol Version Numbers(s) supported by available ACSs, the DS, and also any

URL to be used for the 3DS Method call. The data will be organised by card range as configured by a DS. The information provided on the Protocol Version Number(s) supported by ACSs and the DS can be utilised in the App-based, Browser-based and 3RI flows. It is a 3DS2 specification requirement that this exchanges happens at least once every 24 hours and at most every hour. (unit: hours)

- **3DS Server Operator ID** - DS assigned 3DS Server identifier. Each DS can provide a unique ID to each 3DS Server on an individual basis, usually during or at the end of the card scheme compliance process. Requirements for the presence of this field in the AReq and PReq messages are DS specific.
- **HTTPS listening port** - port that AS is listening on for HTTPS communications during authentications. This value is read from the configuration file for the corresponding DS setting and is used to listen for RReq requests.
- **3DS Server URL** - fully qualified URL of the 3DS Server to which the DS will send the RReq message after the challenge has completed. URL is fully customisable to your environment/ load balancing setup and can be different to the server external URL.

Tip

The **3DS Server URL** may have already been set when updating the External URL.

Timeouts

The following settings can be edited:

- **Preparation Response (PRes)** - timeout interval for the PRes message
- **Authentication Response (ARes)** - timeout interval for the ARes message

Server URLs

In the **Server URLs** section, the DS URLs can be input for the addresses of the card scheme DS.

The **Default** URL is used for AReq and PReq message communications with the DS provider.

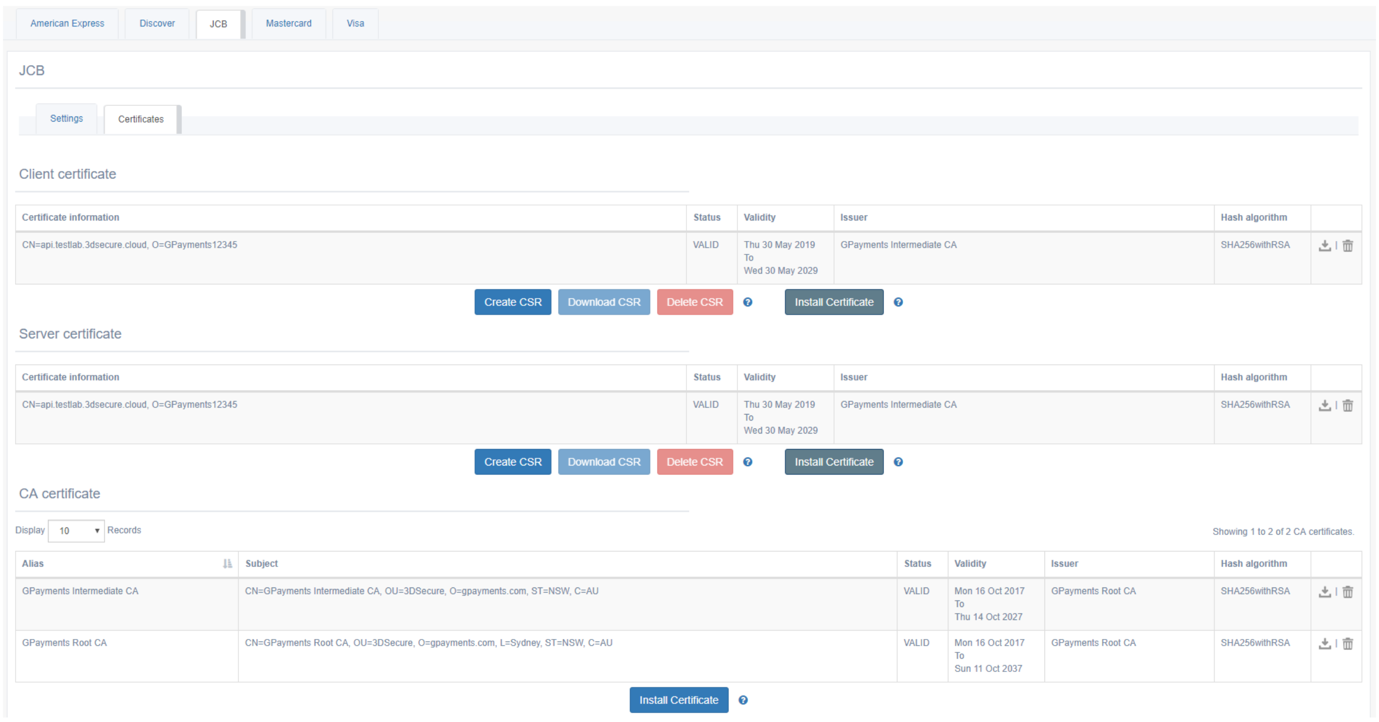
The **PReq** URL can **optionally** be filled in if the DS provider has a separate PReq endpoint for those requests. If a value is entered in this field, it will override the **Default** URL. If this value is empty, **ActiveServer** will send PReq messages to the **Default** URL if it entered.

Manage DS certificates

All the certificates for card schemes supported by ActiveServer can be managed from the **Directory Servers** page, under the **Certificates** tab.

User Access

A user requires the **System admin** role to manage DS certificates.



Client certificate

Certificate information	Status	Validity	Issuer	Hash algorithm
CN=api.testlab.3dsecure.cloud, O=GPayments12345	VALID	Thu 30 May 2019 To Wed 30 May 2029	GPayments Intermediate CA	SHA256withRSA

Server certificate

Certificate information	Status	Validity	Issuer	Hash algorithm
CN=api.testlab.3dsecure.cloud, O=GPayments12345	VALID	Thu 30 May 2019 To Wed 30 May 2029	GPayments Intermediate CA	SHA256withRSA

CA certificate

Display 10 Records

Showing 1 to 2 of 2 CA certificates.

Alias	Subject	Status	Validity	Issuer	Hash algorithm
GPayments Intermediate CA	CN=GPayments Intermediate CA, OU=3DSecure, O=gpayments.com, ST=NSW, C=AU	VALID	Mon 16 Oct 2017 To Thu 14 Oct 2027	GPayments Root CA	SHA256withRSA
GPayments Root CA	CN=GPayments Root CA, OU=3DSecure, O=gpayments.com, L=Sydney, ST=NSW, C=AU	VALID	Mon 16 Oct 2017 To Sun 11 Oct 2037	GPayments Root CA	SHA256withRSA

To manage a card scheme's certificates:

Select the appropriate **card scheme** tab at the top of the page to display its details.

Client and server certificates

In a 3DS2 authentication, both inbound and outbound traffic with the DS is required to be mutually authenticated, meaning **ActiveServer** will act as both a HTTPS **client** and a **server** in different processes.

ActiveServer acts as a client when it connects to the card scheme's DS to send the initial AReq. The **client certificate** is used to verify and authenticate **ActiveServer** in this flow, with this connection also being used to receive the resulting ARes from the DS when it is ready.

If an authentication requires a challenge, **ActiveServer** will act as a server when it is time for the DS to notify us of the authentication results in the RReq and respond with the RRes.

This certificate/s is downloaded from the card scheme, usually after providing a Certificate Signing Request (CSR).

Each certificate has the following table information displayed:

- **Certificate information** - certificate details of the installed certificate.
- **Status** - displays whether the certificate has been signed by a trusted CA. **Valid** indicates the CA Certificate has been successfully installed in **ActiveServer**, whereas **Not Valid** indicates no associated CA Certificate can be found.
- **Validity** - start and end dates for certificate validity.
- **Issuer** - name of the CA issuer that signed the certificate.
- **Hash algorithm** - hash algorithm used for the certificate signature.
- **Export | Delete** - Export or delete a certificate.

The following processes can be carried out for client certificate management:

Create CSR

To assist with the generation of a CSR, **ActiveServer** provides this functionality via the ***Create CSR*** button. However, it also possible to do this process manually if you prefer, using an external method such as ***OpenSSL***.

The certificate content should be filled in as appropriate for the card schemes requirements, with the following options available:

- **Key size** - key size of the request, measured in bits .
- **Common Name** – hostname that will use the certificate, usually a fully-qualified domain name. For the server certificate, this must be the hostname of **ActiveServer**. For the client certificate, this will usually be the same as the sever certificate, however some card schemes may have different requirements. This value will be pre-filled with the **3DS Server URL** setting of the DS if it available.

- **Organization** – legal name of your company or organization.
- **Organization Unit** – departmental or division name for your group.
- **City** – city where your company is located.
- **Province** – province or state where your company is located.
- **Two letter country code** – two-character abbreviation for your country.
- **Hash algorithm** - hash algorithm used to sign the CSR.

Creating a CSR will generate the CSR content and an associated private key to be stored in the database.

Important

Only one CSR for the client certificate and one CSR for server certificates can be stored at a time, per card scheme.

Download CSR

Download CSR will export the CSR content to a `.csr` file in the following file name pattern: *"Common Name"_"Card Scheme".csr*, e.g. `api.testlab.3dsecure.cloud_JCB.csr`.

Download CSR will only be available if a CSR has already been created in the system.

Delete CSR

Delete CSR will delete both the CSR content and associated private key from the system.

Delete CSR will only be available if a CSR has already been created in the system.

Warning

Deleting a CSR is permanent and will invalidate any outstanding CSRs waiting to be signed, meaning they will be unable to be installed.

Install Certificate

Install Certificate will install a signed certificate, either using the raw **certificate content** or **certificate file**.

The certificate can be in one of the following formats: `.pfx`, `.p7b`, `.p12`, `.jks`, `.pem`. **ActiveServer** will try each file type one by one when installing. If the file type requires a password, such as `.p12`, enter the password on the install certificate page.

ActiveServer will attempt to use a private key if it is included with a `.pfx`, `.p12` or `.jks` file, otherwise it will use an existing private key created by the system if it is available. See below for information on creating your own private key.

If the card scheme provider only requires one certificate for client and server connections, the **Server certificate is the same as the client certificate** option can be selected. This will install the certificate to both the client and server sections.

Important

To ensure the correct certificate type is being installed to the correct section, **ActiveServer** will check the certificate extension **X509v3 Extended Key Usage** and perform the following checks:

- Client certificate - must have the value **TLS Web Client Authentication**. If the value **TLS Web Server Authentication** is missing, the *Server certificate is the same as the client certificate* option will be disabled.
- Server certificate - must have the value **TLS Web Server Authentication**

Install certificate will save the certificate to the database, then remove any outstanding CSR content from the system, as well as the local private key if an external private key was provided. After this a new CSR and private key are able to be created if required.

Important: Restart required for new certificates

To use newly installed certificates, the **ActiveSever** instance **must be restarted** to refresh the DS connectors.

Warning

It is only possible to have **one** client and **one** server certificate at a time, **installing** another certificate will cause the current certificate and private key to be overwritten.

Export and delete

Client and server certificates are able to be exported for backup purposes or deleted if required by selecting the relevant icon from the certificate table.

Certificates can be **Exported** in two formats:

- **PKCS12 keystore (.p12 including private key)** - creates a `.p12` file including the certificate and associated private key. Optionally a password can be included for the file.
- **Certificate only (.pem)** - creates a `.pem` file only containing the certificate.

Delete will show a prompt, asking the user to confirm deletion of the certificate, before removing it from the system.

Warning

Deleting the certificate is permanent, with exporting the certificate as a backup being recommended first.

CA Certificate

CA Certificates are used to verify the CA signer of the server/client certificate for this provider and ensure they are from a valid CA. CA certificates will be automatically added if found in the certificate chain during server/client certificate upload, otherwise they should be added manually to validate installed certificates.

If an associated CA has not been installed, the **Certificate Status** of the client or server certificate will be **Not valid**. Deleting a CA will also invalidate previously installed certificates.

The ***Install Certificate*** button will show a prompt to search for a local certificate file. Certificate information and functionality shown is the same as described in the Client and Server certificate section, with the addition of the CA Certificate **Alias** value.

Using external tool for certificate management

You can use your chosen tool for generating a CSR and private key. An example using **OpenSSL** is provided below.

Ensure **OpenSSL** is installed and open a terminal to perform the following:

1. Create a RSA Private Key

   ```
   openssl genrsa -out privateKey.key 2048
   ```

2. Generate the CSR and follow the prompts to enter CSR details

   ```
   openssl req -new -key privateKey.key -out yourCSR.csr
   ```

3. Once the CSR has been signed by the card scheme, combine the provided signed certificate and card scheme CA certificate chain with the generated private key

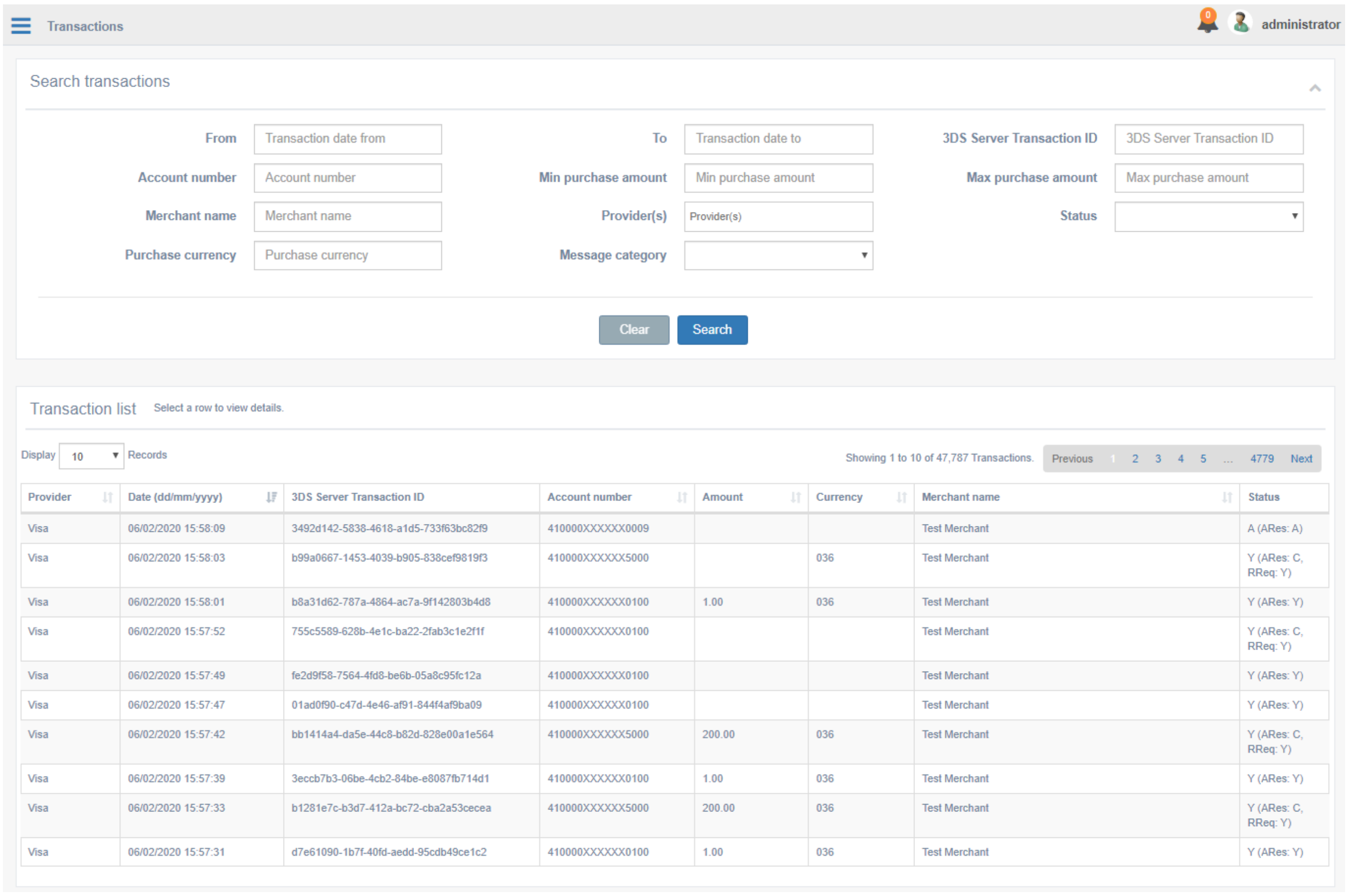
   ```
   openssl pkcs12 -export -out certifcate.p12 -inkey privateKey.key -in
   signedcertificate.crt -certfile ca-chain.crt
   ```

4. Install the certificate as normal

View transactions

Transactions can be accessed from the left menu and has 4 sections:

- Search Transactions
- Transaction List
- Transaction Details
- Transaction Messages



From	Transaction date from	To	Transaction date to	3DS Server Transaction ID	3DS Server Transaction ID
Account number	Account number	Min purchase amount	Min purchase amount	Max purchase amount	Max purchase amount
Merchant name	Merchant name	Provider(s)	Provider(s)	Status	
Purchase currency	Purchase currency	Message category			

Transaction list Select a row to view details.

Display 10 Records

Showing 1 to 10 of 47,787 Transactions.

Provider	Date (dd/mm/yyyy)	3DS Server Transaction ID	Account number	Amount	Currency	Merchant name	Status
Visa	06/02/2020 15:58:09	3492d142-5838-4618-a1d5-733f63bc82f9	410000XXXXXX0009			Test Merchant	A (ARes: A)
Visa	06/02/2020 15:58:03	b99a0667-1453-4039-b905-838cef9819f3	410000XXXXXX5000		036	Test Merchant	Y (ARes: C, RReq: Y)
Visa	06/02/2020 15:58:01	b8a31d62-787a-4864-ac7a-9f142803b4d8	410000XXXXXX0100	1.00	036	Test Merchant	Y (ARes: Y)
Visa	06/02/2020 15:57:52	755c5589-628b-4e1c-ba22-2fab3c1e2f1f	410000XXXXXX0100			Test Merchant	Y (ARes: C, RReq: Y)
Visa	06/02/2020 15:57:49	fe2d9f58-7564-4fd8-be6b-05a8c95fc12a	410000XXXXXX0100			Test Merchant	Y (ARes: Y)
Visa	06/02/2020 15:57:47	01ad0f90-c47d-4e46-af91-844f4af9ba09	410000XXXXXX0100			Test Merchant	Y (ARes: Y)
Visa	06/02/2020 15:57:42	bb1414a4-da5e-44c8-b82d-828e00a1e564	410000XXXXXX5000	200.00	036	Test Merchant	Y (ARes: C, RReq: Y)
Visa	06/02/2020 15:57:39	3eccb7b3-06be-4cb2-84be-e8087fb714d1	410000XXXXXX0100	1.00	036	Test Merchant	Y (ARes: Y)
Visa	06/02/2020 15:57:33	b1281e7c-b3d7-412a-bc72-cba2a53cecea	410000XXXXXX5000	200.00	036	Test Merchant	Y (ARes: C, RReq: Y)
Visa	06/02/2020 15:57:31	d7e61090-1b7f-40fd-aedd-95cdb49ce1c2	410000XXXXXX0100	1.00	036	Test Merchant	Y (ARes: Y)

Search transactions

This section allows you to search through the database for specific transactions. You can filter through transactions by entering information about the transaction. Fields include:

- **From** - only include transactions on and after the specified date
- **To** - only include transactions on and before the specified date

- **3DS Server Transaction ID** - only include transactions with the specified Transaction ID
- **Account number** - only include transactions by cardholder account number (may be represented by PAN or token)
- **Minimum purchase amount** - only include transactions above the specified amount
- **Maximum purchase amount** - only include transactions below the specified amount
- **Merchant name** - only include transactions processed by the specified merchant, can be all or part of the merchant name
- **Provider(s)** - only include transactions processed by the specified card scheme or schemes
- **Status** - only include transactions with the specified result status (e.g. "Y" for `Transaction Successful`, etc.)
- **Purchase currency** - only include transactions performed in the specified currency, defined by currency code
- **Message category** - only include transactions that are either PA (Payment) or NPA (Non-Payment)

Once you have set the desired filters, click **Search** to view results in the **Transaction List** below. Click **Clear** to reset the fields.

Transaction list

Transaction list displays all transactions or a filtered list if you have selected any of the search parameters above.

Transaction details displayed are:

- **Provider** - card scheme used for transaction
- **Date** - date and time that the transaction was processed
- **3DS Server Transaction ID** - 3DS Server transaction ID of the particular transaction
- **Account number** - full account number used in the transaction, may be represented by PAN or token

Important

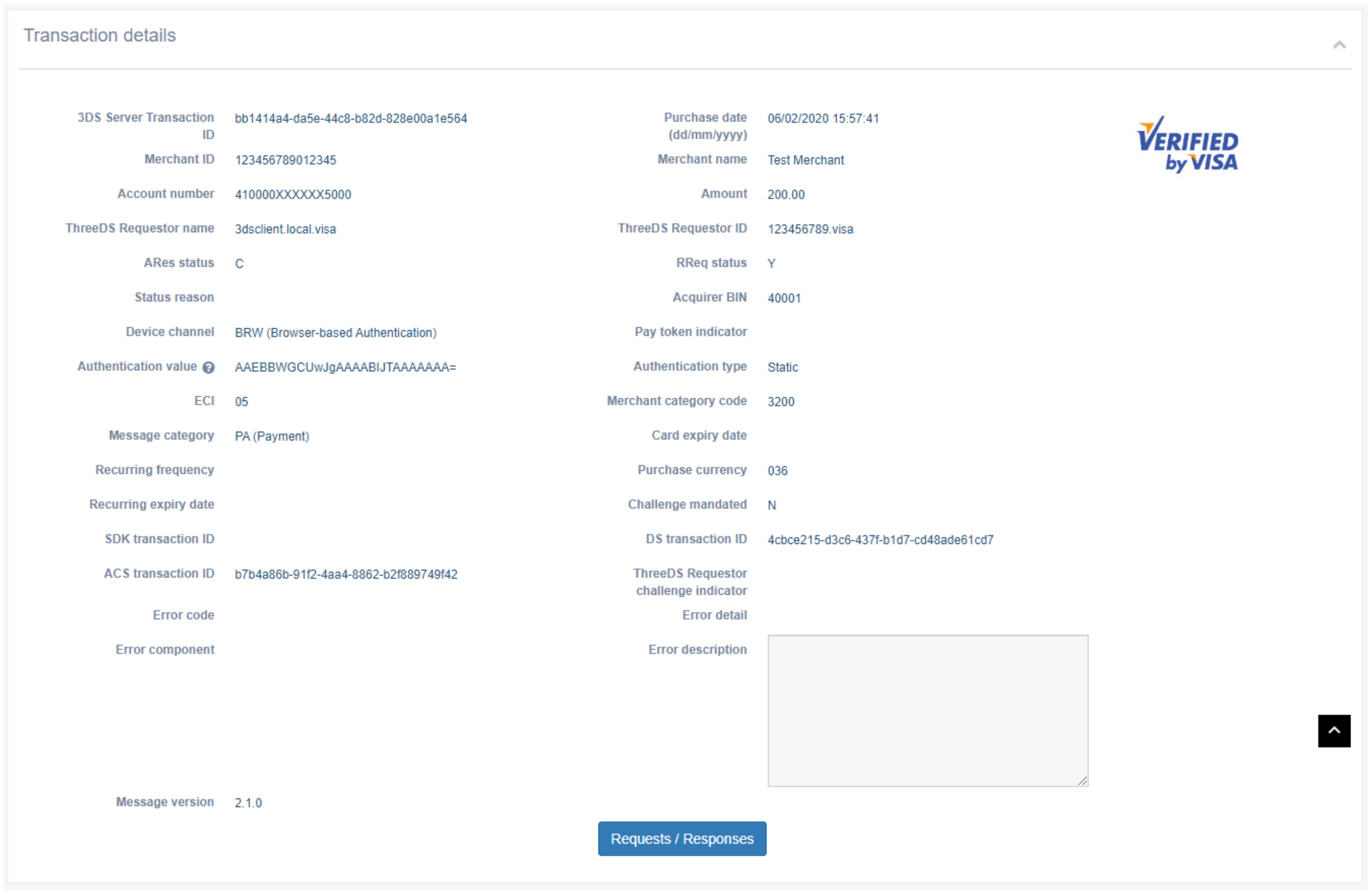
For transactions made using **Auth API v1**, a full PAN is stored encrypted in the database. For **Auth API v2** transactions, PANs are stored in a truncated format (only first 6 and last 4 digits are stored, remaining digits are masked). Therefore, depending on the API version used, some results may be only partially matched based on the **account number** search value. It is recommended to use the other search parameters to narrow down search results if required.

- **Amount** - purchase amount of the transaction
- **Currency** - currency code that the transaction was processed in
- **Merchant name** - name of the merchant processing the transaction
- **Status** - authentication result returned for the transaction, using the formats:
    - **Frictionless (i.e. no challenge)** - "*Final status* (ARes: *ARes status*)", e.g. "Y (ARes: Y)"
    - **Challenge** - "*Final status* (ARes: *ARes status*", RReq: *RReq status*), e.g. "N (ARes: C, RReq: N)"

Select any transaction row to view its details in the Transaction details section below.

Transaction Details

Transaction details displays the details for the transaction selected in the Transaction list.


Transaction details

3DS Server Transaction ID	bb1414a4-da5e-44c8-b82d-828e00a1e564	Purchase date (dd/mm/yyyy)	06/02/2020 15:57:41
Merchant ID	123456789012345	Merchant name	Test Merchant
Account number	410000XXXXXX5000	Amount	200.00
ThreeDS Requestor name	3dsclient.local.visa	ThreeDS Requestor ID	123456789.visa
ARes status	C	RReq status	Y
Status reason		Acquirer BIN	40001
Device channel	BRW (Browser-based Authentication)	Pay token indicator	
Authentication value	AAEBBWGCUwJgAAAABIJTAAAAAAA=	Authentication type	Static
ECI	05	Merchant category code	3200
Message category	PA (Payment)	Card expiry date	
Recurring frequency		Purchase currency	036
Recurring expiry date		Challenge mandated	N
SDK transaction ID		DS transaction ID	4cbce215-d3c6-437f-b1d7-cd48ade61cd7
ACS transaction ID	b7b4a86b-91f2-4aa4-8862-b2f889749f42	ThreeDS Requestor challenge indicator	
Error code		Error detail	
Error component		Error description	
Message version	2.1.0		

The Transaction details displayed are:

- **3DS Server Transaction ID** - universally unique transaction identifier assigned by the 3DS Server to identify a single transaction
- **Purchase date** - date and time of the purchase
- **Merchant ID** - acquirer assigned merchant identifier. This should be the same value that is used in authorisation requests sent on behalf of the 3DS Requestor
- **Merchant name** - merchant name assigned by the acquirer or payment system
- **Account number** - account number used in the authorisation request for payment transactions, may be represented by PAN or token. The first 6 and last 4 digits of the account number are shown, the remaining digits are masked with a `X` character
- **Amount** - purchase amount of the transaction
- **ThreeDS Requestor name** - DS assigned 3DS Requestor name. Each DS will provide a unique name to each 3DS Requestor on an individual basis
- **ThreeDS Requestor ID** - DS assigned 3DS Requestor identifier. Each DS will provide a unique ID to each 3DS Requestor on an individual basis

- **ARes/RReq status** - indicates whether a transaction qualifies as an authenticated transaction or account verification or not. Possible values are:
    - **Authenticated (Y)** - authentication verification successful
    - **Not Authenticated (N)** - not authenticated/account not verified; transaction denied
    - **Unauthenticated (U)** - authentication/account verification could not be performed; technical or other problem
    - **Attempted (A)** - attempts processing performed; Not authenticated/verified, but a proof of attempted authentication/verification is provided
    - **Challenge Required** - additional authentication is required using the CReq/CRes
    - **Rejected (R)** - authentication/account verification rejected; Issuer is rejecting transaction/verification and requests that authorisation not be attempted
- **Status reason** - Provides information on why the Transaction Status field has the specified value. For Payment channel, required if the Transaction Status field = N, U, or R. For Non-payment channel, it is Conditional as defined by the DS. Possible values are:
    - **01** - Card authentication failed
    - **02** - Unknown Device
    - **03** - Unsupported Device
    - **04** - Exceeds authentication frequency limit
    - **05** - Expired card
    - **06** - Invalid card number
    - **07** - Invalid transaction
    - **08** - No Card record
    - **09** - Security failure
    - **10** - Stolen card
    - **11** - Suspected fraud
    - **12** - Transaction not permitted to cardholder
    - **13** - Cardholder not enrolled in service
    - **14** - Transaction timed out at the ACS
    - **15** - Low confidence
    - **16** - Medium confidence

    - **17** - High confidence
    - **18** - Very High confidence
    - **19** - Exceeds ACS maximum challenges
    - **20** - Non-Payment transaction not supported
    - **21** - 3RI transaction not supported
    - **22–79** - Reserved for EMVCo future use (values invalid until defined by EMVCo)
    - **80–99** - Reserved for DS use
- **Acquirer BIN** - Acquiring institution identification code for the merchant's bank
- **Device channel** - Indicates the channel from which the transaction originated. Possible values are:
    - App-based (01-APP)
    - Browser-based (02-BRW)
    - 3DS Requestor Initiated (03-3RI)
- **Pay token indicator** - value of **True** indicates that the transaction was de-tokenised prior to being received by the ACS. This data element will be populated by the system residing in the 3-D Secure domain where the de-tokenisation occurs (i.e. the 3DS Server or the DS). The Boolean value of true is the only valid response for this field when it is present
- **Authentication value** - payment system-specific value provided as part of the ACS registration for each supported DS. Authentication Value may be used to provide proof of authentication, e.g. during authorisation or clearing. This value is stored for a period of 7 days, after which it is masked.
- **Authentication type** - indicates the type of authentication method the Issuer will used to challenge the cardholder if challenge is required, whether in the ARes message or what was used by the ACS when in the RReq message. Possible values are:
    - **01** - Static
    - **02** - Dynamic
    - **03** - OOB
    - **04–79** - Reserved for EMVCo future use (values invalid until defined by EMVCo)
    - **80–99** - Reserved for DS use
- **ECI** - electronic commerce indicator value for the transaction, which identifies status of cardholder authentication for card schemes

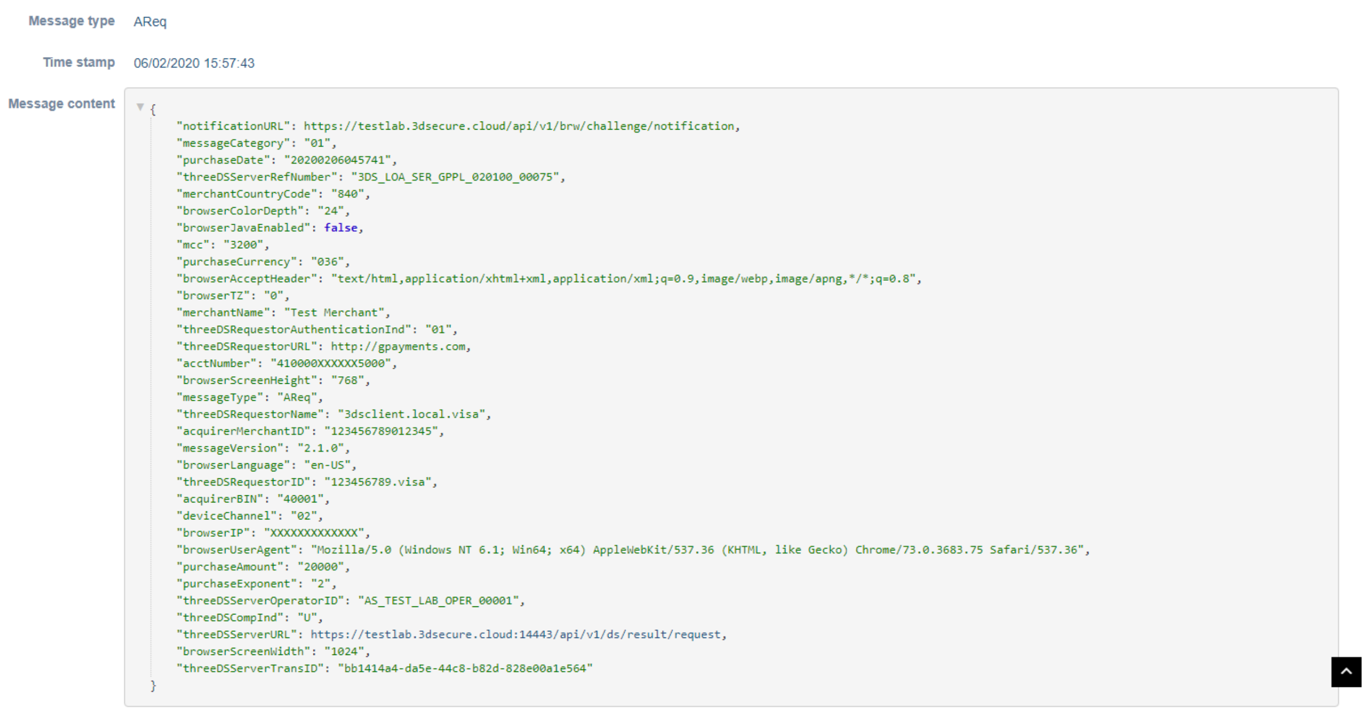
- **Merchant category code** - DS specific code describing the Merchant's type of business, product or service
- **Message category** - identifies the category of the transaction. Possible values are:
    - **01** - PA (Payment)
    - **02** - NPA (Non-Payment)
    - **03–79** - Reserved for EMVCo future use (values invalid until defined by EMVCo)
    - **80-99** - Reserved for DS use
- **Card expiry date** - expiry date of the PAN or token supplied to the 3DS Requestor by the cardholder in **YYMM** format
- **Recurring frequency** - for a recurring transaction, indicates the minimum number of days between authorisations
- **Purchase currency** - currency code in which purchase amount is expressed
- **Recurring expiry date** - for a recurring transaction, date after which no further authorisations shall be performed, in **YYYYMMDD** format
- **Challenge mandated** - indication of whether a challenge is required for the transaction to be authorised due to local/regional mandates or other variable, possible values are:
    - **Y** - Challenge is mandated
    - **N** - Challenge is not mandated
- **SDK transaction ID** - universally unique transaction identifier assigned by the 3DS SDK to identify a single transaction
- **DS transaction ID** - universally unique transaction identifier assigned by the DS to identify a single transaction
- **ACS transaction ID** - universally unique transaction identifier assigned by the ACS to identify a single transaction
- **ThreeDS requestor challenge indicator** - indicates whether a challenge is requested by the merchant for this transaction. For example:
    - For 01-PA, a 3DS Requestor may have concerns about the transaction, and request a challenge.
    - For 02-NPA, a challenge may be necessary when adding a new card to a wallet.
    - For local/regional mandates or other variables.

- Possible values are:
    - **01** - No preference
    - **02** - No challenge requested
    - **03** - Challenge requested: 3DS Requestor Preference
    - **04** - Challenge requested: Mandate
    - **05–79** - Reserved for EMVCo future use (values invalid until defined by EMVCo)
    - **80-99** - Reserved for DS use
    - Note: If the element is not provided, the expected action is that the ACS would interpret as 01 = No preference.

- **Error code** - if available, code indicating the type of problem identified in the message
- **Error detail** - if available, additional detail regarding the problem identified in the message
- **Error component** - if available, code indicating the 3-D Secure component that identified the error. Possible values are:
    - **C** - 3DS SDK
    - **S** - 3DS Server
    - **D** - DS
    - **A** - ACS
- **Error description** - if available, text describing the problem identified in the message
- **Message version** - protocol version identifier utilised by the 3DS Server, set in the AReq message. The message version Number does not change during a 3DS transaction

Transaction Messages

To view the 3DS2 messages for a transaction, click `Requests / Responses` at the bottom of the Transactions section.



Message type AReq

Time stamp 06/02/2020 15:57:43

Message content

```
{
    "notificationURL": https://testlab.3dsecure.cloud/api/v1/brw/challenge/notification,
    "messageCategory": "01",
    "purchaseDate": "20200206045741",
    "threeDSServerRefNumber": "3DS_LOA_SER_GPPL_020100_00075",
    "merchantCountryCode": "840",
    "browserColorDepth": "24",
    "browserJavaEnabled": false,
    "mcc": "3200",
    "purchaseCurrency": "036",
    "browserAcceptHeader": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "browserTZ": "0",
    "merchantName": "Test Merchant",
    "threeDSRequestorAuthenticationInd": "01",
    "threeDSRequestorURL": http://gpayments.com,
    "acctNumber": "410000XXXXXX5000",
    "browserScreenHeight": "768",
    "messageType": "AReq",
    "threeDSRequestorName": "3dsclient.local.visa",
    "acquirerMerchantID": "123456789012345",
    "messageVersion": "2.1.0",
    "browserLanguage": "en-US",
    "threeDSRequestorID": "123456789.visa",
    "acquirerBIN": "40001",
    "deviceChannel": "02",
    "browserIP": "XXXXXXXXXXXXX",
    "browserUserAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36",
    "purchaseAmount": "20000",
    "purchaseExponent": "2",
    "threeDSServerOperatorID": "AS_TEST_LAB_OPER_00001",
    "threeDSCompInd": "U",
    "threeDSServerURL": https://testlab.3dsecure.cloud:14443/api/v1/ds/result/request,
    "browserScreenWidth": "1024",
    "threeDSServerTransID": "bb1414a4-da5e-44c8-b82d-828e00a1e564"
}
```

- **Message type** - 3DS message type, e.g. AReq, ARes, RReq, RRes
- **Time stamp** - date and time the message was sent or received.
- **Message content** - raw JSON message content that was sent or received.

Manage nodes

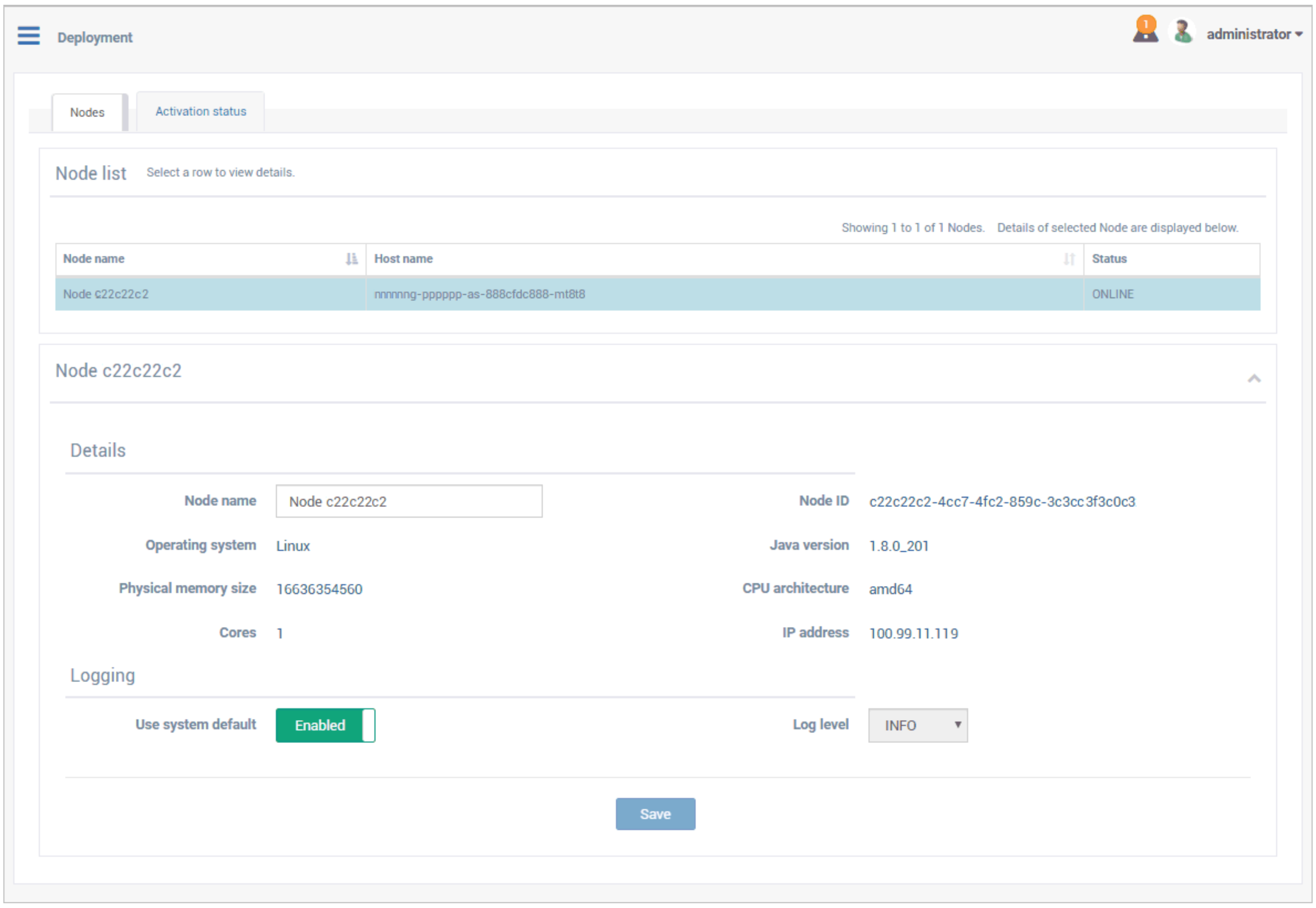
ActiveServer can be run in either a **single node** or **multi node** setup. The details for these nodes can be viewed on the **Deployment** page under the **Nodes** tab. This is also where you can set the logging level for a particular node, if it requires a different setting to the system wide log level setting.

Info

A guide on multi node setups will be provided in a future document version.

User Access

A user requires the **System admin** role to view and edit node settings.



Node list Select a row to view details.

Showing 1 to 1 of 1 Nodes. Details of selected Node are displayed below.

Node name	Host name	Status
Node c22c22c2	nnnnng-pppppp-as-888cfdc888-mt8t8	ONLINE

Node c22c22c2

Details

Node name	Node c22c22c2	Node ID	c22c22c2-4cc7-4fc2-859c-3c3cc3f3c0c3
Operating system	Linux	Java version	1.8.0_201
Physical memory size	16636354560	CPU architecture	amd64
Cores	1	IP address	100.99.11.119

Logging

Use system default	Enabled	Log level	INFO

Node details

Select a node from the **Node list** to view its details:

- **Node name** - editable name that can be given to the node to keep track of status.
- **Node ID** - system generated UUID for the node in the database.
- **Operating system** - operating system being used for the node's server.
- **Java version** - Java version being used for the node's server.
- **Physical memory size** - total amount of physical memory installed on the node's server.
- **CPU architecture** - type of CPU architecture chip installed on the node's server.
- **Cores** - amount of logical cores installed on the node's server.
- **IP address** - IP address that the node is hosted on.

Logging

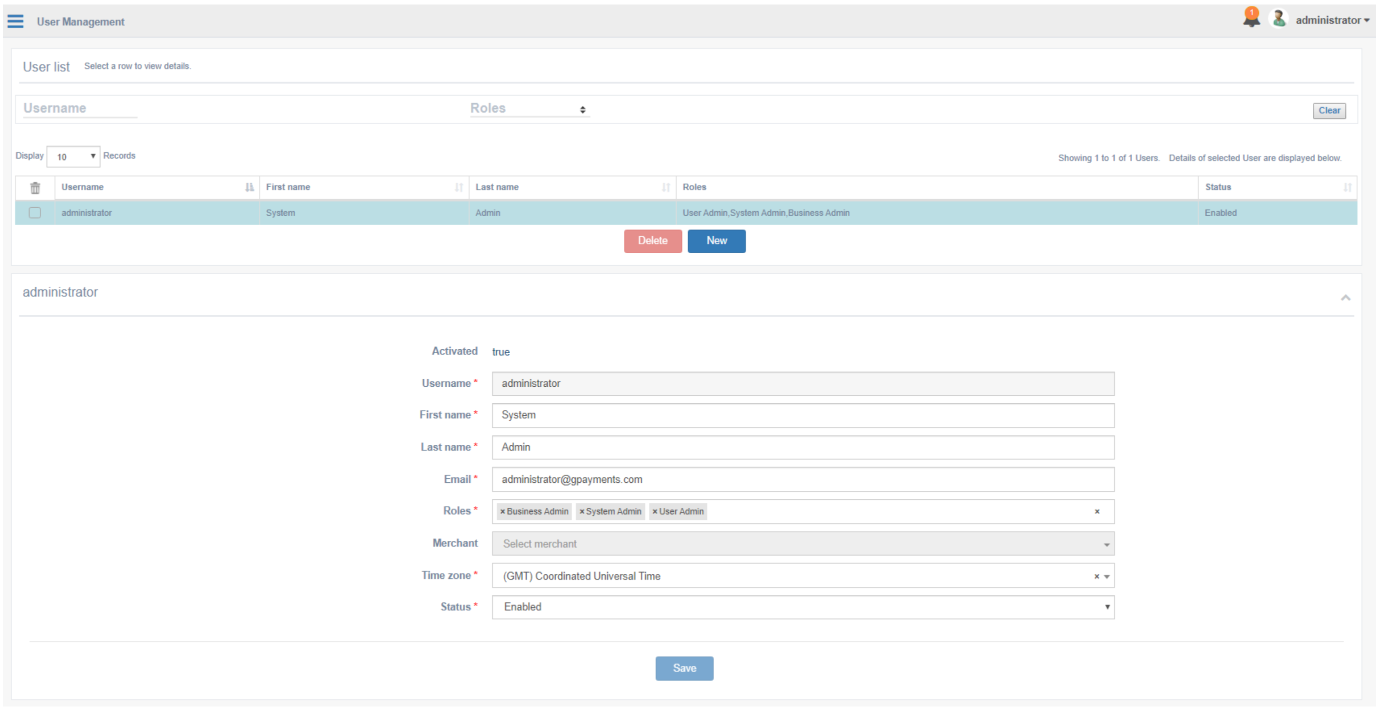
If a different level of logging is required for a particular node, the following settings can be changed:

- **Use system default** - If ***Enabled***, the node will use the system wide log level setting. If ***Disabled***, log level for the node can be changed manually.
- **Log level** Available to be edited if **Use system default** is ***Disabled***, allowing the log level to be changed.

Manage users

Users are created to provide access to the administration interface. User **roles** are assigned to users to provide them with the appropriate access for completing the tasks related to their particular business responsibilities. Refer to the roles and permissions guide for further information on **roles**.

The create, view, edit and delete processes for users are detailed below.



Showing 1 to 1 of 1 Users. Details of selected User are displayed below.

Username	First name	Last name	Roles	Status
administrator	System	Admin	User Admin,System Admin,Business Admin	Enabled

administrator

Activated	true
Username *	administrator
First name *	System
Last name *	Admin
Email *	administrator@gpayments.com
Roles *	× Business Admin × System Admin × User Admin
Merchant	Select merchant
Time zone *	(GMT) Coordinated Universal Time
Status *	Enabled

Create a user

To create a user, first head to the **User Management** page on the administration interface and select the ***New*** button.

Fill in the fields on the **New user** screen using the fields described below:

Important

The **Role/s** and **Merchant** fields below are critical for assigning correct access to the system. We recommend that you review the roles and permissions guide before creating new users.

- **Username** - unique value assigned to the user that is used to login to the administration interface. *Required*
- **First name** - first name of the user. *Required*
- **Last name** - last name of the user. *Required*
- **Email** - valid email address of the user. This address will be used to send email notifications such as password resets or system notifications. *Required*
- **Roles** - multi select box for roles, which can be assigned to the user. *Required*
- **Merchant** - if the user roles assigned give the user **scope** over a single merchant, the user can be assigned an already created merchant to manage. If the user roles assigned give the user **scope** over all merchants or no merchants, this field does not need to be populated and will be unavailable to be select. *Required*
- **Time zone** - default time zone for the display of times and dates on administration interface. *Required*
- **Status** - system status of the user:
  - **Enabled** - user can access administration interface functionality.
  - **Disabled** - user disabled from accessing administration interface functionality.

User Access

A user requires the **User admin** role to create users.

View user details

A user's details can be accessed from the **User Management** page on the administration interface by selecting it from the list of users.

The number of users displayed can be limited by filtering using the following fields:

- **Username** - all or part of the username.

- **Role(s)** - drop down select for a single system role.

The result table will show **Username**, **First name**, **Last name**, **Roles** and **Status** details outlined above.

User Access

A user requires the **User admin** role to view user details.

Edit user details

To edit a user, view its profile and edit available fields.

Important

There must always be one user in the system with the **User Admin** role. If editing a users details causes this check to fail, an error will occur.

User Access

A user requires the **User admin** role to edit user details.

Delete a user

To delete a user, first head to the **User Management** page on the administration interface. Filter the list to find the user and select the **delete check box** adjacent to the Username, in the search result table. Select the ***Delete*** button and confirm on the dialogue box.

Important

There must always be at least one user in the system with the **User Admin** role. If deleting a user causes this check to fail, an error will occur.

User Access

A user requires the **User admin** role to delete users.

Audit logs

Audit logs provides access to a comprehensive log of administrator activity, for audit purposes. It includes logs for changes to settings, merchants, users, and deployment information.

User Access

A user requires the **System admin** role to view audit logs.

Search audit logs

The most recent audit log entries are shown in the **Audit log list**, by default. The list can be filtered using the following:

- **From** - only entries on or after this date will be shown.
- **To** - only entries before or on this date will be shown.
- **User** - full or partial search on the username of the user who performed the audited entry.
- **Revision type** - type of operation performed on the database entity:
    - **Addition** - new record was added to the database entity.
    - **Modification** - existing record was modified in the database entity.
    - **Deletion** - existing record was removed from the database entity.
- **Entity name** - table in the database that the audit affected.

Select the **Search** button to display the relevant **Audit logs**. Use the ***Clear*** button to reset the search fields.

Audit log list

The **Audit log list** shows a table of **Audit logs** with the amount of log entries returned by the search criteria, along with the following log information:

- **Entity name** - table in the database that the audit affected.

- **Revision type** - type of operation performed on the database entity:
    - **Addition** - new record was added to the database entity.
    - **Modification** - existing record was modified in the database entity.
    - **Deletion** - existing record was removed from the database entity.
- **Revision date** - date and time of the audit log, in dd/mm/yyyy format.
- **User** - username of the user who performed the audited entry.
- **IP** - IP address of the user.

Selecting a log entry from the **Audit log list** will show the **Audit log details**, including the modified ***attribute*** and ***old/new*** values.

Configure system settings

Settings allows you to configure system settings for your ActiveServer instance. **Settings** has 3 tabs:

- System
- Security
- ActiveMerchant Migration

System

The **System** tab has 2 sections:

Server URLs

- **External URL** - externally accessible URL, used for authentication callbacks and product activation. The URL can be populated with the form of `https://"your ActiveServer domain name":"server port number"`, e.g. `https://paymentgateway.com:8443`. The `server port number` in the URL pattern is the `as.server.http.port` or `as.server.https.port` value (depending on the protocol being used) set in the **ActiveServer** configuration file.

> ⚠ **Warning**
>
> Updating the **External URL** will also initiate an update of the 3DS Server URL in the Directory Server settings for each card scheme. If the **3DS Server URL** value is empty for any card scheme, it will be updated to use the new **External URL** value, with the HTTPS listening port value also being appended. If the **3DS Server URL** value is already set, no changes will be made.
>
> This is to assist with the setup process as these URLs are generally the same. If your architecture setup has a separate URL assigned for the **3DS Server URL**, this setting should be updated before performing a transaction.

Tip

For a load-balanced setup, this URL may be different to the URL pattern described above and there may be a separate admin UI interface URL that has been configured. Please make sure that the load-balancer for the **External URL** forwards the requests to the server ports mentioned above.

E.g. If the URL `https://paymentgateway.com` has been configured for server callbacks and `https://admin.paymentgateway.com` has been configured for admin UI interface requests, then `https://paymentgateway.com` should be used for the **External URL**.

- **API URL** - Optional URL used to receive authentication and administration API calls. The domain name of this URL will also be used to generate client certificates for the authentication of APIs (x.509). If it is not provided, by default **ActiveServer** will use the domain name in the External URL for client certificate generation. Note this URL does not have to be publicly accessible. The form of the URL is the same as the **External URL**, with the port number being the API port.
- **Admin URL** - Optional URL that is used for user email activation and reset password procedures. This value can only be entered if the Admin port in the **ActiveServer** configuration file is enabled. If the admin port is enabled and this value is empty, a localhost domain will be used with the admin port specified, however this will cause issues accessing the domain from a remote host. If the admin port is disabled, the domain name in the **External URL** will be used. Note this URL does not have to be publicly accessible.

Logging

- **Log level** - verbosity of the console output and system logs. Possible values in least verbose to most verbose order: **ERROR** > **INFO** > **DEBUG**.

Security

- **Session timeout (read only)** - interval a login session is valid for before expiring and requiring the user to enter their login credentials again. By default, the session timeout value is 900 sec (15 min) and is loaded from an internal setting. To change this setting, add the following line into the `application-prod.properties` file and restart the instance:

```
as.settings.session-timeout={time in seconds}
```

For example, to set the session timeout to 1800 seconds (30 minutes), add `as.settings.session-timeout=1800`.

> **Important**
>
> The value must be a positive integer in the range of 300 ~ 3600 seconds (5 ~ 60 minutes).

- **Session failed attempts** - number of failed login attempts permitted before login is temporarily disabled for the time specified by the session lock time. After the time has elapsed, the session can be re-established by providing the correct credentials (unit: attempts)
- **Session lock time** - interval a user will be locked out for if they exceed the failed login attempts amount (unit: minutes)
- **Password expiry period** - number of days a password is valid for before requiring a new password to be created (unit: days)
- **Password history check** - number of unique passwords required to be used before a specific password can be used again (unit: unique passwords)
- **Force two factor login** - **enable** or **disable** two factor authentication for login for **all** users on the server. ActiveServer uses Google Authenticator to provide two factor authentication for users. If this setting is **enabled**, any user who does not have two factor authentication already set up for their account will be forced to set it up on their next login before being able to use any system functionality. Steps to set up the Google Authenticator are provided on screen.

Rotate key

Shows the current encryption key's creation date and allows the user to rotate the key used by selecting **Rotate key**.

HSM

This feature allows the user to update the HSM PIN if it has been changed:

- **Full file name and path of PKCS#11 library** - this value is read from the `application-prod.properties` and can only be changed by updating the `application-prod.properties` file and restarting the server.

- **Slot number of HSM** - this value is read from the `application-prod.properties` and can only be changed by updating the `application-prod.properties` file and restarting the server.
- **HSM PIN** - allows the new HSM PIN to be entered.

Selecting the **Test HSM connection** button will attempt to connect to the HSM using the inputted **HSM PIN**. If the test is successful, the system will show "HSM connection successful", otherwise "Invalid HSM Pin" will be shown.

Selecting the **Update** button will update the database with the **HSM PIN** value. **Restarting the server is required after updating**.

Warning

The system will update the HSM PIN regardless of the test result. This is to allow the PIN to be updated in the **ActiveServer** database before the HSM PIN is changed if required. Make sure the right PIN is entered before updating the system, as having the wrong HSM PIN will cause transactions to fail.

Tip

The HSM PIN management will only be shown if a HSM is in use.

Version 1.0.4

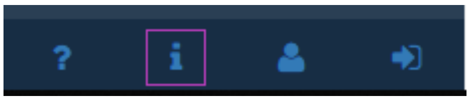
This feature was added in the version 1.0.4 release.

ActiveMerchant migration

The **ActiveMerchant Migration** tab allows a **Business Admin** user to import merchants and acquirers from GPayments **ActiveMerchant** (3DS1 MPI) to assist with the transition from 3DS1 to 3DS2.

For information on how to use the migration feature, refer to the ActiveMerchant migration guide.

View ActiveServer Information

The **About ActiveServer** page can be accessed from the icon at the bottom left corner of the page. 

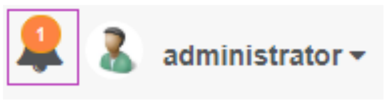
About ActiveServer displays basic information about the software deployment, which includes:

- **ActiveServer version** - software version of the currently installed ActiveServer.
- **OS name** - operating system in use.
- **OS version** - version of the operating system in use.
- **Database name** - database in use.
- **Database version** - version of the database in use.
- **Java edition and version** - Java edition and version installed.
- **Node count** - number of nodes deployed in this instance of ActiveServer.
- **Supported 3DS version** - 3D Secure version supported.
- **3DS Server reference number** - unique reference number for this instance of ActiveServer.
- **EULA** - selecting the **View** button will open the EULA in a PDF viewer, with additional options to download or print the PDF.

Notifications

System notifications are shown to users when important events need their attention. All users will be sent notifications relating to their own account based notifications, such as a password expiring soon. **System admins** are also shown notifications that relate to server events, such as a licensing or usage uploading issue.

Notifications are shown in the top right hand corner of the administration interface by selecting the 🔔 icon.



System notifications

System notifications are shown to **System admins** and can adversely affect system running procedures if not monitored. The following table indicates the possible system notifications and how to resolve them.

Notification	Scenario	Notification Message	Solution
No license	When software is first initialised, there will be no license in the system and this notification will be shown until the product is activated	**License warning**: This instance is not activated. Please add a **Product Activation Key (PAK)** on the **Deployment > Activation status** page. The PAK can be found on your GPayments **MyAccount activation** page.	User should follow the licensing guide to correctly license the product.
License issue: Warn	GPayments' licensing server has indicated that payment is outstanding on the account associated with the current instance, which will be disabled, without any further notice, in a specified period of time.	**License warning**: This instance will be disabled in y days because it has an overdue account. Please contact **GPayments support** for further information.	User should notify the appropriate accounts team to get in contact with GPayments support to resolve billing issue.

Notification	Scenario	Notification Message	Solution
License issue: Stop	GPayments' licensing server has indicated that payment is outstanding on the account associated with the current instance, which has been disabled.	**License warning**: This instance has been disabled because it has an overdue account. Please contact **GPayments support** for further information.	User should notify the appropriate accounts team to get in contact with GPayments support to resolve billing issue.
Upload issue: Warn	If the ActiveServer instance has failed to upload to the GPayments licensing server for a certain period of time, the system will start a warning process giving the user 60 days to rectify the error before the server is disabled.	**License warning**: This instance will be disabled in y days because it has not successfully reported authentication usage to GPayments' licensing server for a period of x days. Please contact **GPayments support** for more information.	User should investigate why usage uploading is failing, or contact GPayments support for assistance resolving the issue.
Upload issue: Stop	If the ActiveServer instance has failed to upload to the GPayments licensing server for a period of 60 days, the server will be disabled.	**License warning**: This instance has been disabled because it has not successfully reported authentication usage to the GPayment's licensing server for a period of 60 days. Please contact **GPayments support** for more information.	User should investigate why usage uploading is failing, or contact GPayments support for assistance resolving the issue.

User notifications

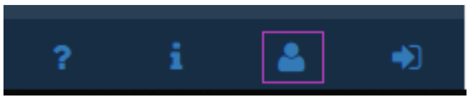
User notifications are shown to all users when an event will impact their account. The following table indicates the possible user notifications and how to resolve them.

Notification	Scenario	Notification Message	Solution
Password expiring	The user's password has only 7 days remaining before it expires.	The password for **user** will expire on **expiry date**.	User should update their password via the **User profile > Change password** screen.

User profile

The **User profile** allows users to edit details relating to their own account and change their passwords.

Select the **Profile** icon in the bottom left hand corner of the administration interface to access your **User profile**. There are two sections, **edit profile** and **change password**.



Note

All users are able to manage their own user profile.

Edit profile

Contains the account details for the user:

- **Username (required)** - name used to login to the administration interface.
- **First name (required)** - first name of the user.
- **Last name (required)** - last name of the user.
- **Email (required)** - fully qualified email address of the user. **This address will be used to send password reset and other system notifications.**
- **Time zone** - local timezone for all time and dates to be displayed in on the administration interface.
- **Two factor login status (required)** - toggle to **enable** or **disable** 2FA for the current user. Toggling from disabled to enabled will prompt the user to setup 2FA using Google Authenticator by following the steps displayed on screen.
- **Enabled** - prompts the user on login to enter the authenticator code associated with this account, otherwise login will fail.
- **Disabled** - 2FA is disabled and not required on login.

Important

If the instance has Force two factor login enabled, the user will not be able to disable 2FA.

Admin API client certificate

- **Download** - allows the download of the client certificate for the user, to be used in Admin API requests. For more information on this functionality, see the API document overview.
- **Revoke** - revokes the current client certificate if the security has been compromised and re-issues a certificate with new ID.

Warning

Revoking a client certificate will invalidate all instances of the certificate, and you will not be able to initiate API requests until the replacement certificate is downloaded.

Master Auth API client certificate

ActiveServer uses X.509 authentication to authorise users using the auth API, described here. If the user is a PSP that connects to **ActiveServer** on behalf of merchants, they may wish to avoid storing multiple client certificates for all merchants users. The **Master Auth API client certificate** can be used in combination with a Merchant token to authenticate a **Business Admin user** on behalf of a merchant. For information on using the master certificate with the merchant token, refer here.

Important

Only a **Business admin** can manage this certificate. As it can be used to authenticate on behalf of all merchants, special care should be taken in order to protect its security.

- **Download** - allows the download of the client certificate for the user, to be used for Authentication API authorisation for all merchants. For more information on this functionality, see the API document overview.
- **Revoke** - revokes the current client certificate if the security has been compromised and re-issues a certificate with new ID.

Warning

Revoking a client certificate will invalidate all instances of the certificate, and you will not be able to initiate API requests until the replacement certificate is downloaded.

CA certificates

- **Download** - allows the download of the servers CA certificates, to be used in Admin API requests. For more information on this functionality, see the API document overview.

Tip

Admin API client certificate and **CA certificate** management is only available once the instance has been activated.

Version 1.0.3

Certificate download was added in the version 1.0.3 release.

Change password

Allows the user to change their password by filling in the following fields:

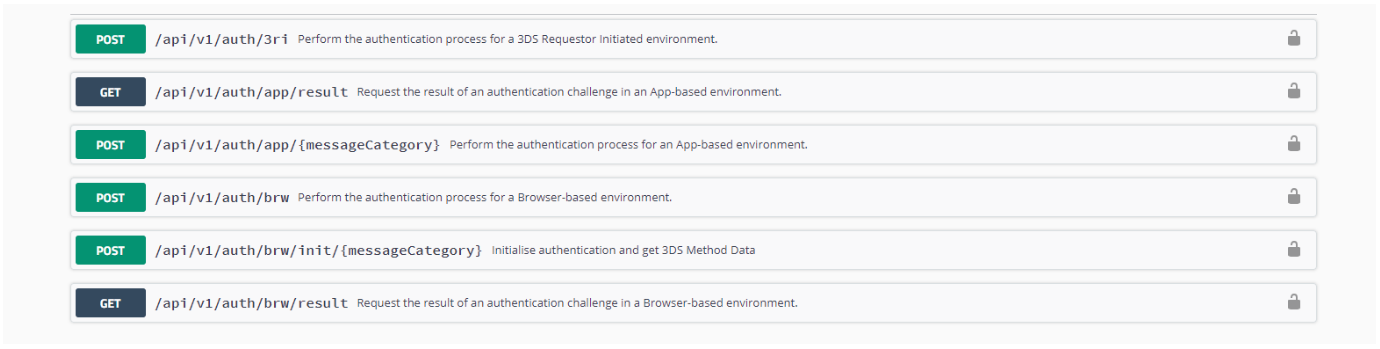
- **Current password (required)** - current password for the user, must be correct or password change will fail.
- **New password (required)** - new password for the user, must conform to the Password history check rules.
    - ***Requirements*** - *Between 8 and 100 characters, must contain at least one letter and one number*.
- **Confirm new password** - new password for the user, must match **New password** field.

API document overview

Getting started

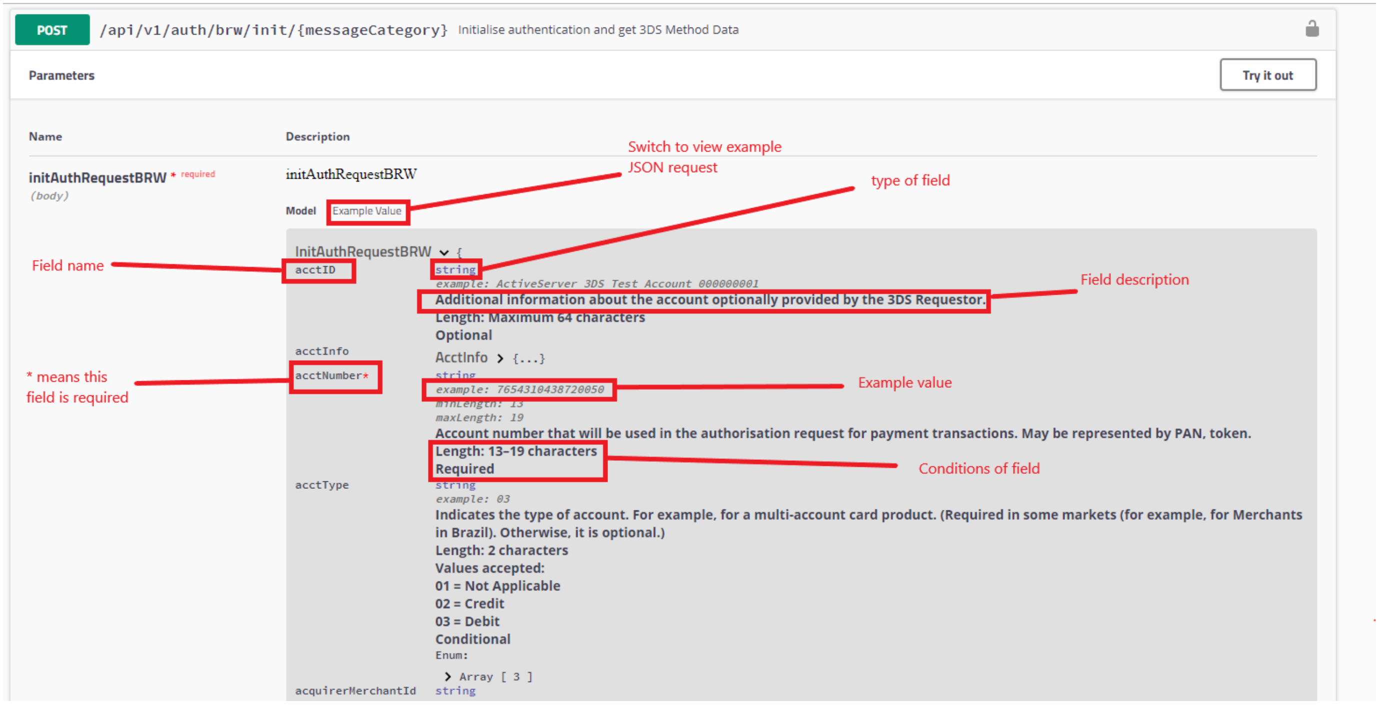
How to read the API Document?

GPayment's ActiveServer API documentation includes the allowed endpoints and the request HTTP method type for each API. For example, `/api/v1/auth/3ri` is the endpoint url and the allowed HTTP method is `POST`.



```
POST /api/v1/auth/3ri  Perform the authentication process for a 3DS Requestor Initiated environment.
GET  /api/v1/auth/app/result  Request the result of an authentication challenge in an App-based environment.
POST /api/v1/auth/app/{messageCategory}  Perform the authentication process for an App-based environment.
POST /api/v1/auth/brw  Perform the authentication process for a Browser-based environment.
POST /api/v1/auth/brw/init/{messageCategory}  Initialise authentication and get 3DS Method Data
GET  /api/v1/auth/brw/result  Request the result of an authentication challenge in a Browser-based environment.
```

If you want to check the details of the endpoints, click on one of the endpoint box's to open up its details. You will see something similar to the screenshot below. The screenshot explains how to read the API documentation.



```
POST /api/v1/auth/brw/init/{messageCategory}  Initialise authentication and get 3DS Method Data

Parameters                                                                Try it out

Name                          Description
initAuthRequestBRW * required initAuthRequestBRW
(body)
                              Model  Example Value

                              InitAuthRequestBRW ⌄ {
                              acctID              string
                                                  example: ActiveServer 3DS Test Account 000000001
                                                  Additional information about the account optionally provided by the 3DS Requestor.
                                                  Length: Maximum 64 characters
                                                  Optional
                              acctInfo            AcctInfo > {...}
                              acctNumber*         string
                                                  example: 7654310438720050
                                                  minLength: 13
                                                  maxLength: 19
                                                  Account number that will be used in the authorisation request for payment transactions. May be represented by PAN, token.
                                                  Length: 13–19 characters
                                                  Required
                              acctType            string
                                                  example: 03
                                                  Indicates the type of account. For example, for a multi-account card product. (Required in some markets (for example, for Merchants
                                                  in Brazil). Otherwise, it is optional.)
                                                  Length: 2 characters
                                                  Values accepted:
                                                  01 = Not Applicable
                                                  02 = Credit
                                                  03 = Debit
                                                  Conditional
                                                  Enum:
                                                   > Array [ 3 ]
                              acquirerMerchantId  string
```

Authentication API

The Authentication API allows merchants and payment gateways to integrate the 3D Secure 2 flow into their eCommerce site. All Authentication APIs start from `/api/v1/auth/...` and follow RESTFul naming conventions. There are 3 main authentication flows available:

- **App-based** - Authentication during a transaction on a Consumer Device that originates from an App provided by a registered agent (merchant, digital wallet, etc). For example, an eCommerce transaction originating during a checkout process within a merchant's App on the Consumer Device.
- **Browser-based** - Authentication during a transaction on a Consumer Device or Computer that originates from a website utilising a browser. For example, an eCommerce transaction originating during a checkout process within a website.
- **3DS Requestor Initiated (3RI)** - Confirmation of account information with no direct cardholder present. For example, a subscription-based eCommerce merchant confirming that an account is still valid.

Field Conditions

The following standards are used to identify the conditions of each field.

- **Required** - Sender shall include the data element in the identified message; ActiveServer checks for data element presence and validates the data element contents.
- **Conditional** - Sender shall include the data element in the identified message if the conditional inclusion requirements are met; ActiveServer checks for data element presence and validates data element contents. When no data is to be sent for a conditional data element, the data element should be **absent**.
- **Optional** - Sender may include the data element in the identified message; ActiveServer validates the data element contents when present. When no data is to be sent for an optional data element, the data element should be **absent**.

The latest Authentication APIs are available from here.

Auth API Authentication

All authentication API endpoints are secured by X.509 authentication which requires the client and server to be mutually authenticated. This is performed by using a client certificate. **ActiveServer** provides two types of client certificate: Merchant client certificate for each merchant, or Master Auth API client certificate for **Business admin** users. The steps to make an Auth API call are:

1. Make sure the instance is activated by following this guide.
2. Download the client certificate:
    - For Merchant client certificate, download from the merchant details page on the administration UI interface.
    - For Master Auth API client certificate, download from the user profile page on the administration UI interface.
3. Download the CA certificate chain from the user profile page. The CA certificate is in `.pem` format with the first certificate being the server CA, followed by GPayments Intermediate CA and finally the GPayments Root CA.

    Note

    If you are using GPayments 3DS Requestor Demo code, the CA certificate chain is already included.

4. You can follow the Integration guide to setup a 3DS requestor from the beginning to call the authentication API. For specific details on using the client certificate method chosen from step 2, see the Demo 3DS Requestor Configuration.

Warning

You will not see the `DOWNLOAD` button if your instance is not activated.

Auth API reference

Channel	Process	End point (API v1)	End point (API v2)
BRW	InitAuth	/api/v1/auth/brw/init/{messageCategory}	/api/v2/auth/brw/init
	Auth	/api/v1/auth/brw	`authUrl` from initAuth response
	Result	/api/v1/auth/brw/result	/api/v2/auth/brw/result
	ChallengeStatus	/api/v1/auth/challenge/status	/api/v2/auth/challenge/status

Channel	Process	End point (API v1)	End point (API v2)
APP	Auth	/api/v1/auth/app/{messageCategory}	/api/v2/auth/app
	Result	/api/v1/auth/app/result	/api/v2/auth/app/result
3RI		/api/v1/auth/3ri/{messageCategory}	/api/v2/auth/3ri

You can check the details of the **API message** for each endpoint in the API document.

Administration API

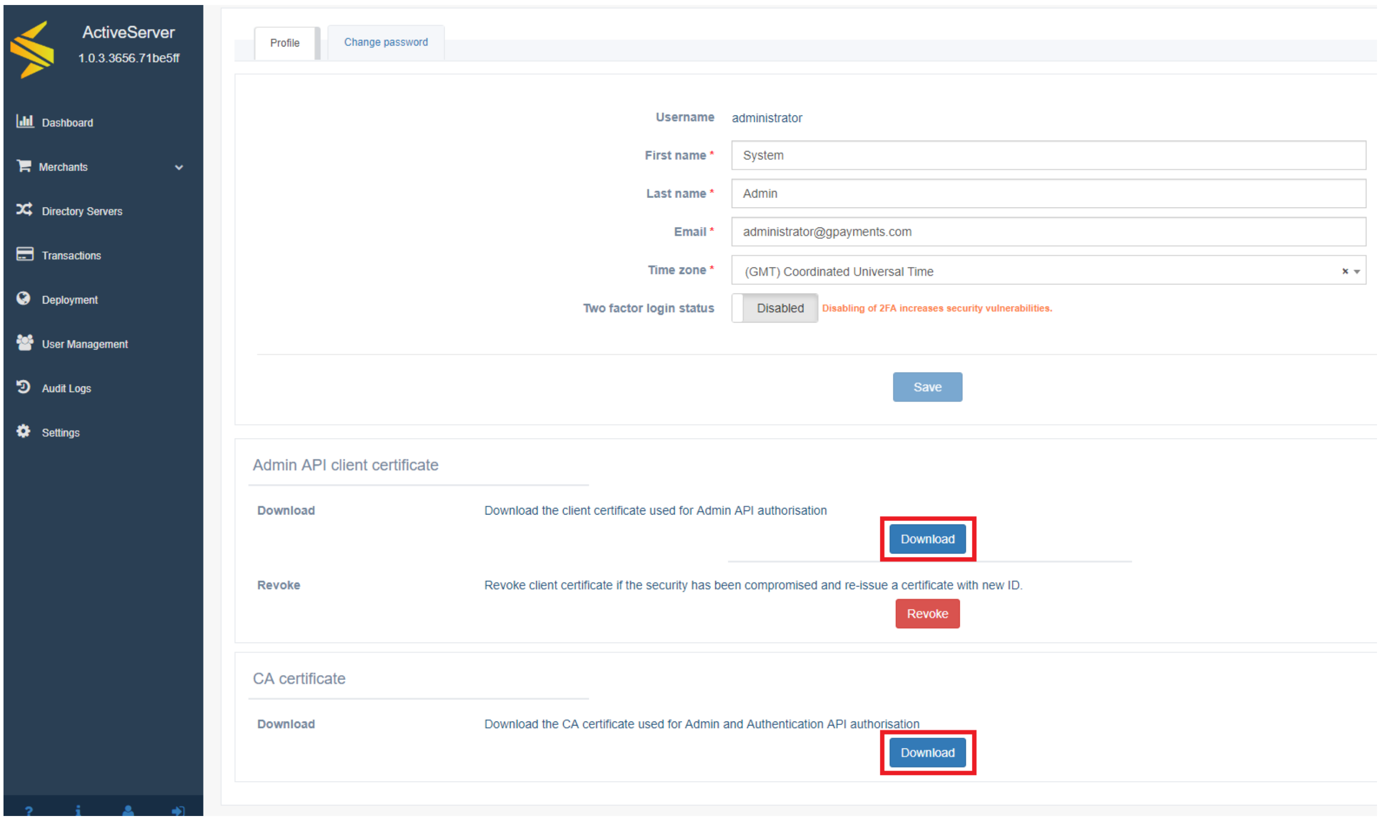
The Administration API is available to perform select administration tasks such as managing merchants. All Admin APIs start from `/api/v1/admin/...` and follow RESTFul naming conventions. For example, `creating merchant` which is described here, can be performed by the POST API endpoint **`/api/v1/admin/merchants`**.

The latest Administration APIs are available from here.

Admin API Authentication

Admin API Authentication is performed the same as the Authentication API, using a client certificate. The steps to make an Admin API call are:

1. Make sure the instance is activated by following this guide.
2. Download the client certificate for the Administration API from the user profile page in the administration interface. This certificate is in `.p12` format.
3. Download the CA certificate chain certificate from the same page. The CA certificate is in `.pem` format with the first certificate being the server CA, followed by GPayments Intermediate CA and finally the GPayments Root CA.
4. Use the downloaded client certificate and CA bundle for API authentication by including it inside the HTTP client request. Refer to the HTTP client you are using on how to do this or you can follow the API quick start for testing.



Username administrator
First name * System
Last name * Admin
Email * administrator@gpayments.com
Time zone * (GMT) Coordinated Universal Time
Two factor login status Disabled Disabling of 2FA increases security vulnerabilities.

Save

Admin API client certificate

Download Download the client certificate used for Admin API authorisation

Download

Revoke Revoke client certificate if the security has been compromised and re-issue a certificate with new ID.

Revoke

CA certificate

Download Download the CA certificate used for Admin and Authentication API authorisation

Download

Admin API Quick Start

To try out the API using cURL, you need to convert the X.509 certificate to PEM format by using **openssl** and using the following `curl` command.

1. Download and install openssl command line from https://www.openssl.org/.
2. Download the client certificate and CA certificate following the Admin API Authorisation section above.
3. Convert P12 to PEM format using OpenSSL
4. Perform the cURL request using your AS Admin API entry point. In this case we are getting a list of merchants from `/api/v1/admin/merchants` :

```
curl -k https://your.server.address:7443/api/v1/admin/merchants --cacert
cacerts.pem --cert crt.pem -v --key key_no_pass.pem
```

5. You should see the following cURL response:

```
< HTTP/1.1 200 OK
< Expires: 0
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< X-XSS-Protection: 1; mode=block
< Pragma: no-cache
< X-Frame-Options: DENY
< Date: Mon, 27 May 2019 09:51:59 GMT
< X-Total-Count: 1
< Connection: keep-alive
< X-Content-Type-Options: nosniff
< Strict-Transport-Security: max-age=31536000 ; includeSubDomains
< Transfer-Encoding: chunked
< Content-Type: application/json;charset=UTF-8
< Link: </api/v1/admin/merchants?page=0&size=20>; rel="last",</api/v1/admin/
merchants?page=0&size=20>; rel="first"
<
* Connection #0 to host your.server.adress left intact
[{"bins":"40001 (Test
Acquirer)","merchantId":"123456789012345","name":"Test
Merchant","status":"ENABLED","merId":"48f3b030-ed1d-4f7f-
aa70-85cda288e0cb"}]%
```

Converting P12 to PEM using OpenSSL

Some HTTP client such as `curl` requires client certificate to be in `.pem` format. Following commands allow extracting the client certificate and its associated private key from `.p12` file into separate PEM formatted files.

1. Open up a terminal.
2. Extract the private key from the `.p12` file using the following command, entering the password when prompted:

```
openssl pkcs12 -in YOUR_P12_FILE_NAME.p12 -nocerts -out key.pem
```

3. Extract the certificate from the `.p12` file using the following command, entering the password when prompted:

```
openssl pkcs12 -in YOUR_P12_FILE_NAME.p12 -clcerts -nokeys -out crt.pem
```

4. Remove the passphrase from the private key

```
openssl rsa -in key.pem -out key_no_pass.pem
```

Error codes

This section provides details of errors that could occur during the running of ActiveServer.

Authentication API Error Codes Overview

- Only error codes in category **3DS Error Codes**, **Transaction Error Codes** and **General Error Codes** can be returned from the Authentication API(`/api/v2/auth/**`).
- **Security Error Codes**, **User Error Codes** or **Setup Error Code** will not be returned during the Authentication API.
- For each error codes, associated HTTP status code from the table below will be returned.
- Error code description contains the possible scenario in which the error code may be returned, and for some error codes common solutions are also highlighted.

Description of each error code categories are as below:

- **3DS Error Codes** - Error codes in this category is defined by the EMVCO Core Protocol specifications. Error codes defined here can be returned from either **3DS Server (ActiveServer)**, **DS**, **ACS** or the **3DS SDK**. The component in which identified the error will return the error response and set the `errorComponent` field as itself in the JSON response (e.g. if error was identified by DS it will set the `errorComponent` to `D`). If the error was identified in components that is outside of ActiveServer, ActiveServer will return the same error JSON back to the **3DS Requestor**. `errorMessageType` , `errorDetail` and `errorDescription` field can be used to interpret the message that was erroneous. Please refer to the `ApiErrorResponse` model for description of each fields.
- **Transaction Error Codes** - Transaction error code defined by ActiveServer. `errorComponent` field will be `S` at all times because the error was identified by 3DS Server.
- **General Code** - Errors that does not fall into either **3DS Error Codes** or **Transaction Error Codes** are returned as **General Code**. The error code may be returned also from the Admin API. Check the **Auth API Description** for descriptions related to the Authentication API.

Tip

Please note that error codes which has the tag **Not returned in Auth API V2** will not be returned as a response for `/api/v2/auth/**`.

3DS Error Codes (101 ~ 405)

Code	Name	HTTP Status Code	Description
101	MESSAGE_RECEIVED_INVALID	400	Received message is invalid. Message is not AReq, ARes, CReq, CRes, PReq, PRes, RReq, or RRes. For example, 3DS Server receives an message from DS as a response to AReq that is not ARes or Erro message.
102	MESSAGE_VERSION_NUMBER_ NOT_SUPPORTED	400	Unsupported message version number. Message Version Number received is not valid for the receiving component. For example, DS sends a `messageVersion` field set to an invalid value, or value that is not supported by the ACS.
103	SENT_MESSAGES_LIMIT_EXCEEDED	500	Message sent exceeds the limit. Exceeded maximum number of PReq messages sent to the DS. **Not returned in Auth API V2**. (PReq is outside the authentication flow and is an internal process between ActiveServer and DS).

Code	Name	HTTP Status Code	Description
201	REQUIRED_DATA_ELEMENT_MISSING	400	A message element required as defined according the specification is missing. This error code will be returned if any of the fields marked as **required** is missing in the request. e.g. `messageCategory` is missing in the call to `/api/v2/auth/brw`. If the `errorMessageType` is `AReq` or missing, and `errorComponent` is `S` then the request from the **3DS requestor** is missing the required fields defined by the Authentication API. Please double check the fields returned in `errorDetail` is present in the request.
202	CRITICAL_MESSAGE_ EXTENSION_NOT_RECOGNISED	400	Message extension that is critical is not present. May be returned from DS or ACS if the `messageExtension` field is missing an identifier.
203	FORMAT_OF_ONE_OR_MORE _DATA_ELEMENTS_IS_INVALID _ACCORDING_TO_THE_SPECIFICATION	400	Data element is not in the required format or value is invalid as defined according the specification. This error code will be returned if any of the fields in the request is not well formatted. e.g. `purchaseAmount` is not numeric in the call to `/api/v2/auth/brw`. If the `errorMessageType` is `AReq` or empty, and `errorComponent` is `S` then request from the **3DS requestor** is not matching the fields defined by the Authentication API. Please double check the formatting of fields returned in `errorDetail` is present in the request.
204	DUPLICATE_DATA_ELEMENT	400	Found duplicate data elements in the request.
301	TRANSACTION_ID_NOT_RECOGNISED	400	Transaction ID received is not valid for the receiving component. For example, 3DS requestor sets the `threeDSSereverTransID` in the `/api/v2/auth/brw` that is different from the one returned by `/api/v2/auth/brw/init`.

Code	Name	HTTP Status Code	Description
302	DATA_DECRYPTION_FAILURE	500	Data could not be decrypted by the receiving system due to technical or other reason. DS may return this error code if data decryption of the SDK Encrypted data failed.
303	ACCESS_DENIED_INVALID_ENDPOINT	401	Endpoint for the API request is invalid. Check the requesting url. Reference number does not represent the participating component. e.g. `acsReferenceNumber` sent from ACS to DS is invalid.
304	ISO_CODE_INVALID	400	ISO code not valid per ISO tables (for either country or currency).
305	TRANSACTION_DATA_NOT_VALID	400	Transaction data is invalid. Please refer to the error description to find out why the transaction data was invalid.
306	MERCHANT_CATEGORY_CODE_MCC_NOT_VALID_FOR_PAYMENT_SYSTEM	400	Merchant category code is invalid. Invalid MCC received in the AReq message and DS may throw this error back to ActiveServer.
307	SERIAL_NUMBER_NOT_VALID	500	Serial number is invalid. **Not returned in Auth API V2**. (PReq is outside the authentication flow and is an internal process between ActiveServer and DS).
402	TRANSACTION_TIMED_OUT	408	Transaction has timed out. In ActiveServer, this error code is returned if transaction timed out during sending the request to the DS e.g. send AReq to the DS.
403	TRANSIENT_SYSTEM_FAILURE	500	System has failed for a short period. For example, a slowly processing back-end system.

Code	Name	HTTP Status Code	Description
404	PERMANENT_SYSTEM_FAILURE	500	System has failed permanently. For example, a critical database cannot be accessed. May be returned if DS settings is not properly configured in ActiveServer such as client certificate for the DS is not installed in ActiveServer.
405	SYSTEM_CONNECTION_FAILURE	500	Failed to connect to the system. For example, the sending component is unable to establish connection to the receiving component.

Transaction Error Codes (1000 ~ 1027)

Code	Name	HTTP Status Code	Description
1001	DIRECTORY_SERVER_NOT_FOUND	500	No directory server was found for a card schem associated with the PAN. May be returned if **Default URL** is empty in the administration UI fc a card scheme. Make sure the **Default URL** is configured in the ActiveServer admin UI dashboard.
1002	ERROR_SAVE_TRANSACTION	500	Error occurred while saving transaction. May be returned if transaction details is failed to be sav into the database during the authentication.
1003	ERROR_SAVE_TRANSACTION_MESSAGE	500	Error returned while saving transaction messag **Not returned in Auth API V2**. If error occurred while saving raw message e.g. saving raw ARec JSON, it will not fail the transaction.
1004	UNHANDLED_EXCEPTION	500	Unhandled exception occurred during the transaction. Please check the error description report error logs for further investigation.

Code	Name	HTTP Status Code	Description
1005	PAN_NOT_PARTICIPATING	400	Primary Account Number (PAN) is not participating. **Not returned in Auth API V2**.
1009	MERCHANT_INTERFACE_DISABLED	400	The interface is disabled for this merchant. **Not returned in Auth API V2**. `MERCHANT_ID_THREEDS_REQUESTOR_ID_INVALID` `(1026)` will be returned instead.
1011	INVALID_LICENSE	403	ActiveServer is running on invalid license. Pleas resolve licensing issue with GPayments.
1013	INVALID_TRANSACTION_ID	400	Transaction ID of 3DS server is not recognised. This error code may be returned if `threeDSServerTransID` is invalid in the given request.
1014	INVALID_REQUESTOR_TRANSACTION_ID	400	Transaction ID of 3DS requestor is not recognised. May be returned if the `threeDSRequestorTransID` is not in UUID form
1015	THREEDS_REQUESTOR_NOT_FOUND	400	Invalid 3DS Requestor ID/Merchant ID. **Not returned in Auth API V2**. `MERCHANT_ID_THREEDS_REQUESTOR_ID_INVALID` `(1026)` will be returned when client certificate c merchantId is invalid.
1016	MISSING_REQUIRED_ELEMENT	400	Required element missing. May be returned if required fields in the authentication API is missing.
1018	ELEMENT_NOT_DEFINED	400	Message element not a defined message. **Not returned in Auth API V2**.
1019	PROTOCOL_OLD	500	Protocol version is too old. **Not returned in Autl API V2**.

Code	Name	HTTP Status Code	Description
1020	ERROR_TRANSMISSION_DATA	500	Errors in data transmission. It will be returned when there is error with sending request and receiving response from the DS. If the reason fo the error is request being timed out, then it will return error code `TRANSACTION_TIMED_OUT (402` instead. If the connection is not established in t first place, then it will return `DIRECTORY_SERVER_NOT_AVAILABLE (1001)`.
1021	PRIOR_TRANS_ID_NOT_FOUND	400	Error in setting requestor prior Transaction ID. Prior Transaction ID couldn't be found. It will be returned if `priorTransID` given in the authentication request is not valid.
1022	INVALID_FORMAT	400	Format of one or more elements is invalid according to the specification. May be returned fields in the authentication API has an invalid format. e.g. `browserInfo` given in `/api/v2/auth/brw` is not in the same format a the one collected by ActiveServer.
1023	CARD_RANGE_IS_NOT_VALID	400	Card range provided is invalid. **Not returned in Auth API V2**.
1024	CACHE_UPDATE_IS_DISABLE	500	Cache update is disabled. **Not returned in Auth API V2**.
1025	CACHE_REFRESH_INTERVAL_IS_NOT_SET	500	Cache refresh interval is not set. **Not returned i Auth API V2**.
1026	MERCHANT_ID_THREEDS_REQUESTOR_ ID_INVALID	400	Invalid `merchantId` is given to the authenticatio request. Make sure that `merchantId` provided i the request matches the client certificate for th merchant or the `merchantToken` if master auth certificate is used. If you have revoked the clien certificate, make sure to update the client certificate or the `merchantToken` in the API request.

Code	Name	HTTP Status Code	Description
1027	UNSUPPORTED_API_VERSION	403	This error may be thrown if the API version you are trying to make a request to is not supportec For example, API version 1 is not supported for ActiveServer using AWS KMS.

General Error Codes (2000 ~ 2010)

Code	Name	HTTP Status Code	Description	Auth API Description
2000	NOT_FOUND	404	Resource not found.	**Not returned in Auth API V2**.
2001	DUPLICATE_RECORD	409	Record already exists.	**Not returned in Auth API V2**.
2002	VALIDATION_ERROR	400	Invalid inputs.	May be returned if the request is not properly formatted as a JSON.
2003	INVALID_REQUEST	400	Invalid request.	**Not returned in Auth API V2**.
2004	CONCURRENCY_FAILURE	409	Failed to update node.	**Not returned in Auth API V2**.
2005	ACCESS_DENIED	401	Access is denied.	Check the error detail for more description for why the access was denied.
2006	METHOD_NOT_SUPPORTED	405	Request HTTP method is not supported.	**Not returned in Auth API V2**

Code	Name	HTTP Status Code	Description	Auth API Description
2007	INTERNAL_SERVER_ERROR	500	Internal server error.	Internal server error has occurred in ActiveServer, may be due to some configuration issue or setup issue. Please refer to the error description for more details.
2008	DATA_INTEGRITY_VIOLATION_ERROR	400	A specified value violated the integrity constraints. May occur if attempting to insert of update results in violation of an integrity constraint. e.g. unique primary keys are not inserted into the table.	**Not returned in Auth API V2**
2009	SESSION_TIMED_OUT	408	Session has timed out.	May be returned if the transaction has already finished.

Security Error Codes (3001 ~ 3024)

Code	Name	HTTP Status Code	Description
3001	JDK_NOT_SUPPORT_SHA224WITHRSA	500	JDK used does not support the SHA224 with RSA algorithm.
3002	NO_SUCH_ALGORITHM	500	No such algorithm.

Code	Name	HTTP Status Code	Description
3003	INVALID_CERT	400	The certificate's public key is not compatible with the corresponding private key.
3004	INVALID_CHAIN	400	ActiveServer is unable to build the full certificate chain as one or more intermediate certificates cannot be found in the CA certificate store. You should either install/ import a certificate which contains the full chain or install the missing intermediate certificates before attempting again.
3005	NO_PRIVATE_KEY_FOUND	400	No private key found.
3006	INVALID_CERTIFICATE_CONTENT	400	Invalid certificate content
3007	CERTIFICATE_IO_READ	400	Unable to read certificate.
3008	SUCH_PROVIDER_EXCEPTION	500	No such provider exception.
3009	NO_KEY	400	The certificate could not be installed because this object does not have an existing key.
3010	CERTIFICATE_CHAIN_BAD_FORMAT	400	Certificate chain has invalid format.
3011	MISMATCHED_PASSWORDS	400	Password fields do not match.
3012	IMPORT_CERTIFICATE	400	No certificate found for the importing certificate. Please import client certificate.
3013	IMPORT_NO_CERTIFICATE	400	There is no certificate to export.
3014	FAILED_TO_INITIALIZE	500	Failed to initialise.
3015	ENCRYPTION_FAIL	500	Failed to encrypt.
3016	DECRYPTION_FAIL	500	Failed to decrypt.

Code	Name	HTTP Status Code	Description
3017	INVALID_HSM_PROVIDER	500	The specified provider name for hardware encryption is not supported
3018	INVALID_PKCS11_CONFIG	500	Invalid PKCS11 config path
3019	FAILED_TO_INITIALIZE_PKCS11	500	Failed to initialise PKCS11.
3020	IMPORT_FAIL	500	Failed to import.
3021	NOT_SUPPORTED_IBM_PROVIDER	500	Only SUN provider is supported.
3022	UNABLE_TO_LOAD_KEYSTORE	500	Loading keystore failed.
3023	UNABLE_TO_LOAD_CERTIFICATE	500	Loading certificate failed.
3024	INVALID_KEY_SIZE	500	Key size is invalid.

User Error Codes (4000 ~ 4022)

Code	Name	HTTP Status Code	Description
4000	DUPLICATE_EMAIL	400	E-mail already in use.
4001	LAST_ADMIN_DELETE_NOT_ALLOWED	400	You need to be at least a System Admin user to perform this action.
4002	ACCOUNT_IS_LOCKED	401	Your account is locked.
4003	ACCOUNT_IS_DISABLED	401	Your account is disabled.
4004	ACCOUNT_WILL_BE_LOCKED	401	Your account will be locked after another wrong try. If you have been forgotten your password please click on "Lost your password"

Code	Name	HTTP Status Code	Description
4005	ACCOUNT_WAS_LOCKED	401	Password has been locked for 1 hour.
4006	ACCOUNT_IS_INACTIVE	401	Your account was not activated.
4007	PASSWORD_POLICY_MATCH	401	The password should be minimum eight characters, with at least one letter and one number.
4008	LOGIN_ALREADY_IN_USE	401	Username already in use.
4009	EMAIL_ALREADY_IN_USE	401	Email already in use.
4010	INVALID_TOTP_CODE	400	Invalid totp authentication code.
4011	EMAIL_SENDING_FAILED	400	Failed to send email.
4012	EMAIL_NOT_REGISTERED	400	Your email is not registered.
4014	FAILED_TO_CREATE_ACCOUNT	500	Failed to create the account.
4015	TWO_FA_MANDATORY	400	Using two factor login is mandatory.
4016	PASSWORD_EXPIRED	403	The password for user was expired.
4017	PASSWORD_EXPIRED_WARNING	403	The password for user is going to be expired on.
4018	PASSWORD_HISTORY_MATCHED	403	The password matched with the previous historical passwords.
4019	INVALID_TOKEN	400	An invalid token.
4020	INVALID_HSM_PIN	400	Invalid HSM Pin.
4021	INVALID_PASSWORD	400	Invalid password.
4022	EMAIL_INVALID_ACTIVATION	403	Account activation code is invalid.

Setup Error Code (5000)

Code	Name	HTTP Status Code	Description
5000	SETUP_NOT_ALLOWED	500	Setup is not allowed.

Glossary

This page provides a list of terms relating to 3D Secure 2, some are not used elsewhere in this documentation but are included for completeness of the subject area. Familiarise yourself with them now or refer back to this page when you come across an unfamiliar word.

Term	Acronym	Definition
3DS Client		The consumer-facing component, such as a browser-based or mobile app online shopping site, which facilitates consumer interaction with the 3DS Requestor for initiation of the EMV 3-D Secure protocol.
3DS Integrator		An EMV 3-D Secure participant that facilitates and integrates the 3DS Requestor Environment, and optionally facilitates integration between the Merchant and the Acquirer.
3DS Requestor		The initiator of the EMV 3-D Secure Authentication Request, known as the AReq message. For example, this may be a merchant or a digital wallet requesting authentication within a purchase flow.
3DS Requestor App		An App on a Consumer Device that can process a 3-D Secure transaction through the use of a 3DS SDK. The 3DS Requestor App is enabled through integration with the 3DS SDK.
3DS Requestor Environment		This describes the 3DS Requestor controlled components of the Merchant / Acquirer domain, which are typically facilitated by the 3DS Integrator. These components include the 3DS Requestor App, 3DS SDK, and 3DS Server. Implementation of the 3DS Requestor Environment will vary as defined by the 3DS Integrator.
3DS Software Development Kit	**3DS SDK**	3-D Secure Software Development Kit. A component that is incorporated into the 3DS Requestor App. The 3DS SDK performs functions related to 3-D Secure on behalf of the 3DS Server.
3DS Requestor Initiated	**3RI**	3-D Secure transaction initiated by the 3DS Requestor for the purpose of confirming an account is still valid. The main use case being recurrent transactions (TV subscriptions, utility bill payments, etc.) where the merchant wants perform a Non-Payment transaction to verify that a subscription user still has a valid form of payment.

Term	Acronym	Definition
3DS Server	**3DSS**	Refers to the 3DS Integrator's server or systems that handle online transactions and facilitate communication between the 3DS Requestor and the Directory Server.
3-D Secure	**3DS**	Three Domain Secure, an eCommerce authentication protocol that for version 2 onwards enables the secure processing of payment, non-payment and account confirmation card transactions.
Access Control Server	**ACS**	A component that operates in the Issuer Domain, which verifies whether authentication is available for a card number and device type, and authenticates specific Cardholders.
Attempts		Used in the EMV 3DS specification to indicate the process by which proof of an authentication attempt is generated when payment authentication is not available. Support for Attempts is determined by each DS.
Authentication		In the context of 3-D Secure, the process of confirming that the person making an eCommerce transaction is entitled to use the payment card.
Authentication Request Message	**AReq**	An EMV 3-D Secure message sent by the 3DS Server, via the DS, to the ACS to initiate the authentication process.
Authentication Response Message	**ARes**	An EMV 3-D Secure message returned by the ACS, via the DS, in response to an Authentication Request message.
Authentication Value	**AV**	A cryptographic value generated by the ACS to provide a way, during authorisation processing, for the authorisation system to validate the integrity of the authentication result. The AV algorithm is defined by each Payment System.
Authorisation		A process by which an Issuer, or a processor on the Issuer's behalf, approves a transaction for payment.
Authorisation System		The systems and services through which a Payment System delivers online financial processing, authorisation, clearing, and settlement services to Issuers and Acquirers.
Bank Identification Number	**BIN**	The first six digits of a payment card account number that uniquely identifies the issuing financial institution. Also referred to as an Issuer Identification Number (IIN) in ISO 7812.

Term	Acronym	Definition
Base64		Encoding applied to the Authentication Value data element as defined in RFC 2045.
Base64 URL		Encoding applied to the 3DS Method Data, Device Information and the CReq/CRes messages as defined in RFC 7515.
Card		Card is synonymous with the account of a payment card, in the *EMV 3-D Secure Protocol and Core Functions Specification*.
Certificate Authority	**CA**	An entity that issues digital certificates.
Cardholder		An individual to whom a card is issued or who is authorised to use that card.
Challenge		The process where the ACS is in communication with the 3DS Client to obtain additional information through Cardholder interaction.
Challenge Flow		A 3-D Secure flow that involves Cardholder interaction as defined in the *EMV 3-D Secure Protocol and Core Functions Specification*.
Challenge Request Message	**CReq**	An EMV 3-D Secure message sent by the 3DS SDK or 3DS Server where additional information is sent from the Cardholder to the ACS to support the authentication process.
Challenge Response Message	**CRes**	The ACS response to the CReq message. It can indicate the result of the Cardholder authentication or, in the case of an App-based model, also signal that further Cardholder interaction is required to complete the authentication.
Consumer Device		Device used by a Cardholder such as a smart phone, laptop, or tablet that the Cardholder uses to conduct payment activities including authentication and purchase.
Device Channel		Indicates the channel from which the transaction originated. Either: • App-based (01-APP) • Browser-based (02-BRW) • 3DS Requestor Initiated (03-3RI)
Device Information		Data provided by the Consumer Device that is used in the authentication process.

Term	Acronym	Definition
Directory Server	**DS**	A server component operated in the Interoperability Domain; it performs a number of functions that include: authenticating the 3DS Server, routing messages between the 3DS Server and the ACS, and validating the 3DS Server, the 3DS SDK, and the 3DS Requestor.
Directory Server Certificate Authority	**DS CA** or **CA DS**	A component that operates in the Interoperability Domain; generates and Certificate Authority (DS distributes selected digital certificates to components participating in 3-D Secure. Typically, the Payment System to which the DS is connected operates the CA.
Directory Server ID		Registered Application Provider Identifier (RID) that is unique to the Payment System. RIDs are defined by the ISO 7816-5 standard.
Electronic Commerce Indicator	**ECI**	Payment System-specific value provided by the ACS to indicate the results of the attempt to authenticate the Cardholder.
Frictionless		Used to describe the authentication process when it is achieved without Cardholder interaction.
Frictionless Flow		A 3-D Secure flow that does not involve Cardholder interaction as defined in EMVCo Core Spec Section 2.5.1.
Message Authentication Code	**MAC**	A symmetric (secret key) cryptographic method that protects the sender and recipient against modification and forgery of data by third parties.
Merchant		Entity that contracts with an Acquirer to accept payments made using payment cards. Merchants manage the Cardholder online shopping experience by obtaining the card number and then transfers control to the 3DS Server, which conducts payment authentication.
Non-Payment Authentication	**NPA**	3DS authentication type with no transaction attached, used for identity verification
One-Time Passcode	**OTP**	A passcode that is valid for one login session or transaction only, on a computer system or other digital device.

Term	Acronym	Definition
Out-of-Band	**OOB**	A Challenge activity that is completed outside of, but in parallel to, the 3-D Secure flow. The final Challenge Request is not used to carry the data to be checked by the ACS but signals only that the authentication has been completed. ACS authentication methods or implementations are not defined by the 3-D Secure specification.
Preparation Request Message	**PReq**	3-D Secure message sent from the 3DS Server to the DS to request the ACS and DS Protocol Versions that correspond to the DS card ranges as well as an optional 3DS Method URL to update the 3DS Server's internal storage information.
Preparation Response Message	**PRes**	Response to the PReq message that contains the DS Card Ranges, active Protocol Versions for the ACS and DS and 3DS Method URL so that updates can be made to the 3DS Server's internal storage.
Proof or authentication attempt		Refer to Attempts.
Registered Application Provider Identifier	**RID**	Registered Application Provider Identifier (RID) is unique to a Payment System. RIDs are defined by the ISO 7816-5 Standard and are issued by the ISO/IEC 7816-5 Registration Authority. RIDs are 5 bytes.
Results Request Message	**RReq**	Message sent by the ACS via the DS to transmit the results of the authentication transaction to the 3DS Server.
Results Response Message	**RRes**	Message sent by the 3DS Server to the ACS via the DS to acknowledge receipt of the Results Request message.

Document control

Release notes

ActiveServer 1.3.0

[Release Date: 07/02/20]

Change	Description
[#347] Enhancement	Added support for using AWS KMS as an encryption type
[#375] Fix	Fixed an issue that potentially allowed unauthorised access to the login page via the Admin API
[#589] Enhancement	Added a new optional field (addrMatch) to the Auth API v1, allowing the user to specify if the cardholder billing and shipping addresses match
[#621] Enhancement	Added support for the PRes cache to handle up to 19 digit PANs when calling the /enrol API
[#637] Enhancement	Improved keystore handling process to prevent potential conflicts
[#639] Enhancement	Added v2 of the authentication API, including changes to PAN storage and encryption keys - full PAN is no longer stored, only a truncated version for v2 transactions
[#642] Enhancement	Improved error handling for HTTP 302 redirections on administration interface
[#644] Enhancement	Improved the performance of the PReq message process for card range caching
[#652] Change	Authentication Value provided as the proof of authentication is now only stored for 7 days, after which it is masked
[#656] Enhancement	Added support for specifying the folder for log file collection
[#657] Fix	Fixed an issue that could cause transaction processing to stop when no threeDSMethodData was received

Change	Description
[#660] Enhancement	Improved performance for database connection pool
[#679] Change	PANs in the system now show the first 6 and last 4 digits, with the remaining digits being truncated and masked
[#682] Change	Changed the s3.bucket-name property path setting to be more flexible
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.2.2

[Release Date: 21/11/19]

Change	Description
[#167] Enhancement	Enhanced 3DS Method notification process to be more robust
[#627] Change	Changed security header policy to apply to both HTTP and HTTPS
[#628] Fix	Fixed an issue with content security policy header for administration UI
[#629] Fix	Fixed an issue with message validation for field authenticationType

ActiveServer v1.2.1

[Release Date: 15/11/19]

Change	Description
[#584] Enhancement	Normalised HTTP response status for server requests
[#586] Enhancement	Added additional HTTP headers to enhance page security
[#616] Fix	Fixed an issue with currency code exponent for UAH (980)

Change	Description
[#617] Fix	Fixed an issue when searching for merchants using an acquirer BIN

ActiveServer v1.2.0.1

[Release Date: 04/11/19]

Change	Description
[#610] Fix	Fixed an issue with Oracle database initialisation

ActiveServer v1.2.0

[Release Date: 01/11/19]

Change	Description
[#293] Enhancement	Added the payTokenInd to Auth APIs to support the conditional EMVCo field EMV Payment Token Indicator
[#351] Change	Merchants must now be created or edited to have a unique combination of Merchant name and Merchant ID
[#404] Fix	Fixed an issue for users with a merchant role being unable to access dashboard
[#494] Change	Removed padding from Base64url encoding as per EMVCo bulletin
[#542] Enhancement	Added support for importing Merchant and Acquirer profiles from ActiveMerchant
[#546] Change	Purchase amount on transaction reports are now shown and searched for in major units rather than minor units
[#561] Enhancement	Improved indexing for database table performance
[#581] Enhancement	Added a warning dialogue to restart instance when a DS certificate is added

Change	Description
[#583] Enhancement	Added a new Admin URL setting to allow separate access to the administration interface
[#590] Enhancement	Improved the process of keystore initialisation during server startup
[#599] Change	Removed the global settings for Cache refresh interval, Preparation Response (PRes) timeout and Preparation Response (PRes) timeout. These settings can still be managed per card scheme on the DS settings page
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.1.4

[Release Date: 27/09/19]

Change	Description
[#559] Enhancement	Updating the External URL will now automatically update all 3DS Server URLs in the Directory Server settings if they have an empty value
[#579] Fix	Fixed database index errors that occurred during Mastercard automated compliance testing
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.1.3

[Release Date: 20/09/19]

Change	Description
[#573] Fix	Fixed an issue concerning key generation for certain HSMs
[#574] Enhancement	Added a confirmation dialogue when rotating keys
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.1.2

[Release Date: 19/09/2019]

Change	Description
[#383] Enhancement	The ActiveServer EULA is now accessed from the administration UI about page and has been removed from the release package
[#424] Change	Managing Acquirer BINs via the Admin API now uses string values rather than UUID's of Acquirers in the system. As such, the Acquirer Admin API endpoints have been removed. The administration UI now takes either an existing Acquirer BIN or a value can be entered
[#450] Change	Setting the admin.port now restricts all administration interface UI requests to that port number
[#507] Enhancement	Added dsTransID and messageVersion to API responses for BRW, APP and 3RI channels
[#519] Enhancement	Added a Master Auth API client certificate which can be used to authenticate on behalf of any merchant in the system
[#547] Enhancement	Added additional warning dialogues for users when there is a possibility of overriding existing private keys on Directory Server certificate page
[#548] Enhancement	Added a new challenge status API endpoint (/api/v1/auth/challenge/status), allowing the 3DS Requestor to optionally provide a cancel reason when cancelling a challenge request
[#552] Enhancement	Enhanced the performance of installation wizard
[#555] Change	Changed a listener port opened by ActiveServer to be internally used only
[#557] Change	Removed the CRes and ACS Method timeout settings as they correspond to 3DS SDK timeouts
[#560] Change	Changed the Admin API endpoints for Merchants (certificate export/revoke and key rotate) and removed unused parameters from request and responses. Also removed the Admin API endpoints for settings
[#565] Fix	Fixed an issue where a user was able to exceed the session failed attempts amount

Change	Description
[#569] Fix	Fixed an issue causing the PReq not to be sent if the PReq value was not set in Directory Server settings
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.1.1

[Release Date: 30/08/2019]

Change	Description
[#509] Enhancement	Added a new monitoring endpoint for timing out non-completed transactions to support 3DS Requestor sample code v1.1
[#537] Enhancement	Added an optional merchant name field to authentication APIs to allow the merchant name in a merchant profile to be overridden
[#541] Enhancement	Added sample database connector settings to application-prod.properties for DB2 and PostgreSQL
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.1.0

[Release Date: 16/08/2019]

Change	Description
[#151] Enhancement	Added functionality to import CA certificate chain during client/server certificate installation if included in certificate
[#152] Enhancement	Added functionality to specify a separate PReq endpoint if DS provider requires this setup
[#371] Fix	Fixed a bug causing the administration interface session timeout not to work, this setting is now in the configuration properties

Change	Description
[#425] Change	Changed audit log reports to better show what values have been changed
[#447] Enhancement	RReq and RRes messages are now shown on Transaction Details page
[#461] Enhancement	Added support for PostgreSQL type databases
[#483] Enhancement	Added timed logs for auth API messages for debug log level
[#487] Enhancement	Added functionality to override the 3DS Server reference number when performing Mastercard compliance testing
[#488] Enhancement	Redesigned the DS Certificate page to more easily manage CSRs as well as streamlining buttons
[#493] Change	Default Test Merchant is no longer able to be deleted, as it is used for test purposes
[#497] Enhancement	Added support for DB2 type databases
[#499] Enhancement	Common name of DS CSRs will now be pre-filled if 3DS Server URL is available
[#505] Change	When browser info collecting or the 3DS method is skipped, actual error message with required fields missing is now shown
[#508] Enhancement	Added ECI value to be shown on Transaction Details page
[#516] Change	Changed error message on login page to eliminate risk of username enumeration
[#520] Change	Changed the moment.js file to be loaded locally rather than from an external CDN
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.0.5

[Release Date: 04/07/2019]

Change	Description
[#322] Fix	Fixed issue that could cause times and dates on administration interface to not display in users local time zone (set from user profile)
[#378] Enhancement	Added functionality to download CA certificate bundle from merchant details page
[#401] Change	For new installations, changed the default system keystore filename pattern to be as_sys_"randomUUID.jks"
[#402] Fix	Fixed issue causing "3DS Server Transaction ID", "Min purchase amount", "Max purchase amount" not to display correct transaction search results
[#412] Fix	Fixed issue causing a user to not lock after exceeding maximum password attempts
[#422] Fix	Fixed issue causing incorrect value to be displayed for Directory Servers > Settings > HTTPS callback port
[#428] Change	Updated /api/v1/auth/3ri auth API request to require a {messageCategory}
[#433] Change	Removed .html suffix from all pages
[#446] Enhancement	Improved error messages for invalid values on merchant details page
[#448] Enhancement	Improved logic and error handling for importing Directory Server certificates
[#449] Enhancement	Changed system labels for improved readability - Directory Server > Settings > 3DS Server URL (previously External URL), Directory Server > Settings > HTTP listening port (previously HTTPS callback port), Settings > 3DS2 > API URL (previously Auth API URL)
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.0.4

[Release Date: 31/05/2019]

Change	Description
[#386] Fix	Fixed an issue that could cause an error during the activation process when a HSM is being used
[#390] Enhancement	Added functionality to change the HSM PIN via the Settings > Security page
[#380] Enhancement	Added Amazon Aurora MySQL 5.7 to compatible databases

ActiveServer v1.0.3

[Release Date: 27/05/2019]

Change	Description
[#376] Change	Updated `enrol` API response to provide result enumeration as `00` or `01` values
[#379] Fix	Fixed issue that could cause dashboard historical data not to display
[#380] Fix	Fixed issue causing merchants with old DS enum values to show an error when accessed

ActiveServer v1.0.2

[Release Date: 24/05/2019]

Change	Description
Database Support	Added support for MSSQL Server 2017
[#301] Enhancement	Updated the Admin API endpoints to use .x509 authentication

Change	Description
[#349] Change	Changed log file format from as.dd-mm-yyyy.log to as.yyyy-mm-dd.log and to be stored in base logs folder
[#356] Change	Changed default values for DS ports in application-prod.properties to be in the 9600 range
[#368] Fix	Fixed issue that was causing `enrol` API to return an Internal Server Error
[#373] Enhancement	Added CA certificate download to User Profile page to be used with API requests

ActiveServer v1.0.1

[Release Date: 17/05/2019]

Change	Description
[#326] Fix	Fixed issue causing side menu to load slowly on some browsers
[#327] Fix	Fixed compatibility issue when using Oracle DB
[#328] Change	Added acsReferenceNumber to the AuthResponseApp API
Other	Minor bug fixes, performance and security enhancements

ActiveServer v1.0.0

[Release Date: 09/05/2019]

Change	Description
Release	Initial release

Legal Notices

Confidentiality Statement

Copyright Statement

Disclaimer

Ltd with respect to any products, protocols or standards contained in this document are not to be relied upon.

Liability

In no event shall GPayments Pty Ltd be liable for any special, incidental, indirect or consequential damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) whether in an action of contract, negligence or other tortuous action, rising out of or in connection with the use or inability to use this information or the products, protocols or standards described herein, even if GPayments has been advised of the possibilities of such damages.