Version 1.1.0:1

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**. Please 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**
>
> **Please 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 web administration interface can be accessed using a Chrome, Firefox or Edge browser

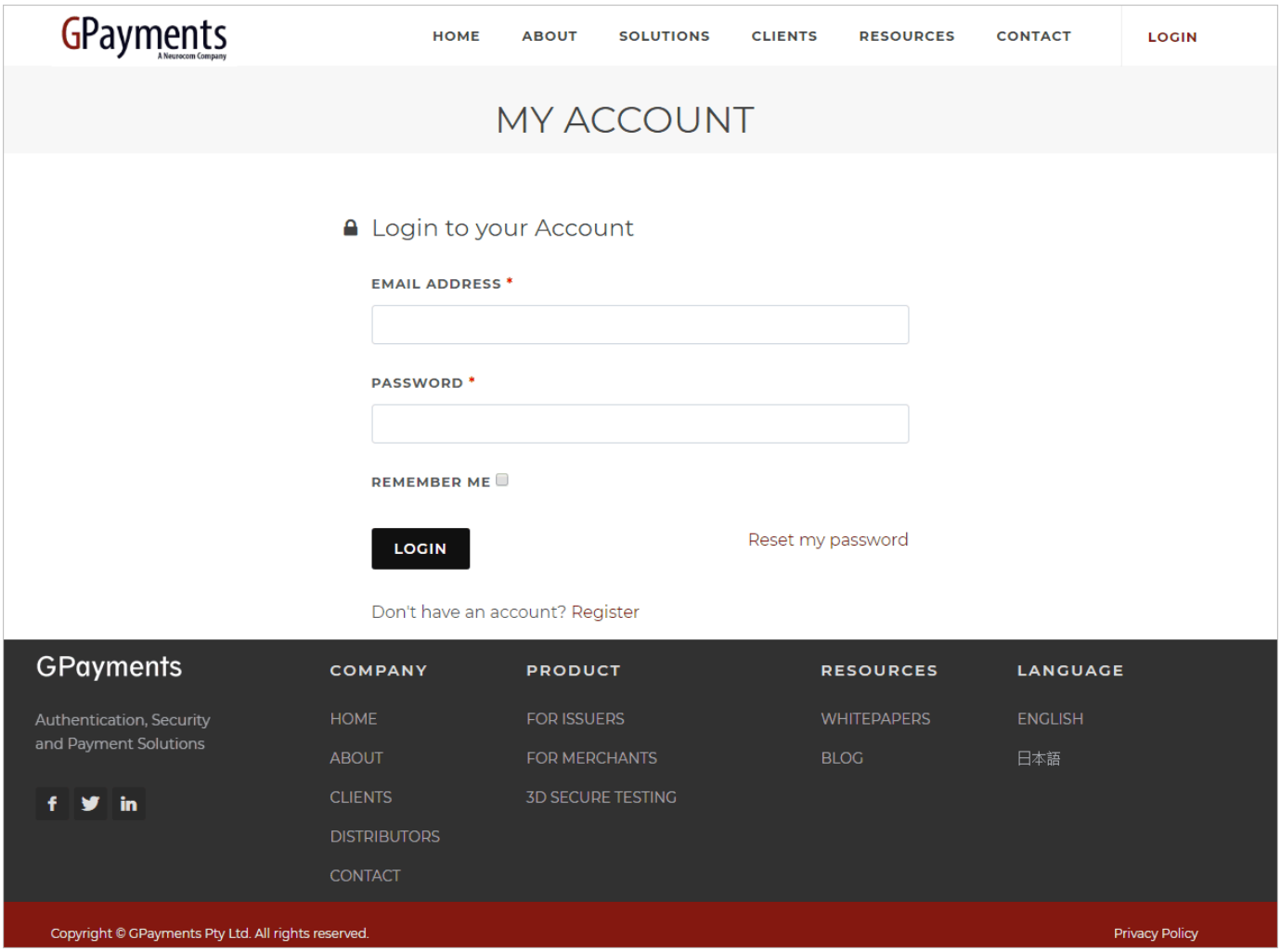
Database:

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

Database	Compatible Versions
PostgreSQL	8.4 and later
DB2	11.1 and later

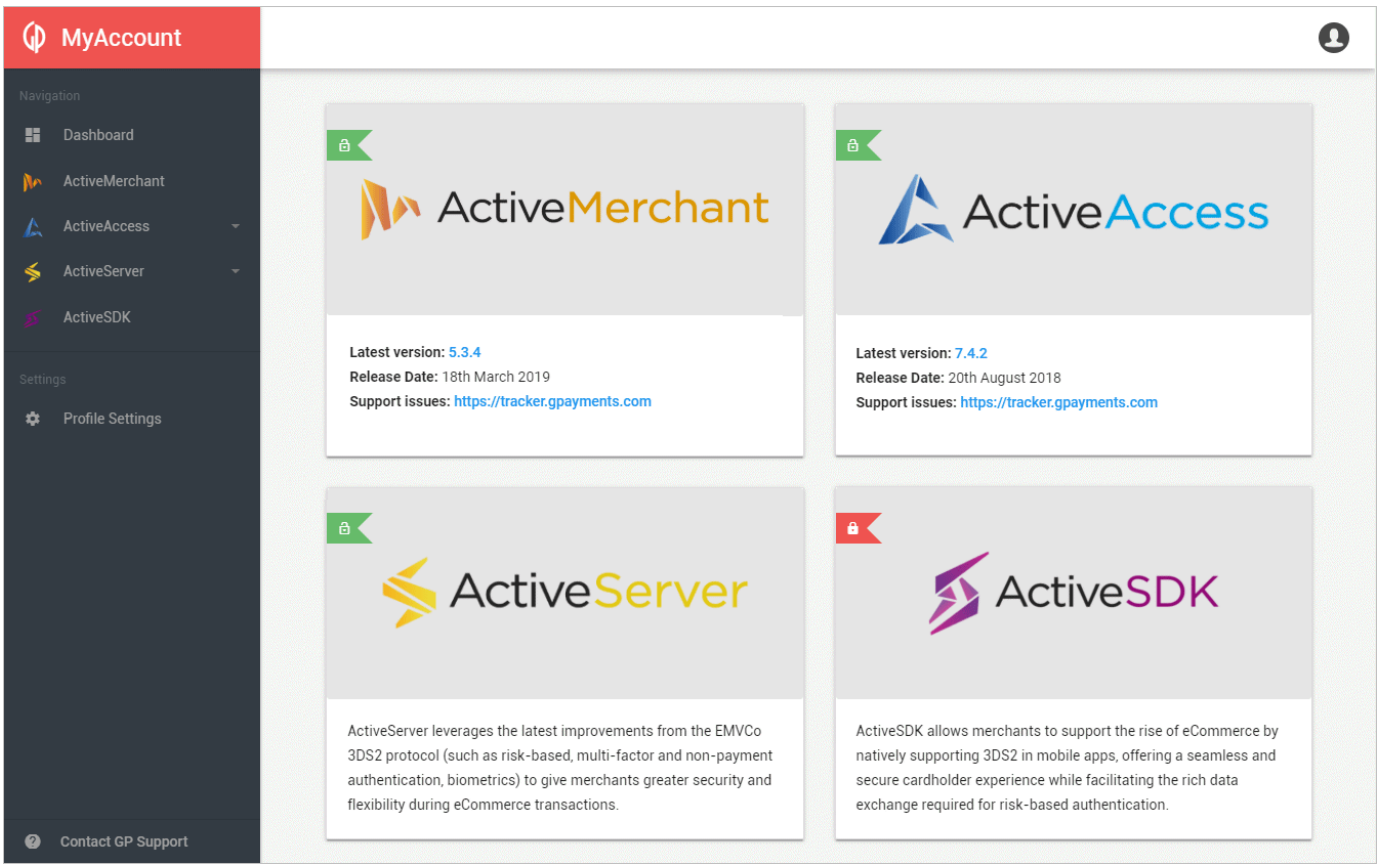
Download ActiveServer

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



If you don't have an account, register at: https://login.gpayments.com/register

Once you are logged in you will see the **MyAccount Dashboard**.



Select **ActiveServer > Download** and download the release package.

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

Once part of an organisation, if your company has already purchased **ActiveServer** you will be able to download the package. If you are unable to download the software and have confirmed

you are in the correct organisation and have already purchased the product, contact techsupport@gpayments.com for assistance. If you would like to talk to our friendly sales team about purchasing **ActiveServer**, you can contact them 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
├── EULA.pdf
├── 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.
- `EULA.pdf` - A copy of the End User License Agreement.
- `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 cleared 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
- Oracle
- MS SQL

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

To use a 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
```

MS SQL

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

MS SQL Database Properties (Example)

```
as.db.vendor=sqlserver
as.db.url=jdbc:sqlserver://<Your MS SQL DB Host>:1433
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.

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. Please 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`.

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.

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>
```

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 Union Pay

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 Union Pay 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`.

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.

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

Set your AWS Bucket path in the following properties:

Amazon S3 keystore (Example)

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

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>
   ```

AWS CREDENTIALS

AWS Credentials include an `access_key_id` and a `secret_access_key`. The AWS Credentials can be set in a number of ways:

1. Set your AWS Credentials in the following properties:

   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>
   ...
   ```

2. 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. 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>
   ```

3. 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.

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. Please consult your HSM's documentation for details on the library that should be use.
- `as.keystore.pkcs11.slot=`
  The slot number on your HSM. Please consult your HSM's documentation for details on the slot number that should be use.

Info

Please 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.

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, please 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, please consult your email server administrator for details.

> **Info**
>
> Please 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.

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

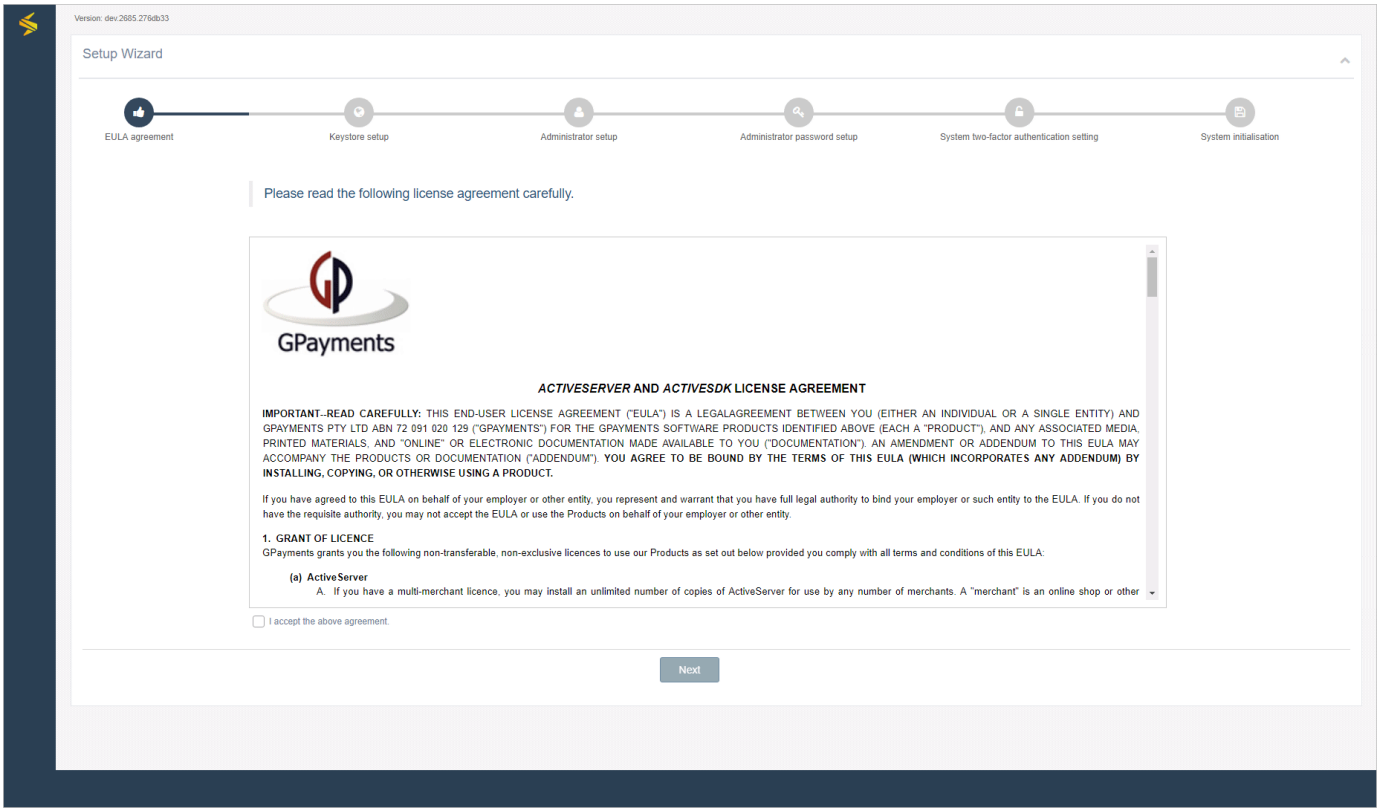
Once ActiveServer is up and running, you can access the Administrator UI via the **Administration** URL.

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:

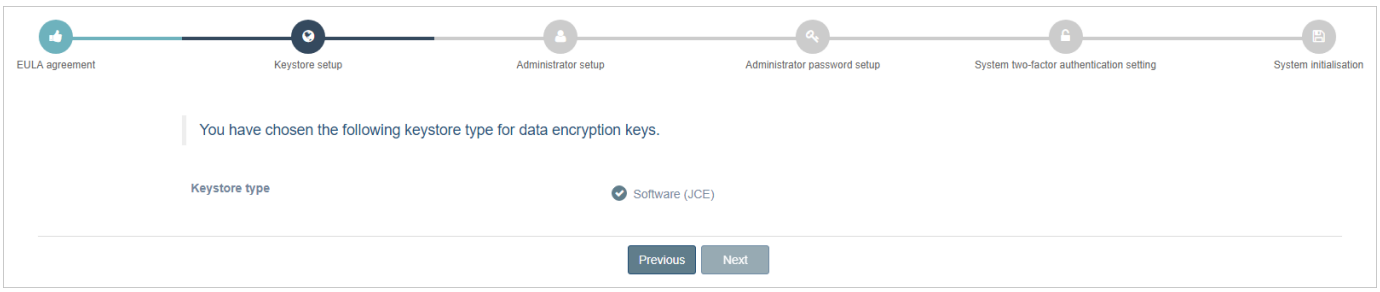
- EULA agreement
- Keystore setup
- Administrator setup
- Administrator password setup
- System two-factor authentication setting
- System initialisation

EULA agreement



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

Keystore setup



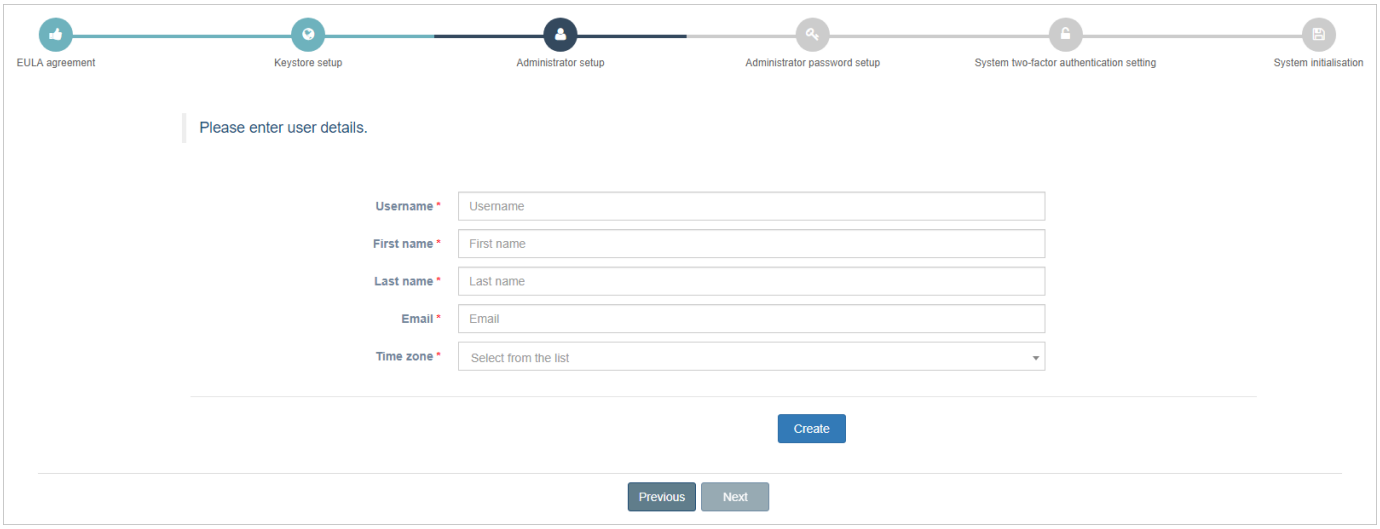
- Select the **Keystore type**.

  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.

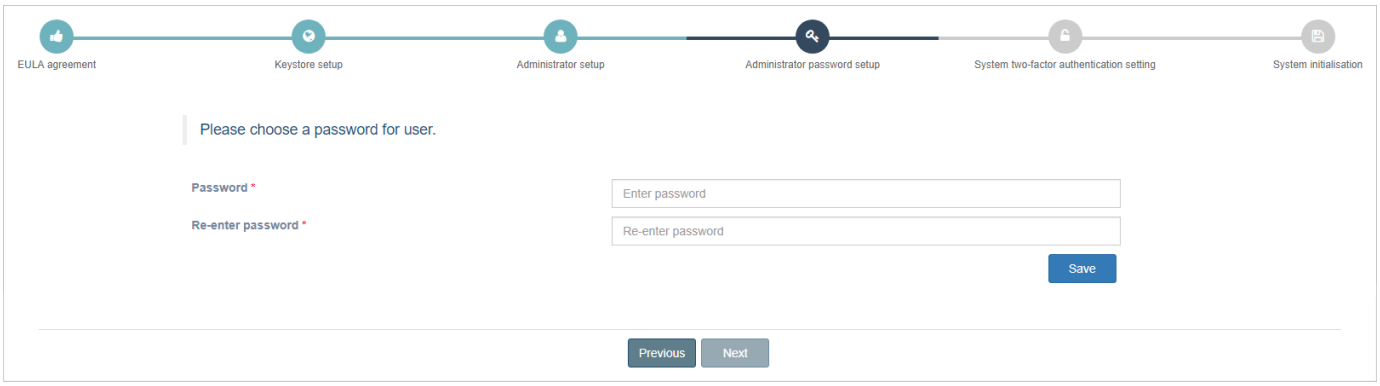
- Select the ***Next*** button to continue.

Administrator setup



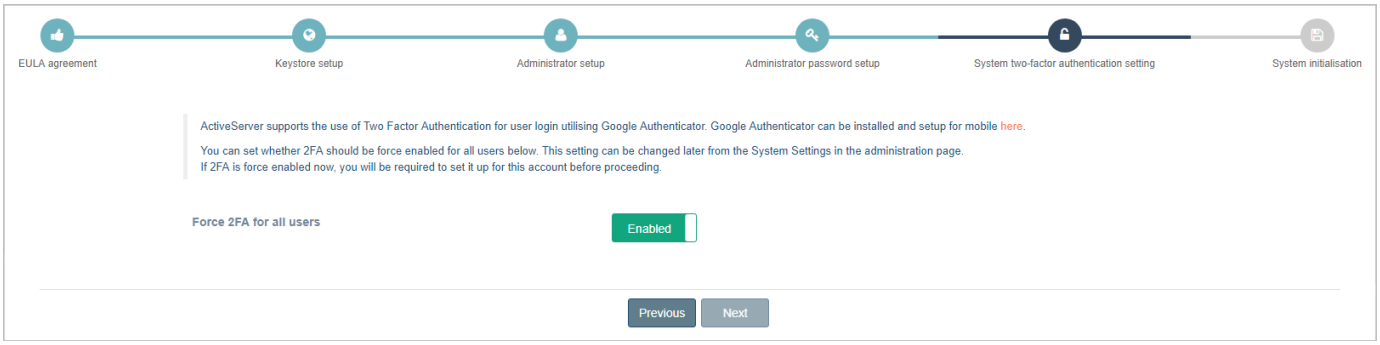
- Enter the user details for your Administrator account.
- Select the ***Create*** button to create the account.
- Select the ***Next*** button to continue.

Administrator password setup



- 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 two factor authentication for signing into the Administrator UI.

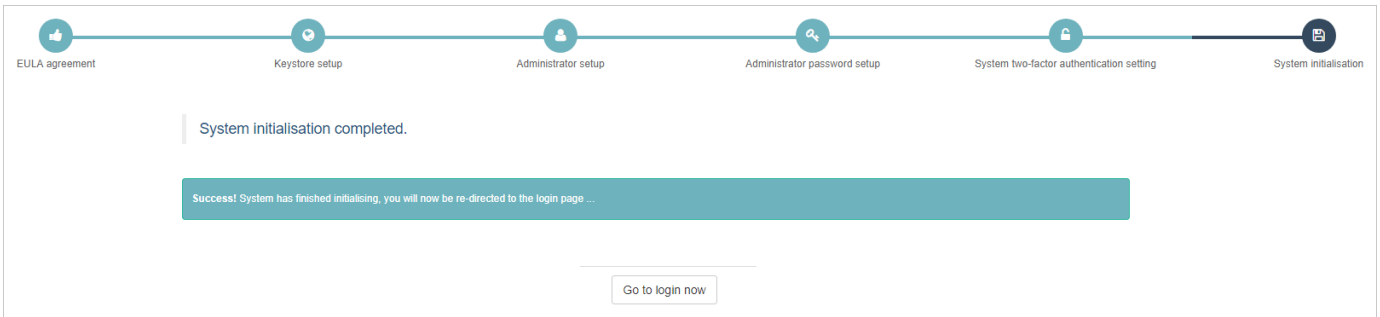
> **Note**
>
> To use this feature, you must have Google Authenticator installed on a mobile device.
>
> Refer to Install Google Authenticator for setup instructions for Google Authenticator.

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

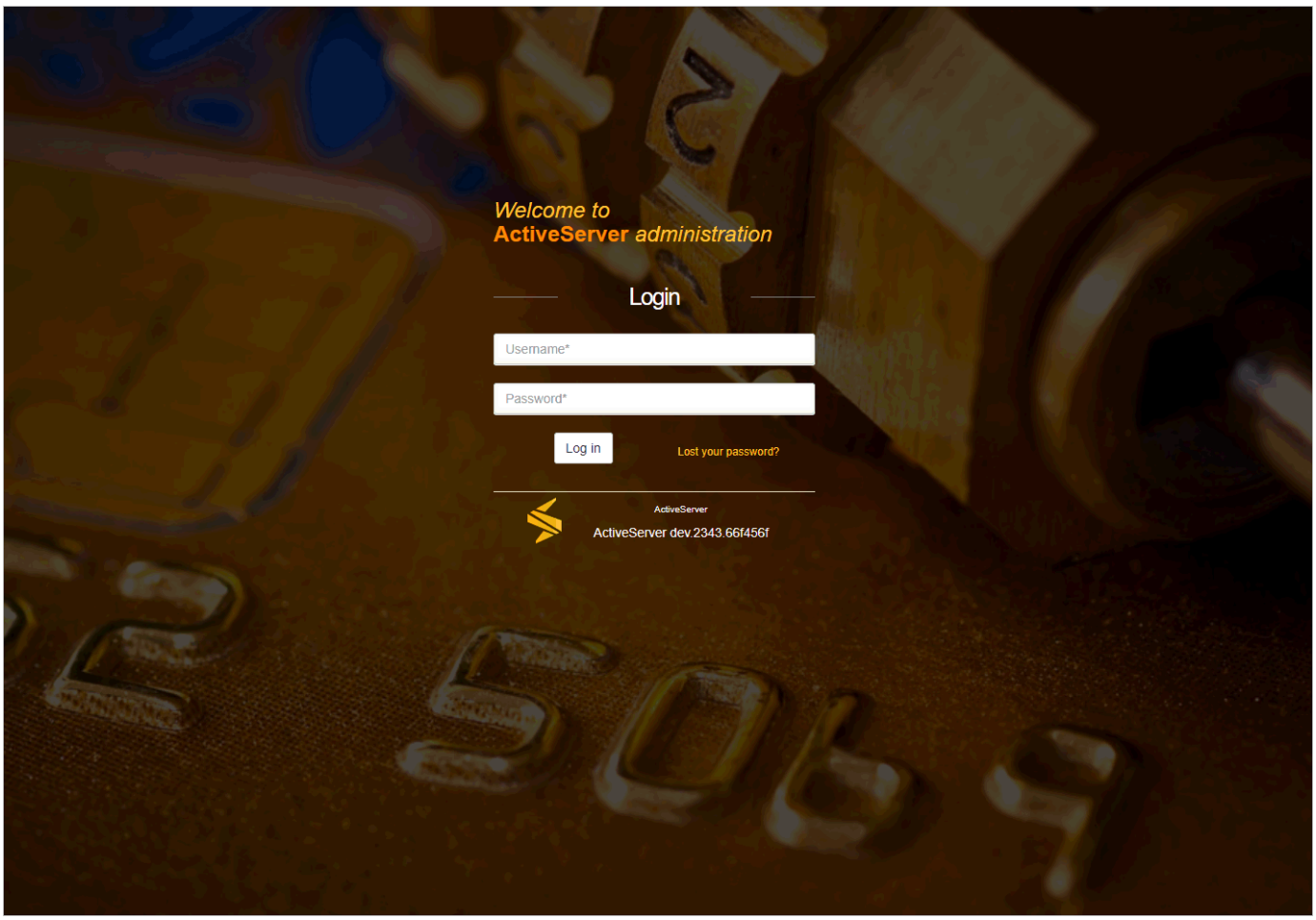
To mandate two factor authentication for all users:

- ***Enable*** the toggle, adjacent to **Force 2FA for all users**.
- Select the ***Next*** button to continue.

System initialisation



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

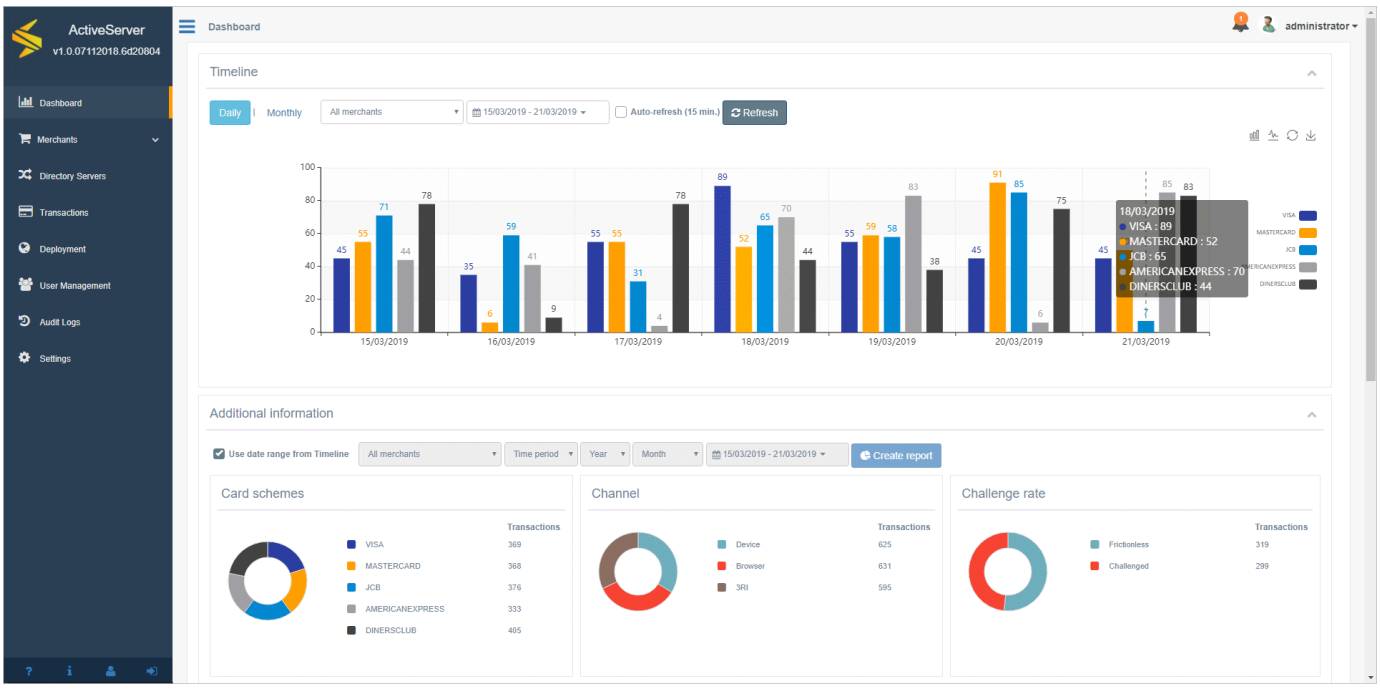
Listed below is the overview for all **ActiveServer** functionality, including user interface outline. Under each section is a link to the relevant guides that will assist you with performing tasks.

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.



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

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.

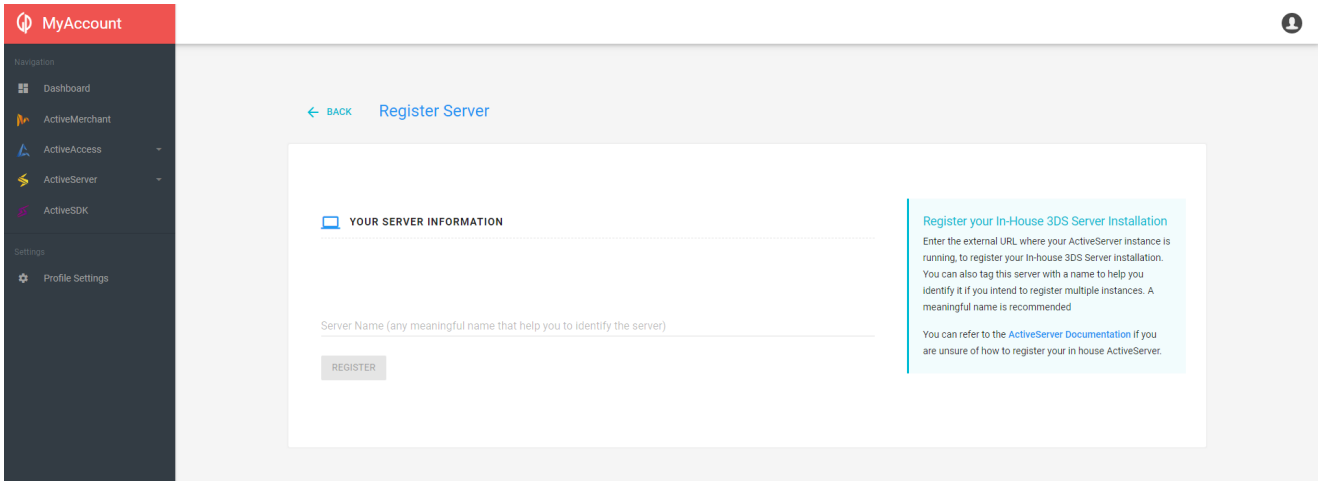
3. Configure the External URL and Auth API URL

1. On the administration interface, navigate to **Settings > 3D Secure 2** 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`.
    - **Auth API URL** - URL in which the authentication API is accessed. This URL domain name is used to issue the client certificates, which are used to authenticate the merchants making the authentication API calls. By default, if this URL is not specified it will use the **External URL**, set during activation, to issue the client certificate.
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**.



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

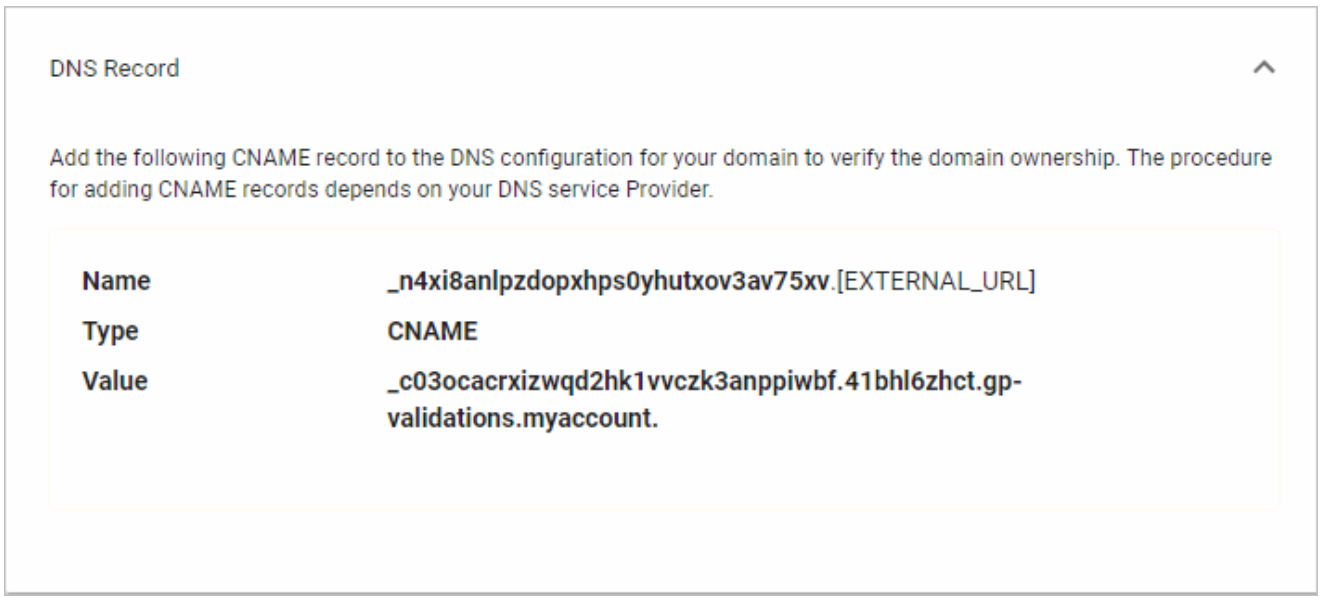
If you choose this method, make sure the **External URL** you specified in the previous step is publicly accessible.

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:



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. 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

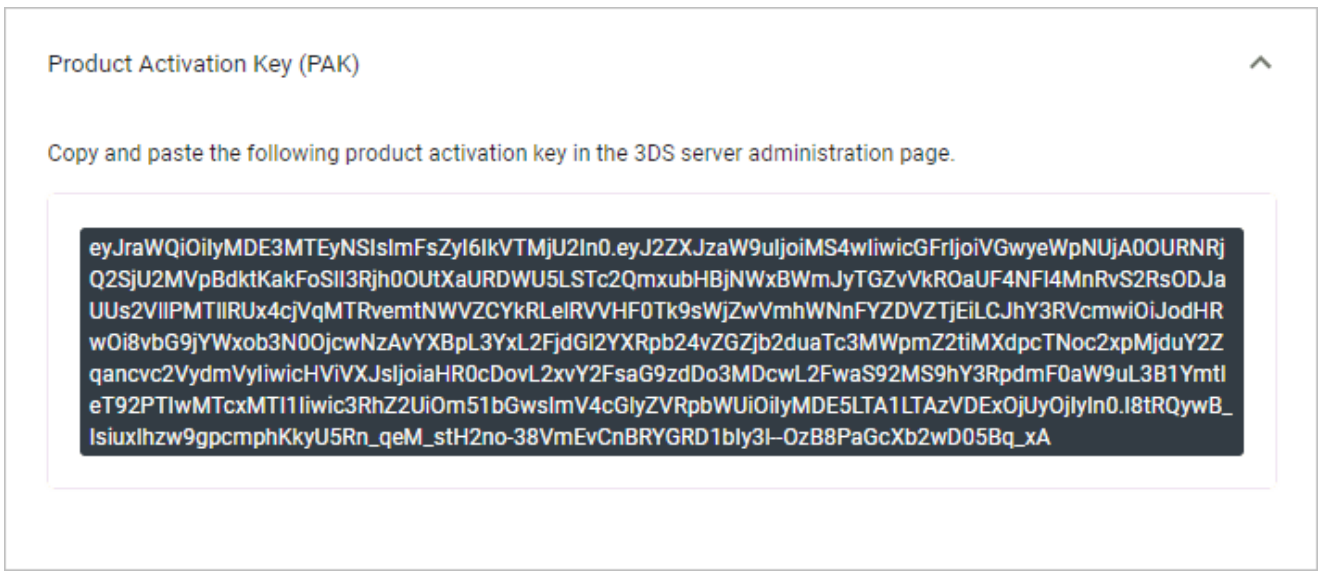
ID	Name	Mandatory	Group	Comments
AD003	OS Version	C	Core	Version of the OS, e.g. 16.04.5 LTS
AD004	Database Name	C	Core	Name of the database e.g. MySQL
AD005	Database Version	C	Core	Version of the database e.g 5.7
AD006	Java Edition and Version	C	Core	Edition of version of Java used e.g. OpenJDK 1.8.120
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.



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.

Upgrade instance

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. 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.

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**.

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				✔	✔
		Create multiple merchants	All merchants			✔		

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
		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			✔		
		Download all merchant certificates	All merchants			✔		

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
		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				✔		
		Delete acquirer				✔		

Page	Sub page	Permission	Scope	System Admin	User Admin	Business Admin	Merchant Admin	Merchant
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		✔	✔	✔	✔	✔

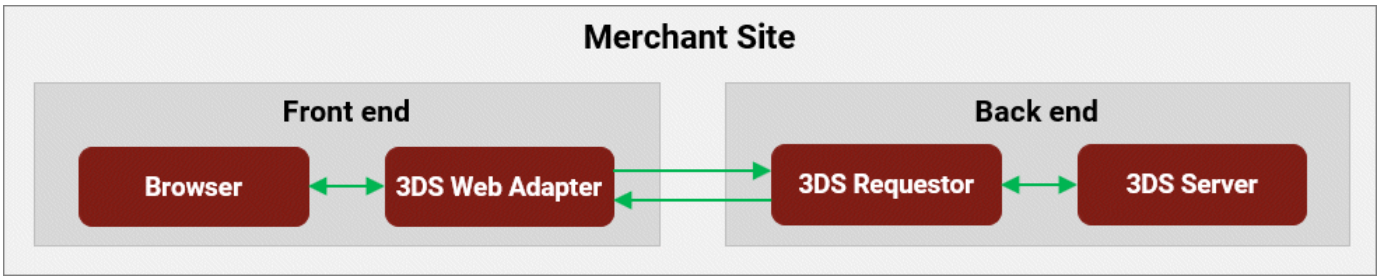
Integration overview

To integrate 3DS2 authentication with a merchant, or a payment gateway, eCommerce site, the backend system of the eCommerce site needs to implement **ActiveServer's** Authentication API.

API calls are operations that an application can invoke at runtime to perform certain tasks. All API requests are made in JSON format, which is a lightweight format for transporting data. For details of the API documentation, please refer to the API document overview.

This section provides an introductory guide on how to integrate your merchant **web server** to connect with **ActiveServer**, and perform a test transaction. For information regarding merchant **App** integration, please 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 the 3DS Server:



- **3DS web adapter** - The 3DS web adapter is the merchant site 3DS2 component and is used to pass 3D Secure data from the consumer device to the 3DS Requestor. For example, the 3DS web adapter can be a `javascript (.js)` that performs responses to the actions in the browser and sends 3DS authentication requests to the 3DS Requestor.
- **3DS Requestor** - The 3DS Requestor is a controller and is used as a bridge between the 3DS web adapter and the 3DS Server. It receives the 3DS authentication requests from the 3DS web adapter, formulates the requests, and sends the requests to the 3DS Server. It also receives the authentication results from the 3DS Server and forwards the results to the 3DS web adapter.

Making a transaction

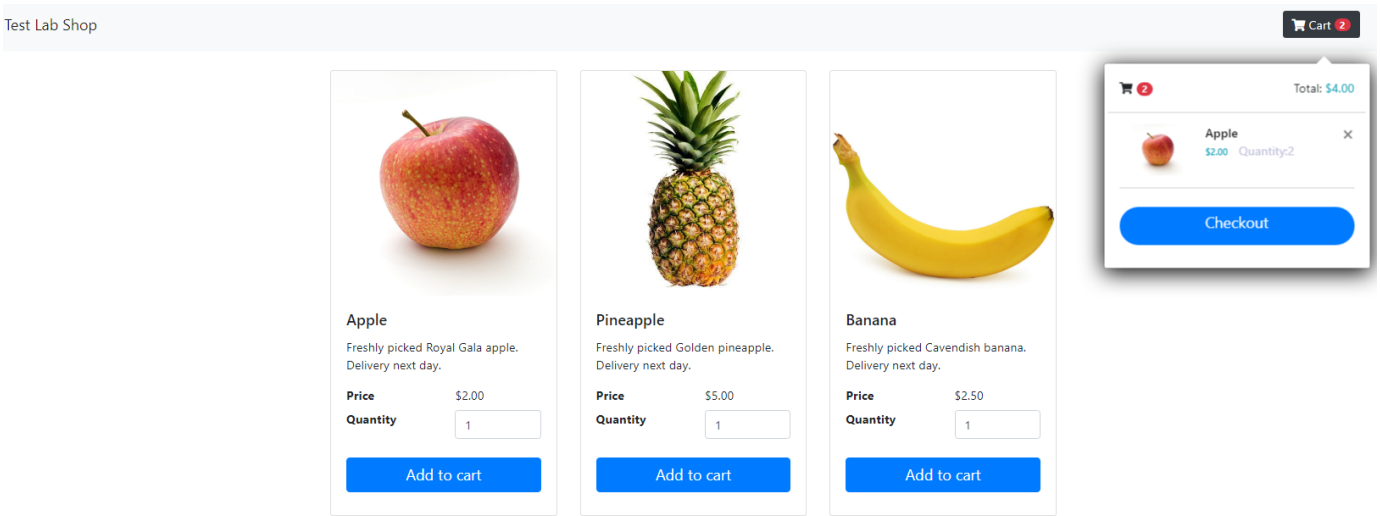
To simulate a transaction with 3DS2, you can use this simple merchant website to see how the Authentication API works.

> **Tip**
>
> As this merchant website is used in examples throughout this integration guide, please try using it before continuing.

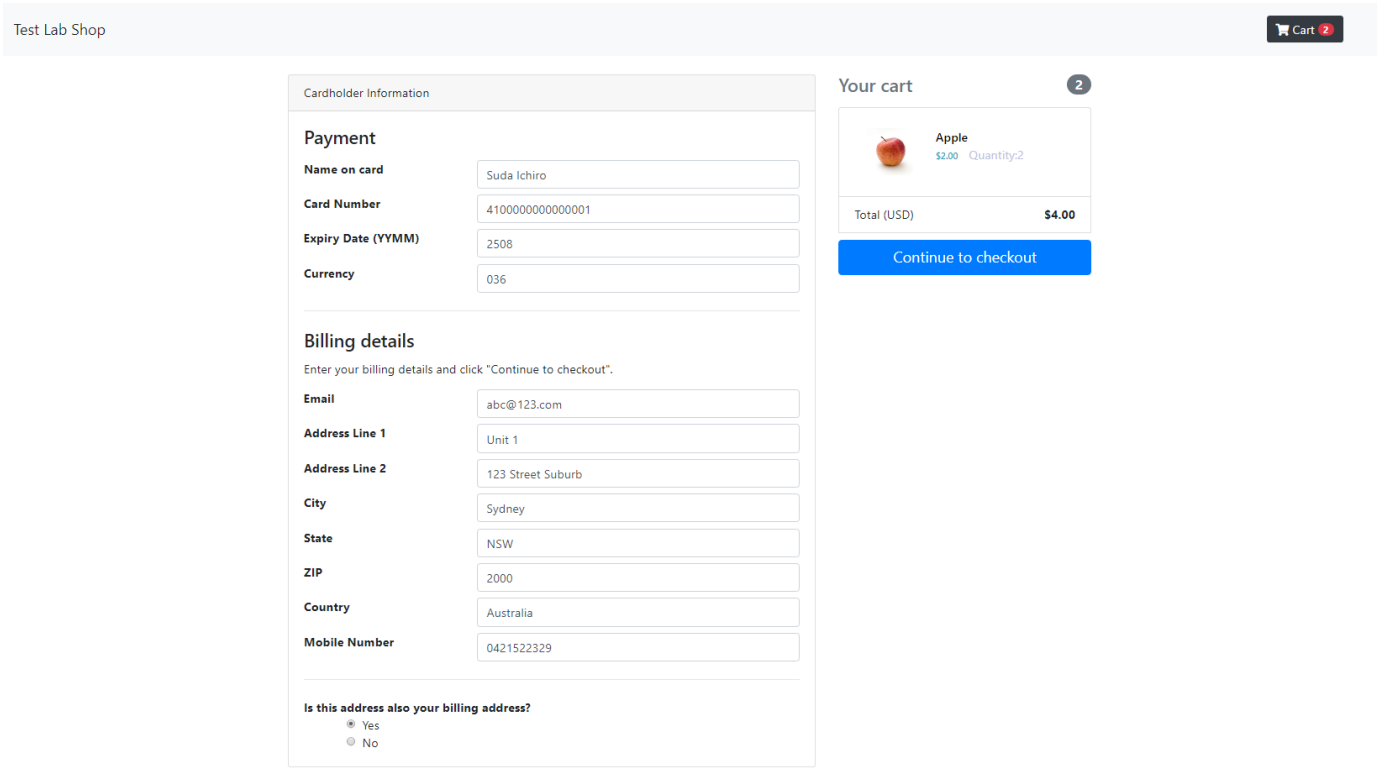
Frictionless flow

To initiate a frictionless transaction, open the merchant website and add an item to the cart.

Select the **Cart** icon in the top right to display the cart contents.

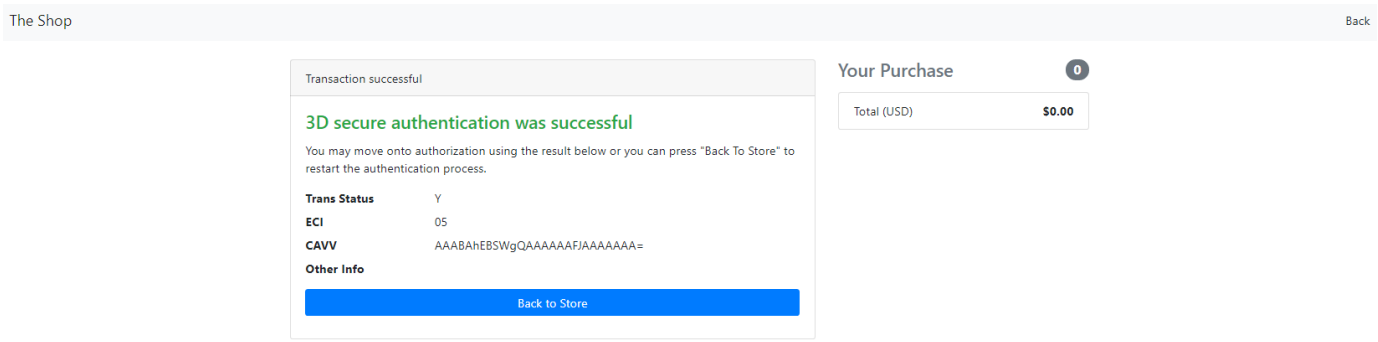


Select the ***Checkout*** button to move to the checkout page.



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

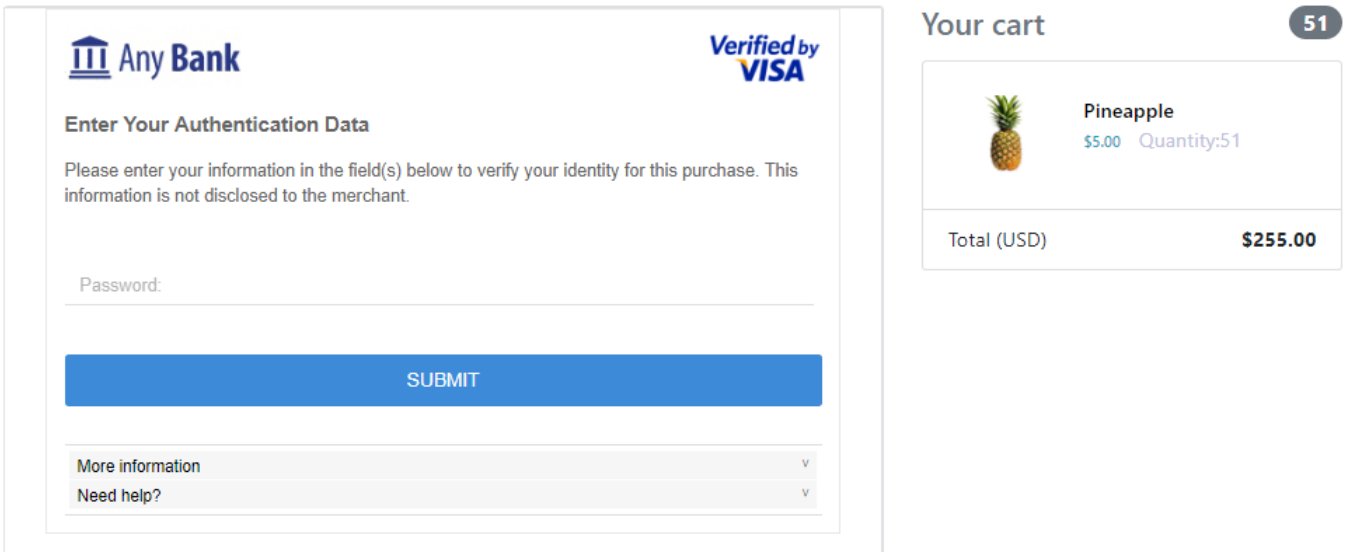
The **3DS web adapter** will collect the cardholder information and send it to the **3DS Requestor**. The **3DS Requestor** will formulate this into an API request and forward it to the **3DS Server**, which will initiate 3DS2 messaging. The **3DS Requestor** will then wait for the authentication result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



This completes a transaction using **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 Store*** button and again add an item to the cart and go to the checkout page. This time, use the card number **4100000000000050** and continue to 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**.



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*** in the footer below to learn more about the **Authentication processes**.

Authentication processes

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 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 ***Continue to checkout***, the **3DS web adapter** sends an `initialise authentication` message to the 3DS Requestor. The `initialise authentication` message contains a unique **transaction ID (`transId`)** plus the cardholder's information. The 3DS Requestor receives the `initialise authentication` message, formats it according to the 3DS Server API, and sends it to the 3DS Server by calling: **`/api/v1/auth/brw/init/`** .

> **Note**
>
> - `/auth` indicates the request is an authentication request.
> - `/brw` indicates the request is from a browser. The 3DS Server also accepts requests from mobile applications via `/app` .
> - `/init` indicates the request is to initialise the authentication.

When **ActiveServer** receives the **`/api/v1/auth/brw/init/`** message, it will collect the browser information and be ready for the authentication process.

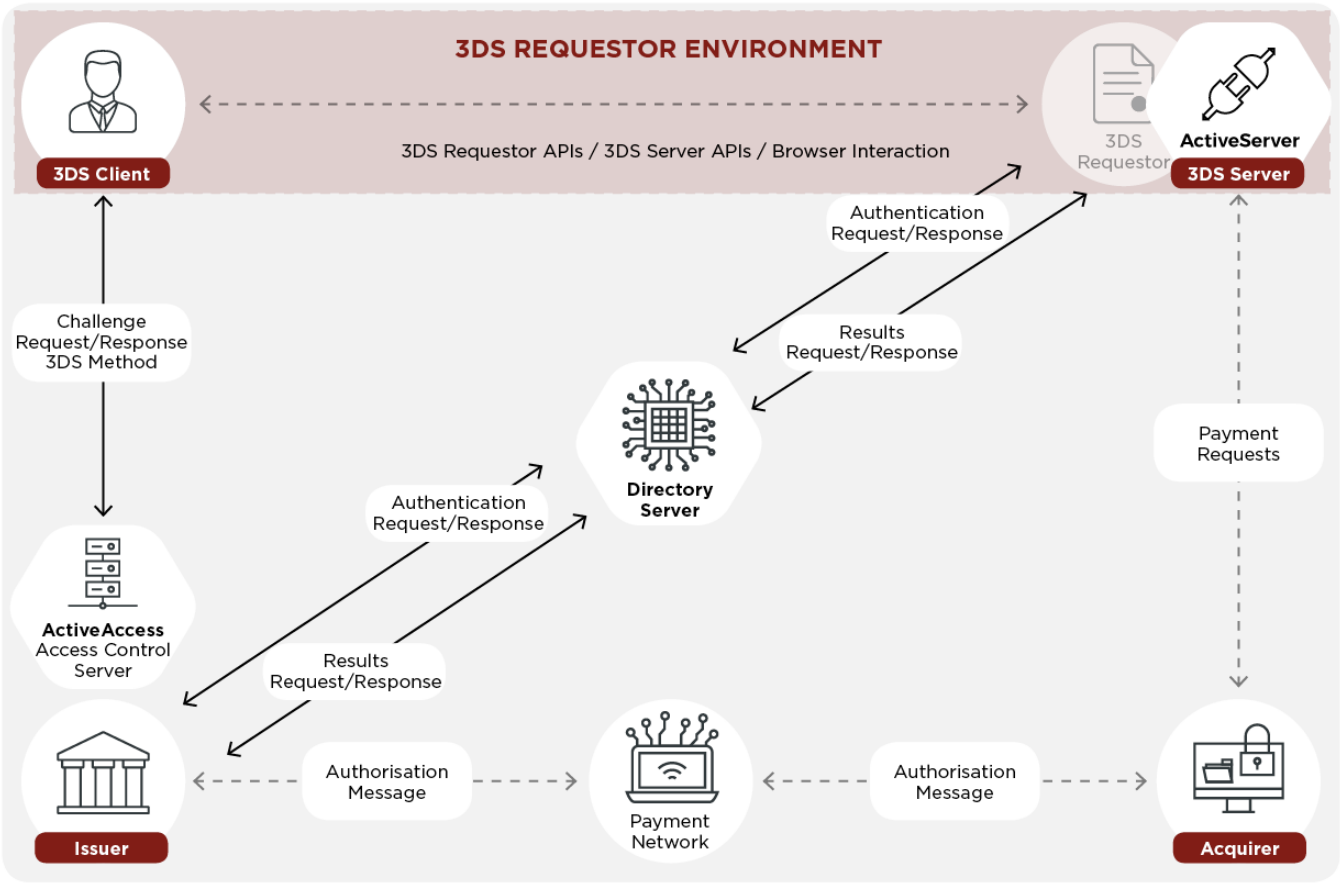
> **Note**
>
> The 3DS Server and the ACS automatically collect the browser information, and this process is **NOT** part of the **3DS Requestor**.

Process 2: Execute Authentication

Once the browser information collection is complete, authentication can be executed by the merchant by calling `/api/v1/auth/brw`. This will cause **ActiveServer** to initiate the 3DS2 messaging process. 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, **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:



> *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 calls `/api/v1/auth/brw/result` to 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.

What's next?

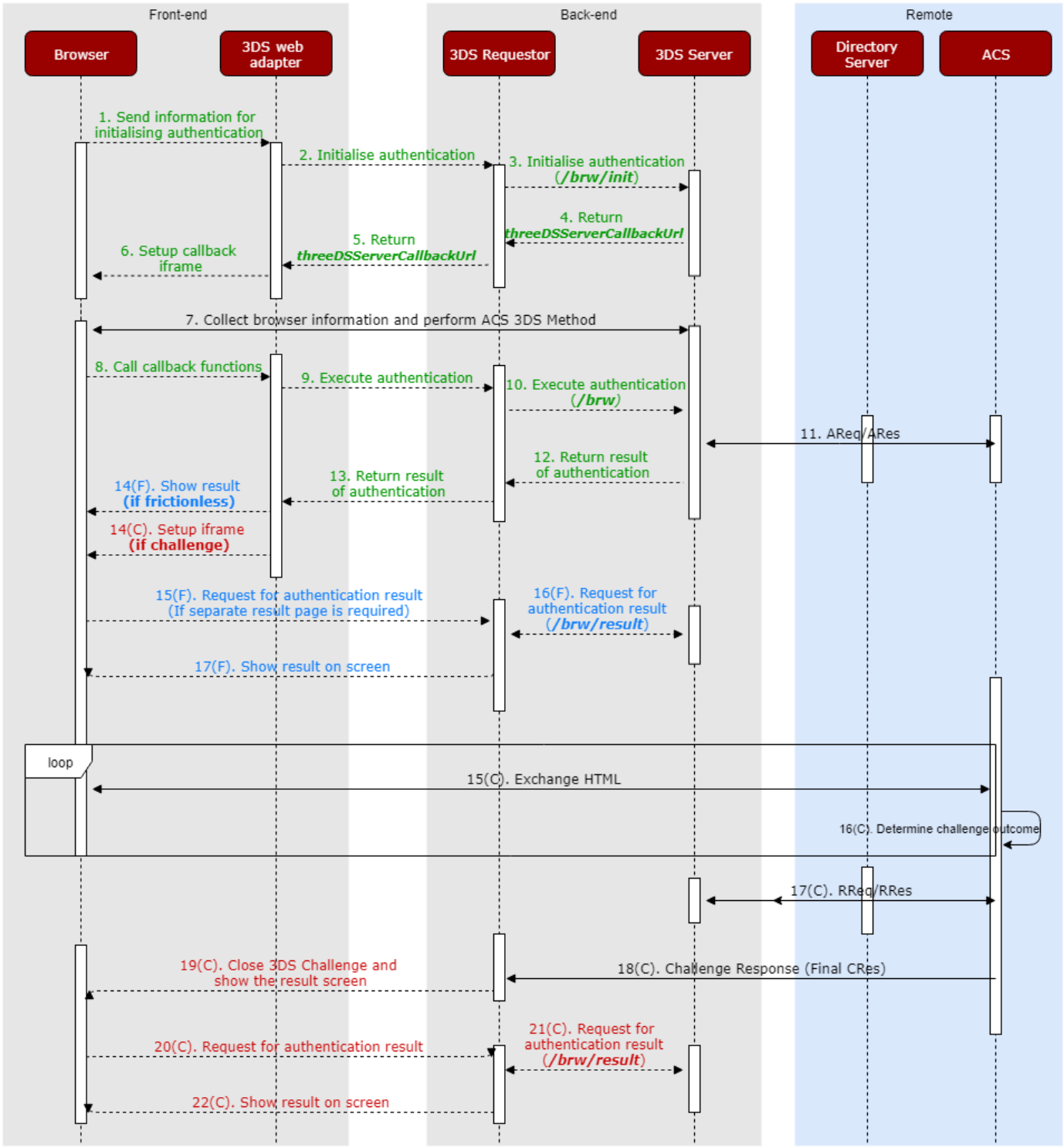
Select **Next** in the footer below to learn more about the **Authentication sequence**.

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.

> **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)



- 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**

   - Information about the cardholder obtained from the checkout page, such as the card number and the cardholder name, 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 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 (`/brw/init/{messageCategory}`)**
    - The **3DS Requestor** obtains the information from the front end and makes a POST API call to `/brw/init/{messageCategory}` to initialise authentication.
    - An important field sent here is the `eventCallbackUrl` , which will be required to start Step 8 to allow the **3DS Server** to callback to this URL to notify the end of the browser information collection.
4. **Return `threeDSServerCallbackUrl`**
    - A successful response from `/brw/init/{messageCategory}` contains `threeDSServerCallbackUrl` and `threeDSServerTransID` .
5. **Return `threeDSServerCallbackUrl`**
    - The **3DS Requestor** returns the `threeDSServerCallbackUrl` back to the **3DS web adapter**.
6. **Setup callback `iframe`**
    - Insert a hidden `iframe` , with `src` set to `threeDSServerCallbackUrl` . This will allow the **3DS Server** to connect to the **3DS Requestor**. The **3DS Server** will callback to this `iframe` .
7. **Collect browser information**
    - The **3DS Server** 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**
    - 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**
    - `callbackFn` can be either `_on3DSMethodSkipped()` or `_on3DSMethodFinished()`, both of which will end up calling `doAuth()`. 3DS-web-adaptor 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 (`/brw`)**
    - The **3DS Requestor** will make a call to `/brw`, which will initiate the authentication processes.
11. **AReq/ARes**
    - An Authentication Request (AReq) is sent from the **3DS Server** via the Directory Server to an ACS. An Authentication Response (ARes) containing the authentication results is sent from the ACS to the **3DS Server**.
12. **Return result of authentication**
    - `/brw` returns a `tranStatus` to the **3DS Requestor**.
13. **Return result of authentication**
    - 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)**

- 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)**

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

16(F). **Request for authentication result (`/brw/result`)**

- The **3DS Requestor** will ask for a result receipt from the **3DS Server** by calling `/brw/result` .

17(F). **Show result on screen**

- The result screen is rendered using the authentication result.

Challenge flow specific

14(C). **Setup `iframe` (if challenge)**

- 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 the **3DS Server**. The **3DS Server** 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**

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

20(C). **Request for authentication result**

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

21(C). **Request for authentication result (`/brw/result`)**

- The **3DS Requestor** asks for result receipt from `/brw/result`, same as Step 16(F).

22(C). **Show result on screen**

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

> **What's next?**
>
> Select **Next** in the footer below to access the **Integration guide** and go through the process of integrating a merchant checkout process with **ActiveServer**.

Introduction

The **Integration guide** set of documents will take you through the process of integrating the 3DS2 flow with a sample merchant checkout site from the beginning, using **Java** as the backend language.

The sample merchant checkout site provided to is created using Springboot project. For reference, the HTML pages were created using:

- Bootstrap 4.1.3
- Font Awesome 5.2.0
- JQuery 3.3.1

The sample merchant checkout page contains **three main pages**:

- `index.html` - shopping site for choosing items to purchase.
- `checkout.html` - checkout page containing cardholder information and processing screen.
- `result.html` - transaction result page for viewing if transaction was successful or not.

Prerequisites

The following are prerequisites to using this guide:

- Core Java knowledge
- Core web-technologies knowledge (HTML, CSS, Javascript)
- JDK 1.8
- IDE of your choice
- Apache Maven, for installation, please refer to https://maven.apache.org/install.html
- A Git client
- An activated and running ActiveServer instance

Checkout the Sample Code

The 3DS Requestor demo code can be found in this Github repository:

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

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

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

Once the repository is cloned, you will find all the required demo code of this tutorial under directory `3ds-requestor-springboot` .

To run the sample merchant site

The GPayments *sample code package* above contains two folders:

- **initial** - Sample merchant checkout page **without** 3DS2 authentication flow.
- **final** - Final merchant checkout page **with** 3DS authentication flow. This will be the final code that you will have at the end of this step-by-step guide.

To run the sample merchant site with 3DS 2.0 authentication enabled, please follow these steps:

1. Get access to the authentication **API client certificate (.p12 file)**:
    - If testing with a local version of **ActiveServer** - download it from the administration interface under the merchant profile. Please note you need to Activate your ActiveServer Instance first.
    - If testing with **GPayments TestLabs** - use the certificate provided, or **contact GPayments** if you haven't received your certificate file.
2. Copy the downloaded certificate (.p12 file) to the filepath `/final/src/main/resources/certs` to allow mutual authentication between the 3DS Requestor and 3DS Server. Please rename the certificate bundle to `client_certificate.p12` so that the demo code can load the p12 file (the file name is hard coded). Please **contact GPayments** if you encounter problems with activation or certificate downloading.
3. Open a **Terminal** (Linux) or **Command Prompt** (Windows) and navigate to `/final` directory.

4. Execute the following command line on the root directory. Please note that this could take a few minutes the first time its executed while Maven downloads the required dependencies.

```
cd final
mvn spring-boot:run
```

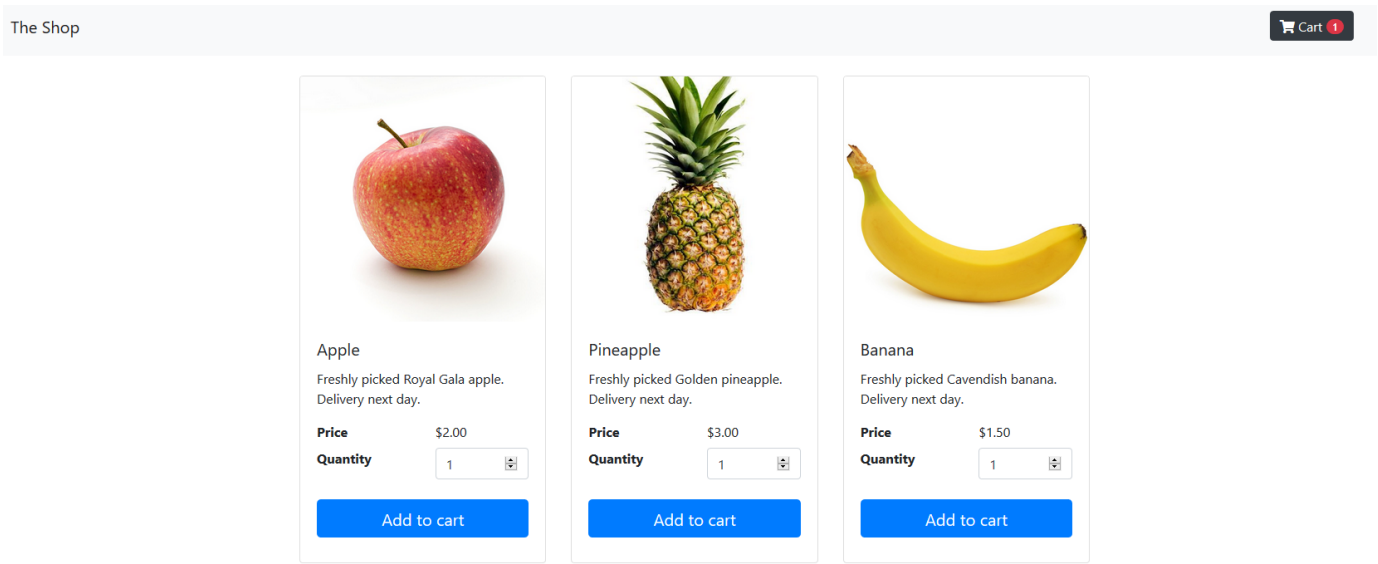
Note

You must ensure Maven is installed and available before running the above command. For Maven installation, please refer to https://maven.apache.org/install.html.

Warning

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

You can view the sample cart page by accessing http://localhost:8082. Try adding some items to the cart and checkout using the default cardholder information.



Note

If port 8082 is in use you may get an error, in which case configure the port by adding the following line to [`/resources/application.properties`] file:

```
server.port = # YOUR_PORT_NUMBER
```

Note

You can also run the initial project which is just an pseudo online shopping site without 3DS 2.0 authentication. To run it, make sure you stop the `final` instance (if you run it already) otherwise the port number will be conflicting.

```
cd initial
mvn spring-boot:run
```

Warning

The sample requestor code is running on HTTP just for demo purposes. The code is not suitable for production.

Depending on the package run, the checkout process will complete with either 3DS2 enabled or disabled.

Whats next?

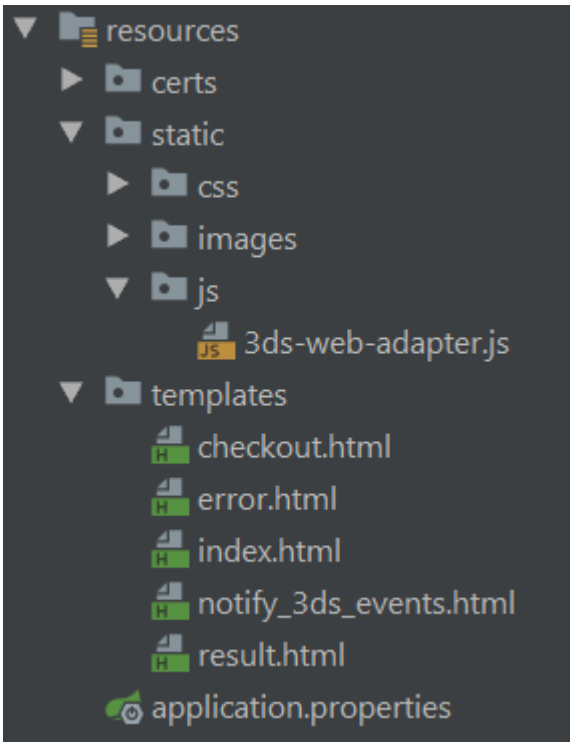
Please select the next button to learn about **Front-end implementation** for a 3DS Requestor.

Front-end implementation

In this section, we show the integration implementation for the front-end of the merchant application using our Sample Code.

> **Note**
>
> Please go through the **Sample Code** and implement that before continuing.

The following diagram shows the main files for the front-end, which can be found in the `/resource` folder in the `/final` project. Please check the **directory tree** if needed. The `3ds-web-adapter.js` is our **3DS-web-adapter**.



Process 1: Initialise the Authentication

To initialise authentication, the front-end needs to:

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

When a user clicks the "**Continue to checkout**" button in the checkout page, it will trigger the browser sending information to the **3DS-web-adapter** for initialising authentication (Step. 1). This is implemented in the `checkout()` function at `checkout.html`.

```
//checkout.html
function checkout() {
    ...
    $("#cardholderInfoCard").remove();
    $("#checkoutCard").removeClass("d-none");

    //move to result screen 2 seconds after
    //setTimeout(function () { window.location.href = "/auth/result"}, 2000);
    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

The `threeDSAuth()` function is implemented in `3ds-web-adapter.js` (Step. 2).

```
//3ds-web-adapter.js
function threeDSAuth(transId, cardHolderInfo, purchaseInfo) {

    var postData = {...}

    ....

    console.log('init authentication');
    $.ajax({
        url: '/auth/init',
        type: 'POST',
        contentType: "application/json",
        data: JSON.stringify(postData),
        success: function (data) {
            console.log('init auth returns:', data);
            $('<iframe id="threeds-container" width="0" height="0"
            style="visibility: hidden;"
            src="' + data.threeDSServerCallbackUrl + '"></iframe>')
                .appendTo('.challengeContainer');
        },
        error: function () {
            alert('error');
        },
        dataType: 'json'

    });

}
```

The `threeDSAuth()` function makes a POST request to the **3DS Requestor** with url `/auth/init` (**line 10**). The object is posted in JSON format. To check the **3DS Requestor** handles this request, please refer here.

> **Note**
>
> `threeDSAuth()` can be modified according to your needs and you can transfer your objects instead of `cardholderInfo`, `purchaseInfo` and `transId`. The data structure can be found in the API document.

After a successful response from `/auth/init` (Step. 5) the `3ds-web-adapter.js` inserts a hidden `iframe` into the checkout page (Step. 6) so that the browser information can be collected by the ACS and 3DS Server using the `threeDSServerCallbackUrl` (Step. 7) **line 16 - 19**.

> **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:

- Implement a `notify_3ds_events.html`.
- Send a `Execute authentication` message to the **3DS Requestor** (Step. 8 and Step. 9).
- 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 html below with variables `transId`, `callbackType` and an optional `param`. To check back-end implementation, please 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" data-th-
value="${callbackName}"/>
    <input type="hidden" id="transId" name="transId" data-th-value="${transId}"/
>
    <input type="hidden" id="param" name="param" data-th-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 checkout 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 `3ds-web-adapter.js` (Step. 8). The value of `callbackName` should be either `_onThreeDSMethodFinished` or `_onThreeDSMethodSkipped`. An explanation of each method is below:

- `_onThreeDSMethodFinished` - notifies that the 3DS method is finished and calls `_doAuth()`, which means browser information collection is finished.
- `_onThreeDSMethodSkipped` - notifies that the 3DS method has been skipped and calls `_doAuth()`, which means browser information collection by the ACS is skipped.

The `_doAuth()` in `3ds-web-adapter` will make a POST request to `/auth` to execute authentication (Step. 9). To check **3DS Requestor** handles the request, please refer here.

```
//3ds-web-adapter.js
function _doAuth(transId) {
    ...
    $.post("/auth", {id: transId}).done(function (data) {
        console.log('auth returns:', data);

        if (!data) {
            alert('error');
            return;
        }

        switch (data.transStatus) {
            case "Y":
                authSuccess(data, transId);
                break;
            case "C":
                startChallenge(data.challengeUrl);
                break;
            ...
        }
}
```

The `_doAuth()` POSTs the request to `/auth` (**line 4**) and gets the result `data`. It then performs different flows based on the returned `transStatus` (**line 14 and 17**).

If the `transStatus` is `Y`, it means the authentication is successful. The **3DS-web-adapter** will switch to the frictionless flow and execute the `authSuccess()` function (Step. 14(F)) which is the next process -> Get Authentication Result.

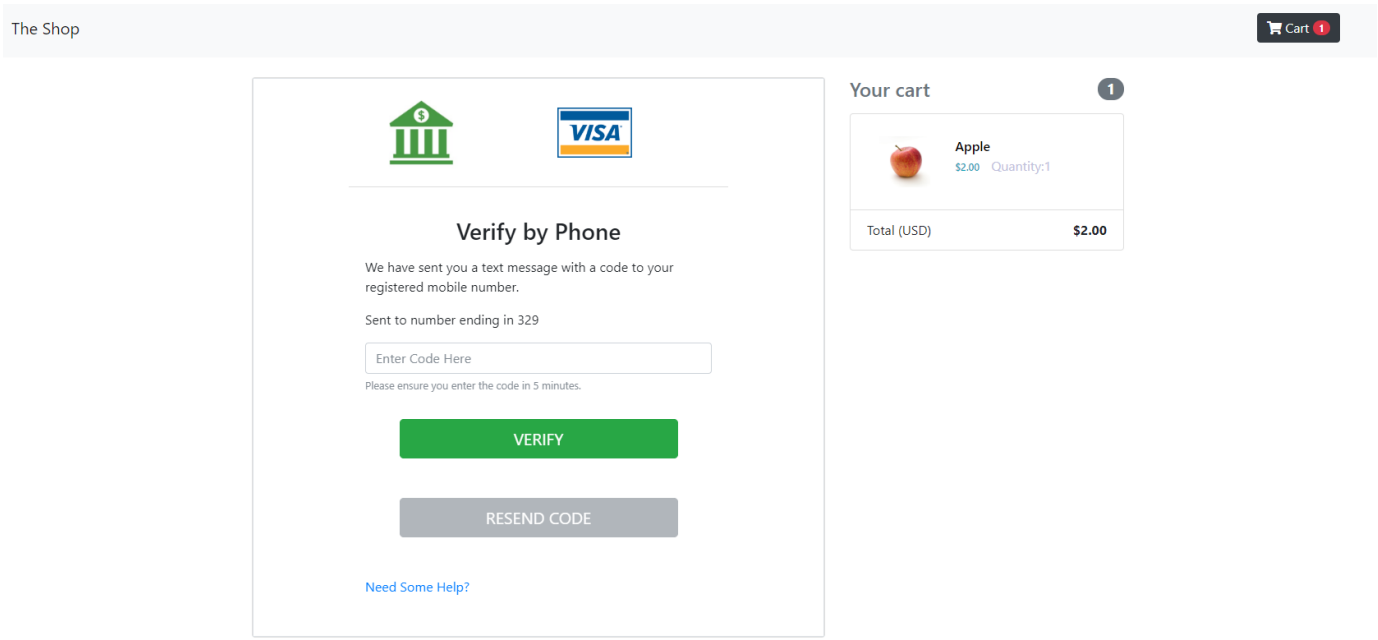
If the `transStatus` is `C`, the `3ds-web-adapter.js` will call `startChallenge()`, which inserts an `iframe` for the challenge window with `src` set to `challengeUrl` (Step. 14(C)).

```
    function startChallenge(url) {
    //remove the spinner
    $(".spinner").remove();

    //create the iframe
    $('<iframe src="'+url+'" class="h-100 w-100 border-0" id="challengeWindow"
name="challengeWindow"></iframe>')
     .appendTo('.challengeContainer');

    }
```

Note

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

You can see that `iframe` class has been set to `h-100` and `w-100`, which is the bootstrap default class to implement css style of `height: 100%!important` and `width: 100%!important` respectively. 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:



> **Info**
>
> In the real scenario the ACS will perform complex risk-based authentication from the information obtained about the cardholder, and is not as simple as checking the purchase amount. Similarly, the authentication method will be determined and implemented by the cardholders issuing bank.

Process 3: Get Authentication Result

To get authentication result, 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)).

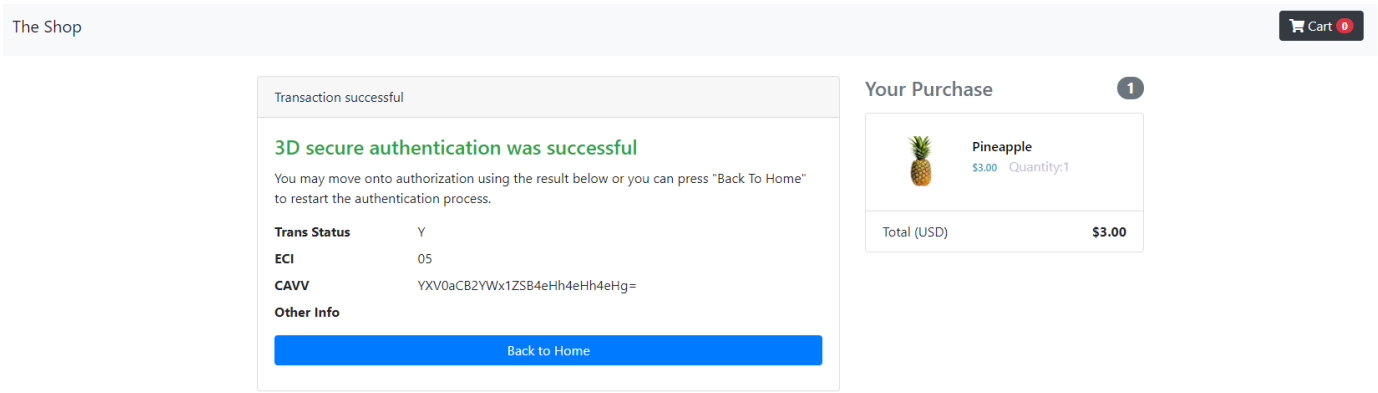
When the authentication result is ready, the **3DS Requestor** will notify the front-end using `_onAuthResult` message in `notify_3ds_events.html`. Then, **3ds-web-adapter** sends `Request for authentication result` request to **3DS Requestor** with url `/auth/result` in the following diagram. To check the **3DS Requestor** handles the `notify_3ds_events.html` message, please refer here.

```
function _onAuthResultReady(transId) {
    //redirect to result page
    window.location.href = "/auth/result?txid=" + transId;
```

Finally, the browser will redirect to `result.html` page to show the result details of the authentication.

```
 <div class="col-sm-8">
     <div class="card">
         <div class="card-header">Transaction successful</div>
         <div class="card-body">
             <h4 class="card-title text-success">3D secure authentication was
successful</h4>
             <p class="card-text">You may move onto authorization using the
result below
                  or you can press "Back To Home" to restart authentication
process.</p>

             <dl class="row">
                 <dt class="col-sm-3">Trans Status</dt>
                 <dd class="col-sm-9" data-th-text="${result.transStatus}">Y</
dd>
                 <dt class="col-sm-3">ECI</dt>
                 <dd class="col-sm-9" data-th-text="${result.eci}">eci value</
dd>
                 <dt class="col-sm-3">CAVV</dt>
                 <dd class="col-sm-9" data-th-text="$
{result.authenticationValue}">cavv value</dd>
                 <dt class="col-sm-3">Other Info</dt>
                 <dd class="col-sm-9"></dd>
             </dl>

             <a href="/" class="btn btn-primary">Back To Home</a>
         </div>
     </div>
 </div>
```

The new result screen will look like the screenshot below:



Success

Congratulations! You have done the front-end integration. After this process you normally move on to performing authorisation using the Transaction Status, ECI and CAVV to complete the transaction.

Whats next?

Please select the next button to learn about **Back-end implementation** for a 3DS Requestor.

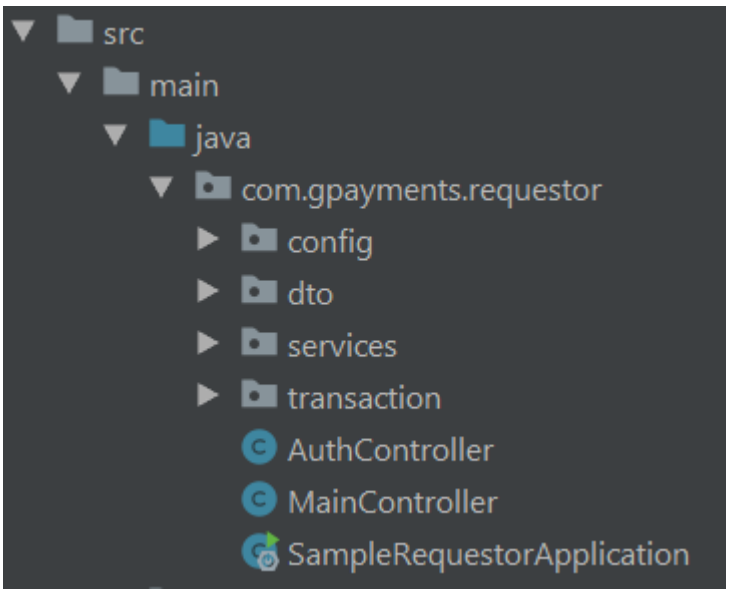
Back-end implementation

In this section, we will show the implementation details for the back-end side of the merchant application using our Sample Code.

> **Note**
>
> Please go through the **Sample Code** and implement that before continuing.

For the back-end, we need to implement a **3DS Requestor**. The following diagram shows the main files in the **3DS Requestor**, which can be found in the `/final` project of our Sample Code. Please check **directory tree** if needed.

The `AuthController` is our authentication controller which deals with requests from the **3DS-web-adapter** and forwards them to **ActiveServer**, such as Step. 2 and Step. 3.



Process 1: Initialise the Authentication

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

- Handle the `Initialise authentication` request from the **3DS-web-adapter** (Step. 2 and Step. 3).
- Receive the response message and return it to the **3DS-web-adapter** (Step. 4 and Step. 5).

In the `AuthController`, the **3DS Requestor** implements a `initAuth` handler method with `@PostMapping("/auth/init")` to handle the `Initialise authentication` request. To check the front-end is sending this request, please refer here.

```
public class AuthController {

  @PostMapping("/auth/init")
  public InitAuthResponseBRW initAuth(@RequestBody InitAuthRequestBRW request) {
    String initBrwUrl = THREE_DS_SERVER_URL + "/api/v1/auth/brw/init/pa";
    // Initialise authentication by making  POST request to /brw/init/
{messageCategory} (Step. 3)
    RequestEntity<InitAuthRequestBRW> req =
        new RequestEntity<>(request, HttpMethod.POST, URI.create(initBrwUrl));

    try {
      ResponseEntity<InitAuthResponseBRW> resp =
          restTemplate.exchange(req, InitAuthResponseBRW.class);

      InitAuthResponseBRW initRespBody = resp.getBody();
      logger.info("initAuthResponseBRW {}", initRespBody);

      // set InitAuthResponseBRW for future use
      transactionInfo.setInitAuthResponseBRW(initRespBody);
      return initRespBody;

    } catch (HttpClientErrorException | HttpServerErrorException ex) {

      logger.error("initAuthReq failed, {}, {}", ex.getStatusCode(),
ex.getResponseBodyAsString());

      throw ex;
    }
  }
}
```

The `initAuth` handler method firstly initiates the url `initBrwUrl = THREE_DS_SERVER_URL + ../brw/init/{messageCategory}` (**line 5**). The request will be posted to this url (Step. 3).

Note

- The String `THREE_DS_SERVER_URL` is the URL provided in the default configuration. `THREE_DS_SERVER_URL` is the `AUTH_API_URL` that can be found in `Settings > 3D Secure 2`, for more details refer to here
- `{messageCategory}` can be either `pa` or `npa`, which represents payment authentication or non-payment authentication respectively. In this example, we are using `pa`.

Then, `initAuth` controller posts the request and waits for the response in **line 11 and 12**. The data type for request and response are `InitAuthRequestBRW` and `InitAuthResponseBRW`, respectively.

- `InitAuthRequestBRW` - Holds all the data necessary to make the API call to `/brw/init/{messageCategory}` (Step. 3).
- `InitAuthResponseBRW` - Holds the information about the response from the API call to `/brw/init/{messageCategory}` (Step. 4).

The data structure of `InitAuthRequestBRW` and `InitAuthResponseBRW` can be found in the API Document.

Note

The following two tips are important for your implementation of the **3DS Requestor**.

Tip

The **3DS Requestor** connects **ActiveServer** via SSL. To make an API call we need to attach the **client certificate** to the RESTTemplate.

- Make sure the client certificate (`.p12` file) and the `cacerts.jks` truststore file inside the `/resources/certs` directory. Please refer to the Introduction to obtain the `.p12` file.
- The implementation of SSL configration is in `RestClientConfig` class. The class can be found in `requestor/config` directory.

Tip

In this sample code, the **3DS Requestor** fills the `InitAuthRequestBRW` request before sending it to the **3DS Server**.

```
//AuthController.java
public InitAuthResponseBRW initAuth(@RequestBody InitAuthRequestBRW request)
{
...
    fillInitAuthRequestBRW(request, THREE_DS_REQUESTOR_URL + "/3ds-notify");
...
}
```

Now take a look at the `fillInitAuthRequestBRW` method in the `AuthController.java`. This method fills the `InitAuthRequestBRW` with default data for the purpose of the demo. However, for production, you may replace this method to load data from your database or from cardholder information sent from the front-end.

- The String `THREE_DS_REQUESTOR_URL` is the URL provided in the default configuration, which is `http://localhost:8082`. You can update it with the URL of your hosted 3DS Requestor.
- We set the `eventCallBackUrl` to `THREE_DS_REQUESTOR_URL/3ds-notify`. This allows the **3DS Server** to make a notification when the browser information collection (Step. 7) is done.

Process 2: Execute Authentication

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

- Handle the `/3ds-notify` message from **ActiveServer** 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 `THREE_DS_REQUESTOR_URL/3ds-notify`. The **3DS Requestor** handles the notification in the `MainController.java` file.

```
// 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;
    // check the callbackType and initialise callbackName
    if ("3DSMethodFinished".equals(callbackType)) {

        callbackName = "_on3DSMethodFinished";

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

        callbackName = "_on3DSMethodSkipped";

    } else {
        throw new IllegalArgumentException("invalid callback type");
    }

    //Pass on the object to the page
    model.addAttribute("transId", transId);
    model.addAttribute("callbackName", callbackName);
    model.addAttribute("callbackParam", param);

    return "notify_3ds_events";
}
```

This handler method takes in the parameter `transId`, `callbackType` and an optional `param`. `callbackType` can be `3DSMethodFinished` or `3DSMethodSkipped`. The handler method returns a String `notify_3ds_events` which means it returns an HTML page with the matching named `notify_3ds_events.html`. To check the front-end implementation of `notify_3ds_events.html`, please refer here.

In the `AuthController`, the **3DS Requestor** implements an `auth` handler method with `@PostMapping("/auth")` to handle the `Execute authentication` request (Step. 9). This method makes a POST API request to **`/brw`** with `threeDSRequestorTransID` and `threeDSServerTransID` (Step. 10). **ActiveServer** will then initiate 3DS processing by creating and sending an `AReq` and processing the `ARes` when received in (Step. 11).

```
//AuthController.java
@PostMapping("/auth")
public AuthResponseBRW auth(@RequestParam("id") String transId) {

    MerchantTransaction transaction = transMgr.findTransaction(transId);

    //create authentication request.
    AuthRequestBRW authRequest = new AuthRequestBRW();
    authRequest.setThreeDSRequestorTransID(transaction.getId());

authRequest.setThreeDSServerTransID(transaction.getInitAuthResponseBRW().getThreeD

    String brwUrl = THREE_DS_SERVER_URL + "/api/v1/auth/brw";
    AuthResponseBRW response = restTemplate.postForObject(brwUrl, authRequest,
AuthResponseBRW.class);

    logger.info("authResponseBRW {}", response);

    return response;
}
```

The `auth` handler method returns the `response` to the front-end. The `response` contains a `transStatus` with value `Y` or `C`, which will trigger either frictionless flow or challenge flow. To check front-end handle `transStatus`, please refer to here. Moreover, to check the data structure of `AuthRequestBRW` and `AuthResponseBRW`, please refer to the API document.

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 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)).

When the authentication result is ready, **ActiveServer** will send a notification via `eventCallBackUrl`. The **3DS Requestor** handles the notification by returning an `_onAuthResult` message to `notify_3ds_events.html`. To check the front-end implementation for `_onAuthResult` message, please refer here.

```
//MainController.java
@PostMapping("/3ds-notify")
public String notifyResult(
    ...
    if ("3DSMethodFinished".equals(callbackType)) {
        callbackName = "_on3DSMethodFinished";
    } else if ("3DSMethodSkipped".equals(callbackType)) {
        callbackName = "_on3DSMethodSkipped";
    } else if ("AuthResultReady".equals(callbackType)) {
        callbackName = "_onAuthResult";
    }
    ...
```

3DS Requestor implements a `result` handler method with `@GetMapping("/auth/result")` to handle the `Request for authentication result` message. This handler method requests the authentication result by calling `/brw/result` to **ActiveServer** and returns the `result` to the front-end. The front-end will display the result using the `result.html` page. To check the implementation at the front-end, please refer here.

```
@GetMapping("/auth/result")
public String result(
        @RequestParam("txid") String transId,
        Model model) {

    MerchantTransaction transaction = transMgr.findTransaction(transId);
    String resultUrl = AuthController.THREE_DS_SERVER_URL + "/api/v1/auth/brw/
result?threeDSServerTransID=" +

transaction.getInitAuthResponseBRW().getThreeDSServerTransID();
    AuthResponseBRW response = restTemplate.getForObject(resultUrl,
AuthResponseBRW.class);
    model.addAttribute("result", response);
    return "result";
}
```

Success

Congratulations! You have done the back-end integration.

Whats next?

Please select the next button to go through the full **Step by step guide** for integrating a merchant website with a **3DS Requestor**.

Step-by-step guide

This section illustrates how to implement the authentication processes to integrate the 3DS2 flow into the simple shopping site project you downloaded in the introduction section. It can also be used as a guide to adding the same functionality to your payment process.

Intellij IDEA is used throughout the tutorial.

If you learn better by working with **finished code**, you can access the final merchant checkout page integrated with 3DS2 from the GPayments sample code package in the `/final` folder. Please refer to the final demo code section for an overview of the 3DS Requestor demo code.

> **Info**
>
> The purpose of this guide is to help you understand the general flow of implementing 3DS Requestor using **Java** so that you can implement it in your checkout page, which may or may not be written in a different programming language.

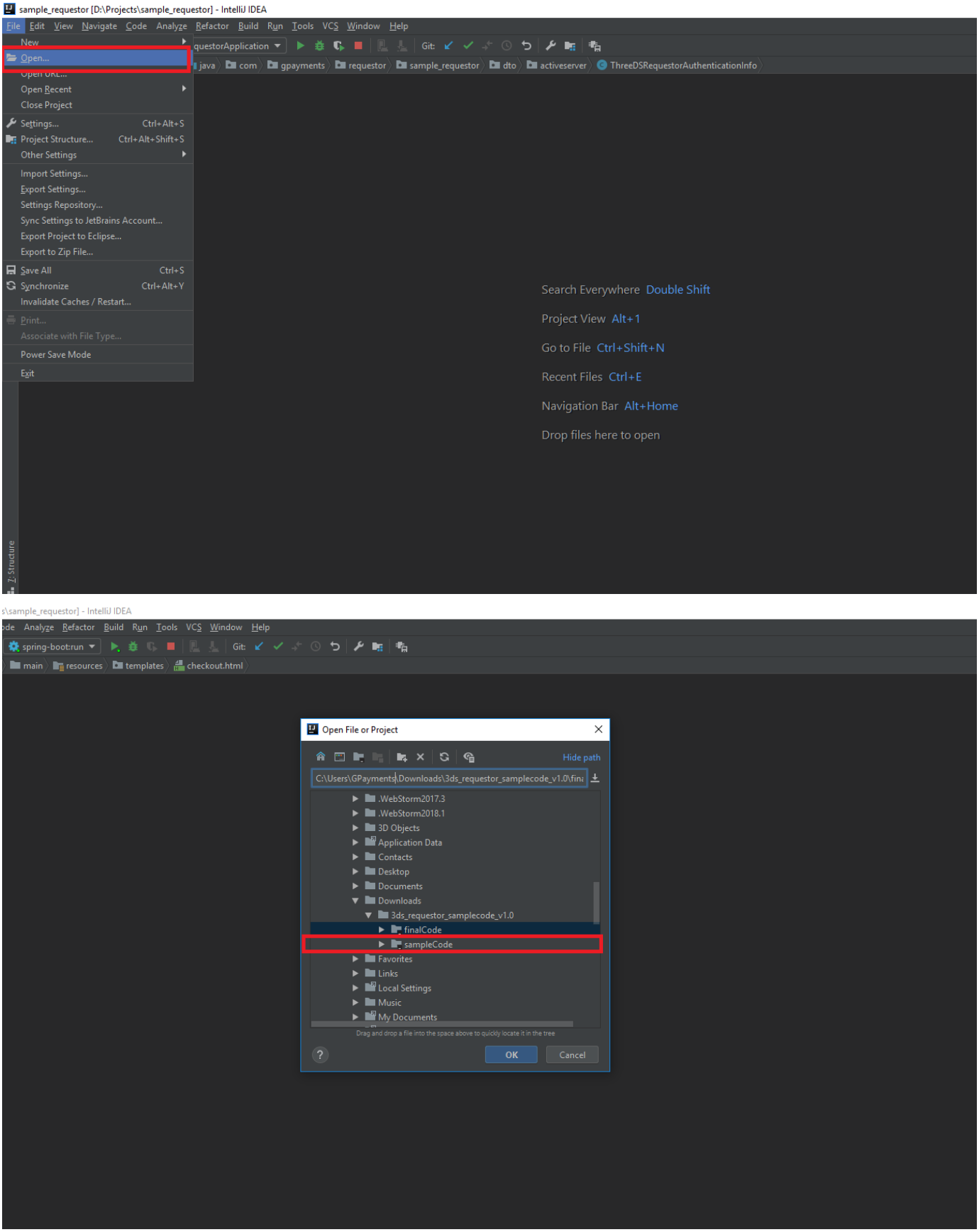
Step 1: Initialise the Authentication

We will first explain the process to initialise authentication, in other words, to call `/brw/init/{messageCategory}`.

`messageCategory` can be either `pa` or `npa`, which represents payment authentication or non-payment authentication respectively.

> **Tip**
>
> Each API endpoint links to its reference in the Authentication API document document, which can be accessed for further information on API usage.

The GPayments sample code package contains a folder called `/initial`. **Open** this directory with your IDE, **File > Open > Browse to initial folder > OK**.



In addition, you are provided with `3ds-web-adapter.js` to help you implement the process of authentication.

Copy and paste `3ds-web-adapter.js` from the `/final` project to the **/resources/static/js** directory in the initial project. Create a `js` directory if necessary. If you are not sure where this file is located, take a look at the final code directory tree.

Add the line highlighted below to import the `3ds-web-adapter.js` file into the `checkout.html` page.

```
....
<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/
popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/
l8WvCWPIPm49"
        crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/
bootstrap.min.js"
        integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/
JmZQ5stwEULTy"
        crossorigin="anonymous"></script>
<script src="/js/3ds-web-adapter.js"></script>
.....
```

Edit `checkout.html` to utilise the adapter. Take a look at the `checkout()` function in `checkout.html`, it should look like the code below. It is using JQuery to get the value of the cardholder input fields from the HTML document and initialising the new variables `transId`, `cardHolderInfo` and `purchaseInfo`:

```
//checkout.html
function checkout() {
    var transId = $('#transId').val();

    // NOTE: Some attributes are set to default values for demo purpose
    var cardHolderInfo = {};

    if($('#billAddrLine1').val()) {
        cardHolderInfo.billAddrLine1 = $('#billAddrLine1').val();
    }

    ....

    if($('#cardExpiry').val()) {
        purchaseInfo.expiryDate = $('#expiryDate').val();
    }
    //remove cardholder information class, checkout button and show spinner
effect
    $("#checkoutButton").remove();

    $("#cardholderInfoCard").remove();
    $("#checkoutCard").removeClass("d-none");

    //move to result screen 2 seconds after for demo purposes
    setTimeout(function () {
        window.location.href = "/auth/result"
    }, 2000);
}
```

Add the highlighted line inside the `checkout()` in `checkout.html` to call the method `threeDSAuth()` (Step.1), which is the start of the authentication process.

```
function checkout() {
    ...
    $("#cardholderInfoCard").remove();
    $("#checkoutCard").removeClass("d-none");

    //move to result screen 2 seconds after
    setTimeout(function () { window.location.href = "/auth/result"}, 2000);
    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

Comment out or delete the line highlighted below, this line was only there for demo purposes to add a two second delay before moving onto the result screen.

```
function checkout() {
    ...
    $("#cardholderInfoCard").remove();
    $("#checkoutCard").removeClass("d-none");

    //setTimeout(function () { window.location.href = "/auth/result"}, 2000);
    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

Note

The code below is the `threeDSAuth()` function defined in `3ds-web-adapter.js`. You can see that `threeDSAuth()` sends objects from the page in JSON format to `/auth/init` **line 10**, e.g. `threeDSRequestorTransID` is initialised by `transId`.

```
//3ds-web-adapter.js
function threeDSAuth(transId, cardHolderInfo, purchaseInfo) {

    var postData = {...}

    ....

    console.log('init authentication');
    $.ajax({
        url: '/auth/init',
        type: 'POST',
        contentType: "application/json",
        data: JSON.stringify(postData),
        success: function (data) {
            console.log('init auth returns:', data);
            $('<iframe id="threeds-container" width="0" height="0"
            style="visibility: hidden;"
            src="' + data.threeDSServerCallbackUrl + '"></iframe>')
                .appendTo('.challengeContainer');
        },
        error: function () {
            alert('error');
        },
        dataType: 'json'

    });

}
```

`threeDSAuth()` can be modified according to your needs and you can transfer your objects instead of `cardholderInfo`, `purchaseInfo` and `transId`. The types of data elements that can be sent are in `InitAuthRequestBRW`, which is inside the **/dto/activeserver** directory in the final code or alternatively you can check the API document.

`3ds-web-adapter.js` makes a POST request to `/auth/init` (Step.2), so a new controller class to handle this request is required.

You will need to create a controller to handle requests from the client side which handles the `/auth/init` request. **Create** a new controller class `AuthController.java` in the **/java/**

sample_requestor directory. **Copy and Paste** the `initAuth` method which handles the `/auth/init` request and calls the Authentication API endpoint `/api/v1/{messageCategory}`.

```
public class AuthController {
  @PostMapping("/auth/init")
  public InitAuthResponseBRW initAuth(@RequestBody InitAuthRequestBRW request) {
    String initBrwUrl = THREE_DS_SERVER_URL + "/api/v1/auth/brw/init/
{messageCategory}";
    // Initialise authentication by making  POST request to /brw/init/
{messageCategory} (Step. 3)
    RequestEntity<InitAuthRequestBRW> req =
        new RequestEntity<>(request, HttpMethod.POST, URI.create(initBrwUrl));

    try {
      ResponseEntity<InitAuthResponseBRW> resp =
          restTemplate.exchange(req, InitAuthResponseBRW.class);

      InitAuthResponseBRW initRespBody = resp.getBody();
      logger.info("initAuthResponseBRW {}", initRespBody);

      // set InitAuthResponseBRW for future use
      transactionInfo.setInitAuthResponseBRW(initRespBody);
      return initRespBody;

    } catch (HttpClientErrorException | HttpServerErrorException ex) {

      logger.error("initAuthReq failed, {}, {}", ex.getStatusCode(),
ex.getResponseBodyAsString());

      throw ex;
    }
  }
}
```

Replace `{messageCategory}` with `pa` if you want to initialise payment authentication, or `npa` if you want to initialise non-payment authentication. In this case, we are using `pa`.

```
//AuthController.java
String initBrwUrl = THREE_DS_SERVER_URL + "/api/v1/auth/brw/init/pa";
// Initialise authentication by making  POST request to /brw/init/
{messageCategory} (Step. 3)
InitAuthResponseBRW initAuthResponseBRW = restTemplate.postForObject(initBrwUrl,
    request, InitAuthResponseBRW.class);

....
```

Replace the String `THREE_DS_SERVER_URL` with the URL provided in the default configuration. `THREE_DS_SERVER_URL` is the `AUTH_API_URL` that can be found in `Settings > 3D Secure 2`, for more details refer to here

```
//AuthController.java
private final String THREE_DS_SERVER_URL = "https://api.as.testlab.
3dsecure.cloud:7443";
....
```

Replace the String `THREE_DS_REQUESTOR_URL` with the URL provided in the default configuration, or update it with the URL of your hosted 3DS Requestor.

```
//AuthController.java
private final String THREE_DS_REQUESTOR_URL = "http://localhost:8082";
....
```

Copy and paste the folder **`activeserver`** into the `/java/sample_requestor/dto` directory. This folder contains all the classes necessary to make the API calls. To initialise authentication, the following classes are the most important:

- `InitAuthRequestBRW` - Holds all the data necessary to make the API call to **`/brw/init/{messageCategory}`** (Step. 3).
- `InitAuthResponseBRW` - Holds the information about the response to the API call **`/brw/init/{messageCategory}`** (Step. 4).

Now take a look at the `fillInitAuthRequestBRW` method in the `AuthController.java`. This method fills the `InitAuthRequestBRW` with default data for the purpose of the demo. However, for production, you may replace this method to load data from your database or from cardholder information sent from `checkout.html`.

```
//AuthController.java
 /**
 * This method is to fill in the InitAuthRequestBRW with demo data, you need to
fill the information from your database
 * @param initAuthRequestBRW
 * @param eventCallBackUrl
 */
private void fillInitAuthRequestBRW(InitAuthRequestBRW initAuthRequestBRW,
String eventCallBackUrl) {

    initAuthRequestBRW.setAcctID("personal account");

    // Fill AcctInfo with default data.
    AcctInfo acctInfo = new AcctInfo();
    acctInfo.setChAccAgeInd("03");
    acctInfo.setChAccChange("20160712");
    acctInfo.setChAccChangeInd("04");
    acctInfo.setChAccDate("20140328");
    acctInfo.setChAccPwChange("20170328");
    acctInfo.setChAccPwChangeInd("02");
    acctInfo.setNbPurchaseAccount("11");
    acctInfo.setPaymentAccAge("20160917");
    acctInfo.setPaymentAccInd("02");
    acctInfo.setProvisionAttemptsDay("3");
    acctInfo.setShipAddressUsage("20160714");
    acctInfo.setShipAddressUsageInd("02");
    acctInfo.setShipNameIndicator("02");
    acctInfo.setSuspiciousAccActivity("02");
    acctInfo.setTxnActivityDay("1");
    acctInfo.setTxnActivityYear("21");
    initAuthRequestBRW.setAcctInfo(acctInfo);

    initAuthRequestBRW.setAcctType("03");
    initAuthRequestBRW.setAuthenticationInd("01");//01 = Payment transaction

    // fills ThreeDSRequestorAuthenticationInfo
    ThreeDSRequestorAuthenticationInfo threeDSRequestorAuthenticationInfo = new
ThreeDSRequestorAuthenticationInfo();
    threeDSRequestorAuthenticationInfo.setThreeDSReqAuthData("login GP");
    threeDSRequestorAuthenticationInfo.setThreeDSReqAuthMethod("02");

threeDSRequestorAuthenticationInfo.setThreeDSReqAuthTimestamp("201711071307");

initAuthRequestBRW.setAuthenticationInfo(threeDSRequestorAuthenticationInfo);

    // fills MerchantRiskIndicator, optional but strongly recommended for the
accuracy of risk based authentication
    MerchantRiskIndicator merchantRiskIndicator = new MerchantRiskIndicator();
    merchantRiskIndicator.setDeliveryEmailAddress("test@123.com");
    merchantRiskIndicator.setDeliveryTimeframe("02");
```

```
    merchantRiskIndicator.setGiftCardAmount("123");
    merchantRiskIndicator.setGiftCardCount("02");
    merchantRiskIndicator.setGiftCardCurr("840");
    merchantRiskIndicator.setPreOrderDate("20170519");
    merchantRiskIndicator.setPreOrderPurchaseInd("02");
    merchantRiskIndicator.setReorderItemsInd("01");
    merchantRiskIndicator.setShipIndicator("01");

    initAuthRequestBRW.setMerchantRiskIndicator(merchantRiskIndicator);

    /**
     * Options for threeDSRequestorChallengeInd - Indicates whether a challenge
is requested for this transaction.
     * Values accepted:
     *  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
     */
    initAuthRequestBRW.setChallengeInd("01");
    initAuthRequestBRW.setEventCallbackUrl(eventCallBackUrl); //Set this to
your url
    initAuthRequestBRW.setMerchantId("123456789012345");

initAuthRequestBRW.setPurchaseDate(LocalDateTime.now().format(DateTimeFormatter.of
    initAuthRequestBRW.setPurchaseInstalData("24");
    initAuthRequestBRW.setRecurringExpiry("20180131");
    initAuthRequestBRW.setRecurringFrequency("6");
    initAuthRequestBRW.setTransType("03");
}
```

Add a new variable `initAuthResponseBRW` to `MerchantTransaction.java` and corresponding `getter` and `setter` methods.

```
//MerchantTransaction.java
import com.gpayments.requestor.requestor.dto.activeserver.InitAuthResponseBRW;

public class MerchantTransaction {

    private InitAuthResponseBRW initAuthResponseBRW;

    ...

    //getter
    public InitAuthResponseBRW getInitAuthResponseBRW() {
        return initAuthResponseBRW;
    }

    //setter
    public void setInitAuthResponseBRW(InitAuthResponseBRW initAuthResponseBRW)
{
        this.initAuthResponseBRW = initAuthResponseBRW;
    }
```

To make an API call to ActiveServer we will need to attach the client certificate to the RESTTemplate.

Copy and paste the client certificate (.p12 file) to the **/resources/certs** directory. Create the `/certs` directory if not present. See this section for more information on getting the client certificate.

Copy and paste the `cacerts.jks` truststore file inside the final code to **/resources/certs** directory. A certificate is required because your 3DS Requestor and the 3DS Server need to be mutually authenticated to allow mutual SSL authentication.

Copy and Paste a configuration class `RestClientConfig.java` from the `final` package to the **/java/sample_requestor/config** directory. This class is an example of how to load the client certificate to a HttpClient. It loads the keystore file you have downloaded from ActiveServer with the truststore provided by GPayments.

```
//RestClientConfig.java
@Configuration
public class RestClientConfig {

  private static final String KEYSTORE_PASSWORD = "123456";
  private static final String CA_CERTS_FILE_NAME = "certs/cacerts.jks";
  private static final String CLIENT_CERTS_FILE_NAME = "certs/
client_certificate.p12";

  @Bean
  public RestTemplate restTemplate()
      throws IOException, UnrecoverableKeyException, CertificateException,
NoSuchAlgorithmException,
      KeyStoreException, KeyManagementException {
    SSLContext sslContext =
        SSLContextBuilder.create()
            .loadKeyMaterial(
                new ClassPathResource(CLIENT_CERTS_FILE_NAME).getURL(),
                KEYSTORE_PASSWORD.toCharArray(),
                KEYSTORE_PASSWORD.toCharArray())
            .loadTrustMaterial(
                new ClassPathResource(CA_CERTS_FILE_NAME).getURL(),
KEYSTORE_PASSWORD.toCharArray())
            .build();

    CloseableHttpClient client =
        HttpClients.custom()
            .setSSLContext(sslContext)
            .build();
    HttpComponentsClientHttpRequestFactory httpRequestFactory =
        new HttpComponentsClientHttpRequestFactory(client);

    return new RestTemplate(httpRequestFactory);
  }
}
```

Replace `KEYSTORE_PASSWORD` with the password provided in the default configuration. If your ActiveServer is a in-house software, this will be the password you have provided when downloading the client certificate from the merchants page in Admin dashboard. For more details on downloading the client certificate please refer to Merchant Security.

```
//RestClientConfig.java
private String KEYSTORE_PASSWORD = "123456";
```

> **Info**
>
> This is a `Java` specific way to make a RESTFul API request with a client certificate. You will need to take similar steps if you are using a different language to implement the server-side code so that your 3DS requestor is mutually authenticated with the 3DS server.

Add the following dependency to `pom.xml` to get Apache HttpClient, which is required for the `SSLContextBuilder` class.

```
<dependencies>
    ...
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.6</version>
    </dependency>
</dependencies>
```

Make sure you have imported all the classes required and there are no errors.

Run the app again then access from http://localhost:8082 and go through the checkout process. You should get a successful response for `/api/v1/auth/brw/init/{messageCategory}` with a valid `threeDSServerCallbackUrl` in the log, which should look similar to this:

```
{
    "threeDSServerCallbackUrl":"https://admin.as.testlab.3dsecure.cloud/api/v1/
auth/brw/callback?transId=cbc559c0-96bd-4078-be3c-
ea0cc80686f0&t=ZIIgV2LqMak5ONQ1w3l3akfpZlMPftFlOE5Garat6lHiJ3T2Kq2vULuLhIQ7l5UKzpn
    "threeDSServerTransID":"ade87add-b50c-42da-bbea-cdcbd85dcbf3"
}
```

The checkout process will stop at the `3ds-notify` stage because this has not been implemented yet, we will handle this in the next section.

Tip

You need to rerun the project for the changes to be propagated. Terminate the currently running app using `CTRL+C` and rerun the project by typing in the command line:

```
mvn spring-boot:run
```

The 3DS server and ACS collects the necessary browser information for risk-based authentication through the callback URL and 3DS method respectively.

Note

After a successful response from `/auth/init` (Step. 5) the `3ds-web-adapter.js` inserts a hidden `iframe` into the checkout page (Step. 6) so that the browser information can be collected by the ACS and 3DS Server using the `threeDSServerCallbackUrl` (Step. 7).

```
//3ds-web-adapter.js
success: function(data) {
    $('<iframe id="threeds-container" width="0" height="0"
style="visibility: hidden;" src="'+ data.threeDSServerCallbackUrl+'"></
iframe>')
    .appendTo('.challengeContainer');
},
```

Info

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

Process 2: Execute authentication

In the previous step, `3ds-web-adapter.js` inserted an `iframe` with callback `src` set to `threeDSServerCallbackUrl`, this allows the 3DS Server to make callbacks to the browser (Step. 7) to collect necessary browser information.

Additionally, we called `/brw/init/{messageCategory}` with `eventCallBackUrl` set to `THREE_DS_REQUESTOR_URL/3ds-notify` **(line. 2)**. This allows the 3DS server to make a notification when the browser information collection is done.

```
//AuthController.java
fillInitAuthRequestBRW(request, THREE_DS_REQUESTOR_URL + "/3ds-notify");
.....
```

Therefore, we need to make a handler method to accept this API call.

Add the following code to `MainController.java`. This handler method takes in the parameter `transId`, `callbackType` and an optional `param`. `callbackType` can be `3DSMethodFinished` or `3DSMethodSkipped`. The handler method returns a String `notify_3ds_events`. To explain, if you are not familiar with Springboot, this means it returns an HTML page with the matching name from the **/resources/templates** directory.

```
// 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;
    // check the callbackType and initialise callbackName
    if ("3DSMethodFinished".equals(callbackType)) {

        callbackName = "_on3DSMethodFinished";

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

        callbackName = "_on3DSMethodSkipped";

    } else {
        throw new IllegalArgumentException("invalid callback type");
    }

    //Pass on the object to the page
    model.addAttribute("transId", transId);
    model.addAttribute("callbackName", callbackName);
    model.addAttribute("callbackParam", param);

    return "notify_3ds_events";
}
```

Create a new HTML file called `notify_3ds_events.html` in the **/resources/templates** directory to call `callbackFn`, which is the page returned by a call to `/3ds-notify`.

```
<!--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" data-th-
value="${callbackName}"/>
    <input type="hidden" id="transId" name="transId" data-th-value="${transId}"/
>
    <input type="hidden" id="param" name="param" data-th-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 checkout 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` provided by the 3DS Server, it will call different methods in `3ds-web-adapter.js` (Step. 8). An explanation of each method is below:

- `_onThreeDSMethodFinished` - notifies that the 3DS method is finished and calls `_doAuth()`, which means browser information collection is finished.

- `_onThreeDSMethodSkipped` - notifies that the 3DS method has been skipped and calls `_doAuth()`, which means browser information collection by the ACS is skipped.

The `_doAuth()` in `3ds-web-adapter` will make a POST request to `/auth` to execute authentication (Step. 9).

Add a new handler method in `AuthController.java` to handle this request. This method should make a POST API request to **`/brw`** with `threeDSRequestorTransID` and `threeDSServerTransID` (Step. 10). The 3DS Server will then initiate 3DS processing by creating and sending an AReq and processing the ARes when received in (Step. 11).

```
//AuthController.java
@PostMapping("/auth")
public AuthResponseBRW auth(@RequestParam("id") String transId) {

    MerchantTransaction transaction = transMgr.findTransaction(transId);

    //create authentication request.
    AuthRequestBRW authRequest = new AuthRequestBRW();
    authRequest.setThreeDSRequestorTransID(transaction.getId());

authRequest.setThreeDSServerTransID(transaction.getInitAuthResponseBRW().getThreeD

    String brwUrl = THREE_DS_SERVER_URL + "/api/v1/auth/brw";
    AuthResponseBRW response = restTemplate.postForObject(brwUrl, authRequest,
AuthResponseBRW.class);

    logger.info("authResponseBRW {}", response);

    return response;
}
```

Rerun your app and go through the checkout process with default values, you should get a successful response for `/api/v1/auth/brw` (Step. 12), which will have transStatus set to `Y`.

```
{
    "authenticationValue":"YXV0aCB2YWx1ZSB4eHh4eHh4eHg=",
    "eci":"06",
    "threeDSServerTransID":"52056514-7504-40c7-886f-2a0452d8edbd",
    "transStatus":"Y"
}
```

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

The response of API call to `/brw` will have a `challengeUrl` set and a `transStatus` set to `C`.


```
{
    "acsChallengeMandated":"Y",
    "authenticationType":"01",
    "challengeUrl":"https://admin.as.testlab.3dsecure.cloud/api/v1/auth/brw/
challenge/init?txid=78cd5068-96ea-48c4-b013-e6843fa8b2e4",
    "threeDSServerTransID":"c8a32fb7-9556-4242-896b-562ee8ca25df",
    "transStatus":"C",
}
```

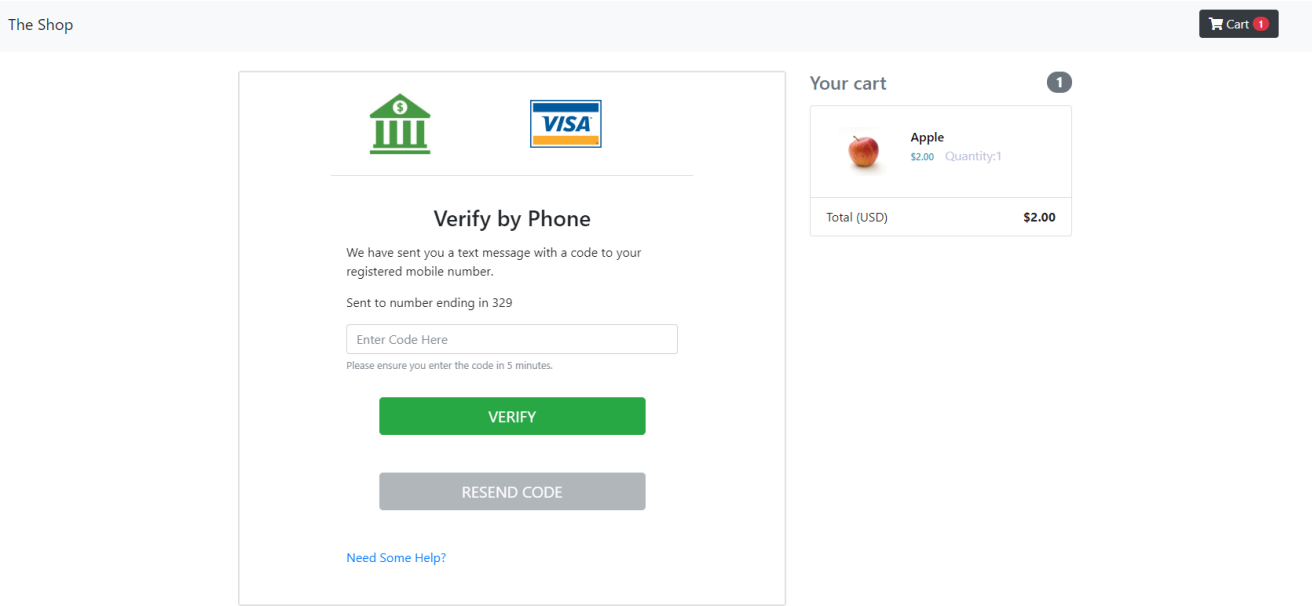

The one time pass code requested in the challenge window is **123456**. However, you will get an error because we have not yet implemented handling the `callbackType` `AuthResultReady`, which we will implement in the next step.

Note

If the `transStatus` is `C`, the `3ds-web-adapter.js` will call `startChallenge(url)`, which is going to insert an `iframe` for the challenge window with `src` set to `challengeUrl` (Step. 14(C)).

```
function startChallenge(url) {
//remove the spinner
$(".spinner").remove();

//create the iframe
$('<iframe src="'+url+'" class="h-100 w-100 border-0" id="challengeWindow"
name="challengeWindow"></iframe>')
.appendTo('.challengeContainer');

}
```

You can see that `iframe` class has been set to `h-100` and `w-100`, which is the bootstrap default class to implement css style of `height: 100%!important` and `width: 100%!important` respectively. 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:



Info

In the real scenario the ACS will perform complex risk-based authentication from the information obtained about the cardholder, and is not as simple as checking the purchase amount. Similarly, the authentication method will be determined and implemented by the cardholders issuing bank.

Process 3: Get Authentication result updates

As mentioned in the previous step, we now need to request for a result update of the authentication so that it can be displayed to the cardholder, this is also required in the frictionless flow (see the warning section below).

> ⚠ **Why a separate result request is necessary?**
>
> For the Frictionless flow, you may wonder why you need to request the result again since you already have the result of authentication available in Step. 12.
>
> 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 3DS Requestor should always have its own mechanism to get the result back from the origin source: the 3DS Server. 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.

For the challenge flow, the result of the authentication will be notified by the 3DS server through the `3ds-notify` endpoint, similar to how it notified the status of 3DS method.

Add a new `else if` statement, highlighted below, to handle "AuthResultReady" `callbackType` inside `MainController.java`.

```
//MainController.java
if ("3DSMethodFinished".equals(callbackType)) {
    callbackName = "_on3DSMethodFinished";
} else if ("3DSMethodSkipped".equals(callbackType)) {
    callbackName = "_on3DSMethodSkipped";
} else if ("AuthResultReady".equals(callbackType)) {
    callbackName = "_onAuthResult";
}
```

`_onAuthResultReady` is called from `notify-3ds-events`. This will redirect the window to `/auth/result?txid=+transId`. If you would like to get the result of the authentication, you can request the resulting receipt inside the handler `/auth/result` by calling **`/brw/result`**.

```
@GetMapping("/auth/result")
public String result(
        @RequestParam("txid") String transId,
        Model model) {

    MerchantTransaction transaction = transMgr.findTransaction(transId);
    String resultUrl = AuthController.THREE_DS_SERVER_URL + "/api/v1/auth/brw/
result?threeDSServerTransID=" +

transaction.getInitAuthResponseBRW().getThreeDSServerTransID();
    AuthResponseBRW response = restTemplate.getForObject(resultUrl,
AuthResponseBRW.class);
    model.addAttribute("result", response);
    return "result";
}
```

Delete or comment out the highlighted code in `result.html`.

```
<div class="col-sm-8">
    <div class="card">
        <div class="card-header">Transaction successful</div>
        <div class="card-body">
            <!--<h4 class="card-title text-success">Thank you for your purchase!
</h4>-->
            <!--<p class="card-text">Want to continue shopping? Press "Back to
Home".</p>-->

            <a href="/" class="btn btn-primary btn-block">Back to Home</a>

        </div>
    </div>
</div>
```

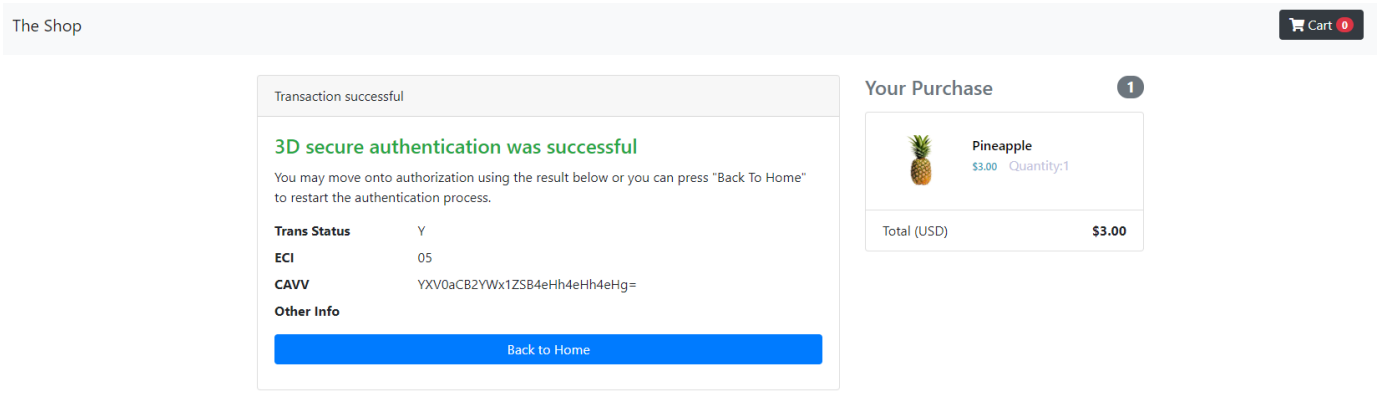
Add the following highlighted code instead to `result.html`, to show the result of the authentication and more description.

```
 <div class="col-sm-8">
     <div class="card">
         <div class="card-header">Transaction successful</div>
         <div class="card-body">
             <h4 class="card-title text-success">3D secure authentication was
successful</h4>
             <p class="card-text">You may move onto authorization using the
result below
                  or you can press "Back To Home" to restart authentication
process.</p>

             <dl class="row">
                 <dt class="col-sm-3">Trans Status</dt>
                 <dd class="col-sm-9" data-th-text="${result.transStatus}">Y</
dd>
                 <dt class="col-sm-3">ECI</dt>
                 <dd class="col-sm-9" data-th-text="${result.eci}">eci value</
dd>
                 <dt class="col-sm-3">CAVV</dt>
                 <dd class="col-sm-9" data-th-text="$
{result.authenticationValue}">cavv value</dd>
                 <dt class="col-sm-3">Other Info</dt>
                 <dd class="col-sm-9"></dd>
             </dl>

             <a href="/" class="btn btn-primary">Back To Home</a>
         </div>
     </div>
 </div>
```

Success

Congratulations! You now have a working merchant checkout page integrated with ActiveServer. Try running the app again and you should see a successful result page with Trans Status, ECI and CAVV.

The new result screen will look like the screenshot below:



> **Tip**
>
> After this process you normally move on to performing authorisation using the Transaction Status, ECI and CAVV to complete the transaction.

Final demo code

The final demo code can be accessed from the GPayments sample code package in the `/final` folder. It will work out-of-box with ActiveServer using the default configuration. Simply run the app by following this guide.

Summary of final 3DS Requestor demo code

3DS Requestor demo code implements several endpoints and components, as described below:

- `/auth/init` - Called by the `3ds-web-adapter.js`, this endpoint accepts authentication initialisation request to trigger the 3DS2 authentication process. In the 3DS Requestor, it calls ActiveServer's authentication initialisation api **`/api/v1/auth/brw/init/{messageCategory}`**.
- `/auth` - Called by the `3ds-web-adapter.js`, this endpoint accepts authentication request to execute 3DS2 authentication after browser information collecting, and optional 3DS method execution. In the 3DS Requestor, it calls ActiveServer's authentication api **`/api/v1/auth/brw`**.
- `/3ds-notify` - Allows ActiveServer to notify the 3DS Requestor about status of 3DS method execution , authentication and challenge results.

- `3ds-web-adapter.js` - Simple javascript library created for 3DS Requestor process scaffolding.
- Classes under package `com.gpayments.requestor.dto.activeserver` - ActiveServer API DTO (Data Transfer Object).

When integrated with merchant website, 3DS Requestor demo code provides the `3ds-web-adapter` for the merchant site to integrate it in the checkout page. Merchant website fills the cardholder information by calling `threeDSAuth()` in `3ds-web-adapter.js` to start the authentication process. Merchant website will also need to provide an `iframe` for the 3DS Requestor to populate associate callback url and 3DS method to execute.

Authentication result will be provided by the 3DS Requestor at the end of the authentication process through the event callback url `/3ds-notify`. Payment gateway or the merchant can proceed to the authorization process from here.

Lightbox challenge window

In the tutorial, the challenge screen was inline, however, depending on your website design, you may want to provide the challenge window inside a popup. The UI itself is provided by the ACS so you don't need to worry about implementing that.

Open `checkout.html`, and find the following section of code, where it defines the challenge window.

```
 <!--checkout.html-->
<div class="card d-none" id="checkoutCard">
    <div class="challengeContainer border">
        <div class="spinner row h-100 justify-content-center align-items-
center">
            <div class="col">
                <div class="sk-fading-circle">
                    <div class="sk-circle1 sk-circle"></div>
                    <div class="sk-circle2 sk-circle"></div>
                    <div class="sk-circle3 sk-circle"></div>
                    <div class="sk-circle4 sk-circle"></div>
                    <div class="sk-circle5 sk-circle"></div>
                    <div class="sk-circle6 sk-circle"></div>
                    <div class="sk-circle7 sk-circle"></div>
                    <div class="sk-circle8 sk-circle"></div>
                    <div class="sk-circle9 sk-circle"></div>
                    <div class="sk-circle10 sk-circle"></div>
                    <div class="sk-circle11 sk-circle"></div>
                    <div class="sk-circle12 sk-circle"></div>
                </div>
                <div class="text-center"><img class="w-25" src="images/visa-
logo.png"/></div>
            </div>
        </div>
    </div>
</div>
```

Comment out line 2 and 3 to disable the inline challenge card.

```
 <!--checkout.html-->
<!--<div class="card d-none" id="checkoutCard">-->
    <!--<div class="challengeContainer border">-->
        <div class="spinner row h-100 justify-content-center align-items-
center">
            <div class="col">
                <div class="sk-fading-circle">
                    <div class="sk-circle1 sk-circle"></div>
                    <div class="sk-circle2 sk-circle"></div>
                    <div class="sk-circle3 sk-circle"></div>
                    <div class="sk-circle4 sk-circle"></div>
                    <div class="sk-circle5 sk-circle"></div>
                    <div class="sk-circle6 sk-circle"></div>
                    <div class="sk-circle7 sk-circle"></div>
                    <div class="sk-circle8 sk-circle"></div>
                    <div class="sk-circle9 sk-circle"></div>
                    <div class="sk-circle10 sk-circle"></div>
                    <div class="sk-circle11 sk-circle"></div>
                    <div class="sk-circle12 sk-circle"></div>
                </div>
                <div class="text-center"><img class="w-25" src="images/visa-
logo.png"/></div>
            </div>
        </div>
    </div>
</div>
<!--Cardholder information -->
<div class="card" id="cardholderInfoCard">
```

Add lines 4, 5, 6, 7, 8, 30 and 31 to place the challenge window inside a modal.

```
<!--checkout.html-->
<!--<div class="card d-none" id="checkoutCard">-->
<!--<div class="challengeContainer border">-->
<div class="modal fade" id="authBox" data-backdrop="static" data-
keyboard="false" tabindex="-1"
             role="dialog" aria-labelledby="exampleModalLabel" aria-
hidden="true">
    <div class="modal-dialog h-100 d-flex flex-column justify-content-center
my-0" role="document">
        <div class="modal-content">
            <div class="modal-body challengeContainer">
                <div class="spinner row h-100 justify-content-center align-
items-center">
                    <div class="col">
                        <div class="sk-fading-circle">
                            <div class="sk-circle1 sk-circle"></div>
                            <div class="sk-circle2 sk-circle"></div>
                            <div class="sk-circle3 sk-circle"></div>
                            <div class="sk-circle4 sk-circle"></div>
                            <div class="sk-circle5 sk-circle"></div>
                            <div class="sk-circle6 sk-circle"></div>
                            <div class="sk-circle7 sk-circle"></div>
                            <div class="sk-circle8 sk-circle"></div>
                            <div class="sk-circle9 sk-circle"></div>
                            <div class="sk-circle10 sk-circle"></div>
                            <div class="sk-circle11 sk-circle"></div>
                            <div class="sk-circle12 sk-circle"></div>
                        </div>
                        <div class="text-center"><img class="w-25" src="images/
visa-logo.png"/></div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
<!--Cardholder information -->
<div class="card" id="cardholderInfoCard">
```

Now scroll down to the end of `checkout.html` to find the following code inside the `checkout()` function.

```
//checkout.html
function checkout() {
    ....


    //remove cardholder information class, checkout button and show spinner
effect
    $("#checkoutButton").remove();

    $("#cardholderInfoCard").remove();
    $("#checkoutCard").removeClass("d-none");

    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

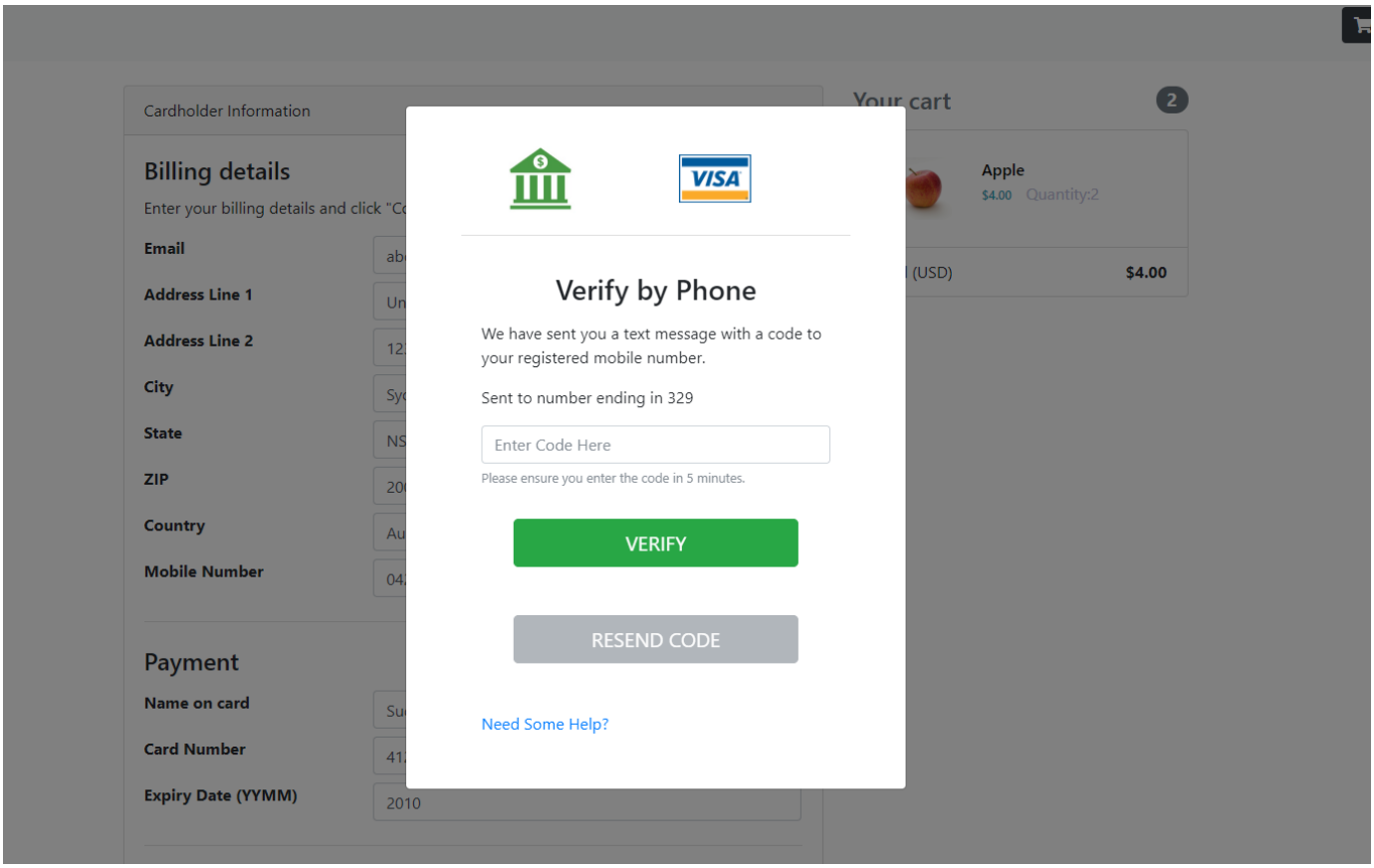
Comment out line 8 and 9, as this is to remove the cardholder information class and show the spinner effect in the card.

```
//checkout.html
function checkout() {
    ....

    //remove cardholder information class, checkout button and show spinner
effect
    $("#checkoutButton").remove();

    //$("#cardholderInfoCard").remove();
    //$("#checkoutCard").removeClass("d-none");

    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

Add line 11 to show the modal box for the popup challenge window instead.

```
//checkout.html
function checkout() {
    ....

    //remove cardholder information class, checkout button and show spinner
effect
    $("#checkoutButton").remove();

    //$("#cardholderInfoCard").remove();
    //$("#checkoutCard").removeClass("d-none");

    $('#authBox').modal();

    threeDSAuth(transId, cardHolderInfo, purchaseInfo);
}
```

Re-run your code you can see that the challenge screen has switched to a popup window.



Sample project default configuration

The following values are default values that are set in the final demo code. You may refer to this section during the step-by-step tutorial.

- THREE_DS_SERVER_URL=https://api.as.testlab.3dsecure.cloud:7443
- THREE_DS_REQUESTOR_URL=http://localhost:8082
- KEYSTORE_PASSWORD=123456

> **Note**
>
> - `THREE_DS_SERVER_URL` is the `AUTH_API_URL` that can be found in `Settings > 3D Secure 2`, for more details refer to here
> - `KEYSTORE_PASSWORD` is the password set when downloading the certificate file (.p12 file) from ActiveServer Administration. For downloading the certificate file, please refer to here.

Support

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

Directory tree

Below is the directory tree of the final merchant checkout site, the sampleCode will look like the below directory at the end of the tutorial.

```
// Final demo code directory tree
|- com.gpayments.requestor
|   |- config
|   |   |- RestClientConfig.java
|   |   |- JacksonConfig.java
|   |- dto
|   |   |- activeserver
|   |   |    |- AcctInfo.java
|   |   |    |- AuthRequestBRW.java
|   |   |    |- AuthResponseBRW.java
|   |   |    |- CardholderInformation.java
|   |   |    |- InitAuthRequestBRW.java
|   |   |    |- InitAuthResponseBRW.java
|   |   |    |- MerchantRiskIndicator.java
|   |   |    |- PhoneNumber.java
|   |   |    |- ThreeDSRequestorAuthenticationInfo.java
|   |   |
|   |   |- CardholderInfo.java
|   |   |- MyCart.java
|   |   |- Item.java
|   |
|   |- services
|   |   |-CardHolderService.java
|   |   |-CartService.java
|   |   |-ShopService.java
|   |
|   |- transaction
|   |     |- MerchantTransaction.java
|   |     |- TransactionManager.java
|   |
|   |- AuthController.java
|   |- MainController.java
|   |- SampleRequestorApplication.java
|
|- resources
    |-static
    |   |-css
    |   |   |-spinner.css
    |   |   |-style.css
    |   |
    |   |-images
    |   |   |-apple.jpg
    |   |   |-banana.jpg
    |   |   |-pineapple.jpg
    |   |   |-visa-logo.jpg
    |   |
    |   |-js
    |   |   |-3ds-web-adapter.js
    |
    | -certs
```

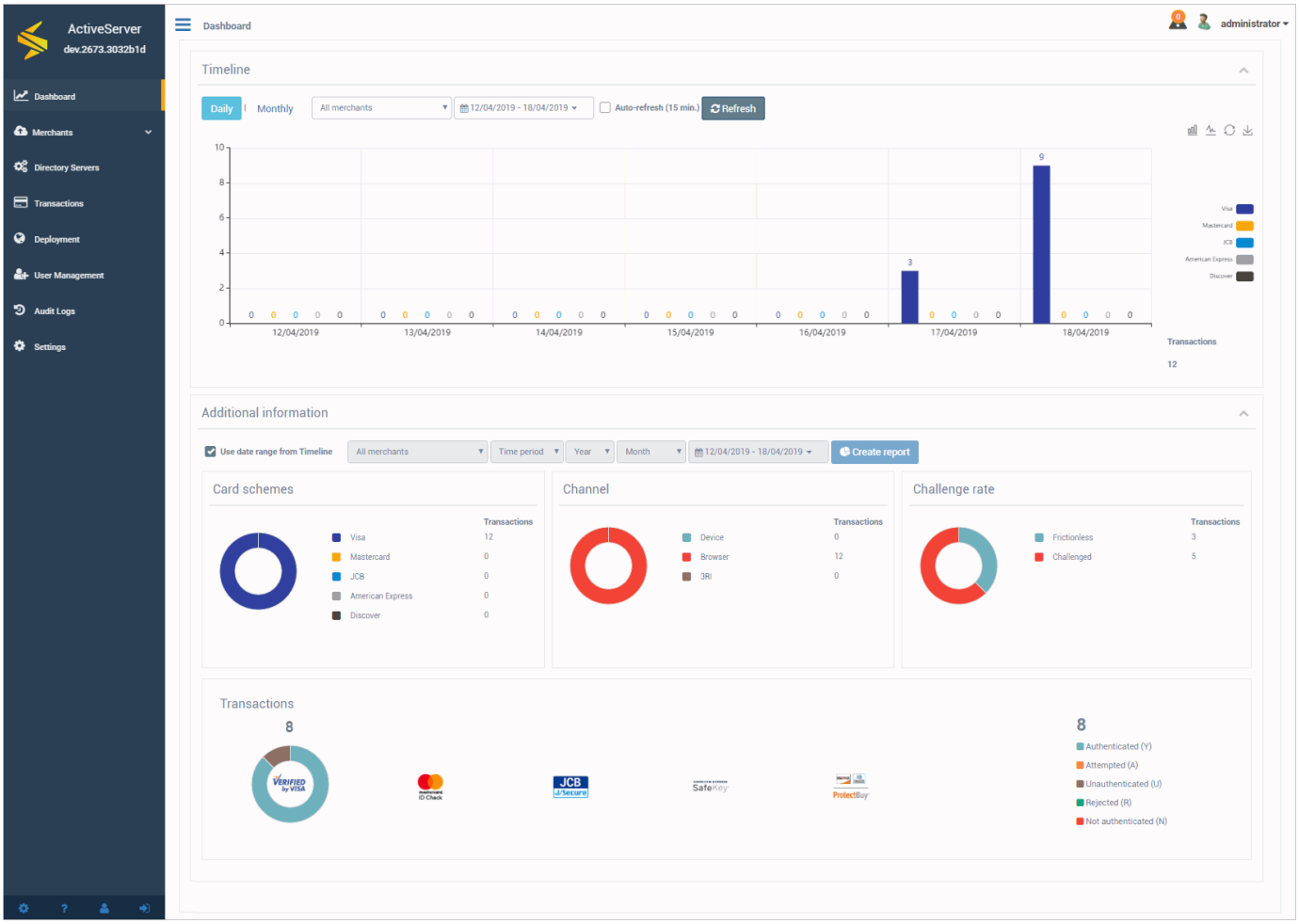
```
|   |- client_certificate.p12
|   |- cacerts.jks
|
|-templates
|   |- checkout.html
|   |- index.html
|   |- notify_3ds_events.html
|   |- result.html
|   |- error.html
|
|-application.properties
```

Using the dashboard

The **Dashboard** has two sections:

- Timeline
- Additional information

> **Tip**
>
> Each dashboard section can be collapsed/expanded by clicking the section header.



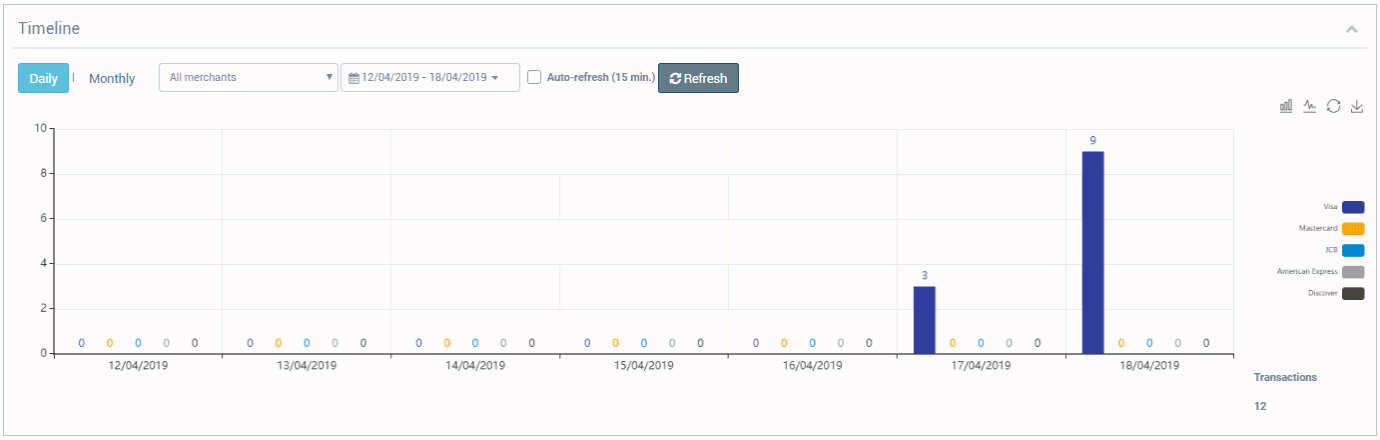
Timeline

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

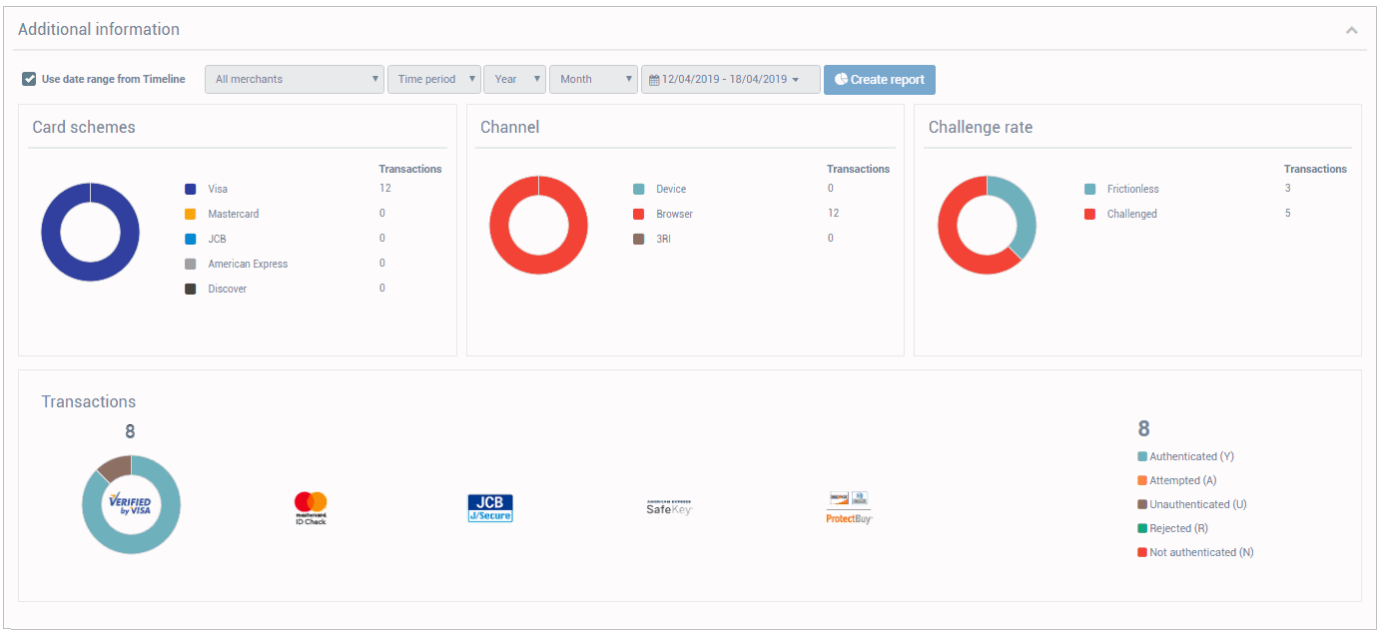
If **Use date range from Timeline** is selected, date range and merchant selection options will be greyed out and the options chosen in the **Timeline** section will be used to show a synchronised view across both sections.

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.



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.

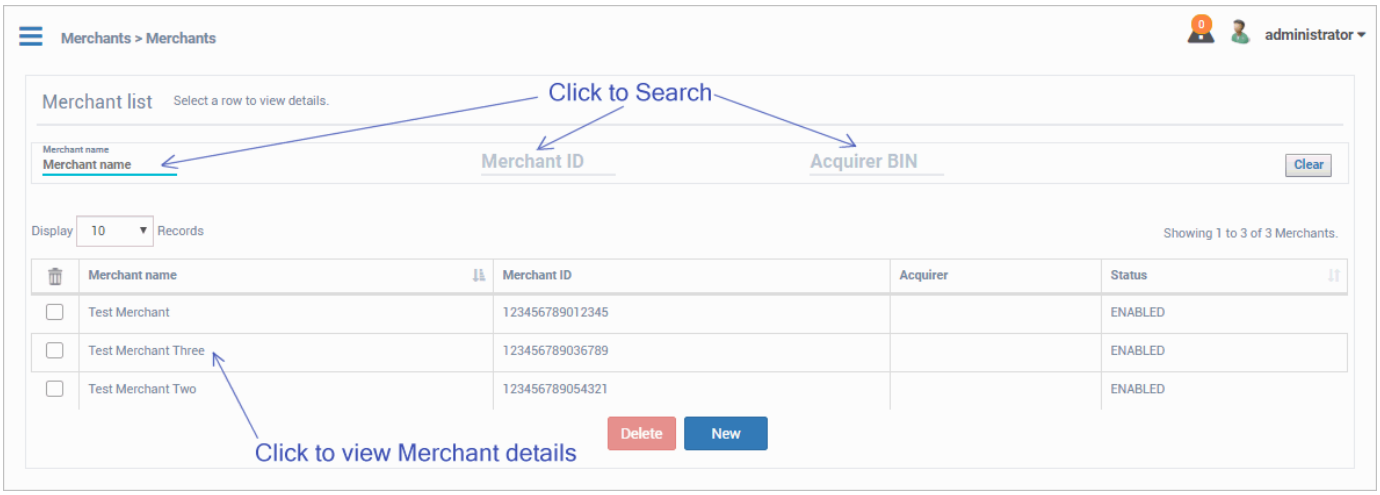
Transactions

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.



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.

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

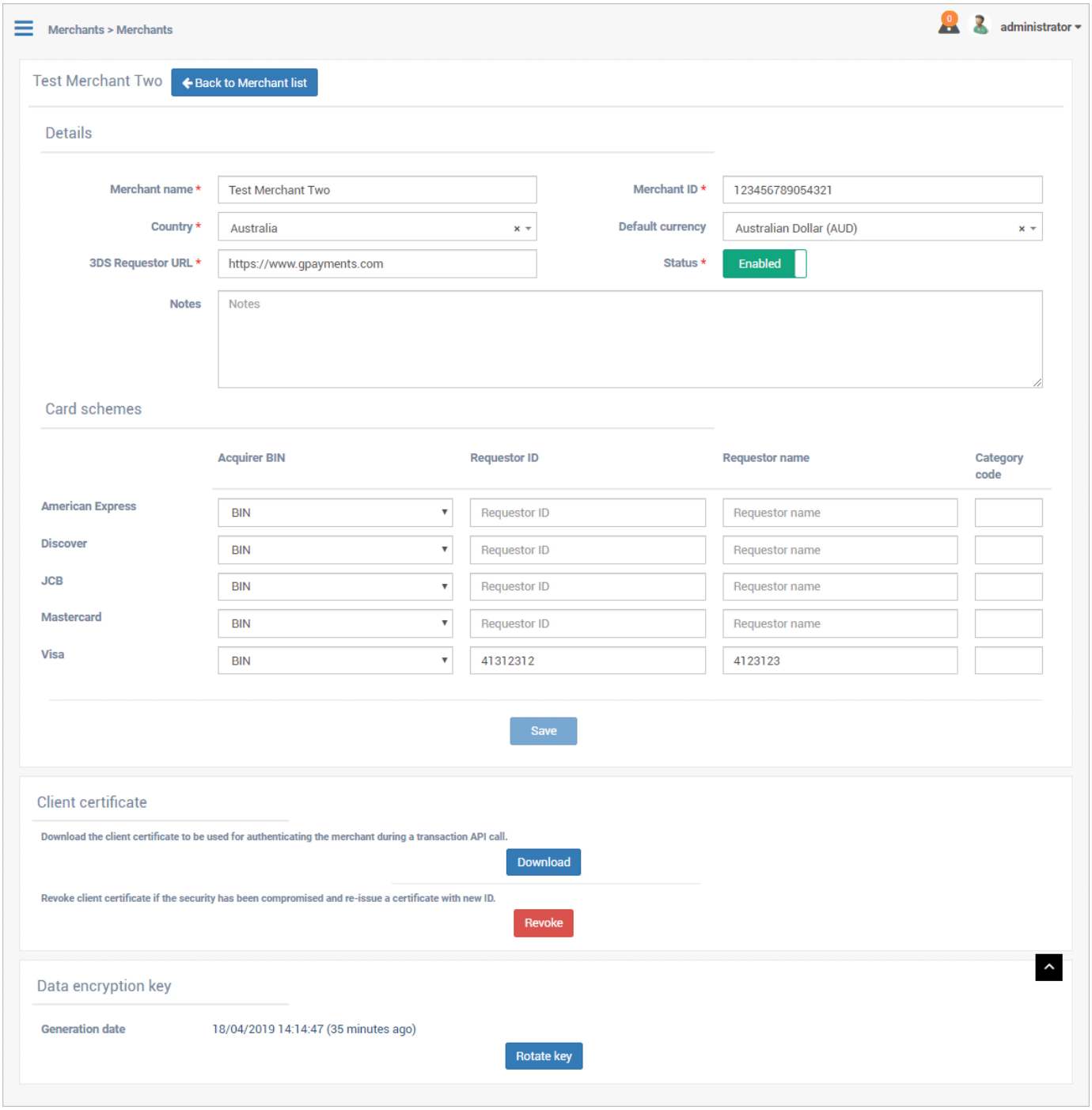
These are the card scheme specific details used for authentication requests:

- **Acquirer BIN** - acquiring institution identification code as assigned by the DS that is receiving the AReq message. *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.



Merchant Security

The Merchant's **Client Certificate**, 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**
>
> **Client certificate** and **CA certificate** management is only available once the instance has been activated.

Client Certificate

The 3DS Requestor client certificate is required for a merchant to include in the authentication API requests for SSL authentication:

- **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.

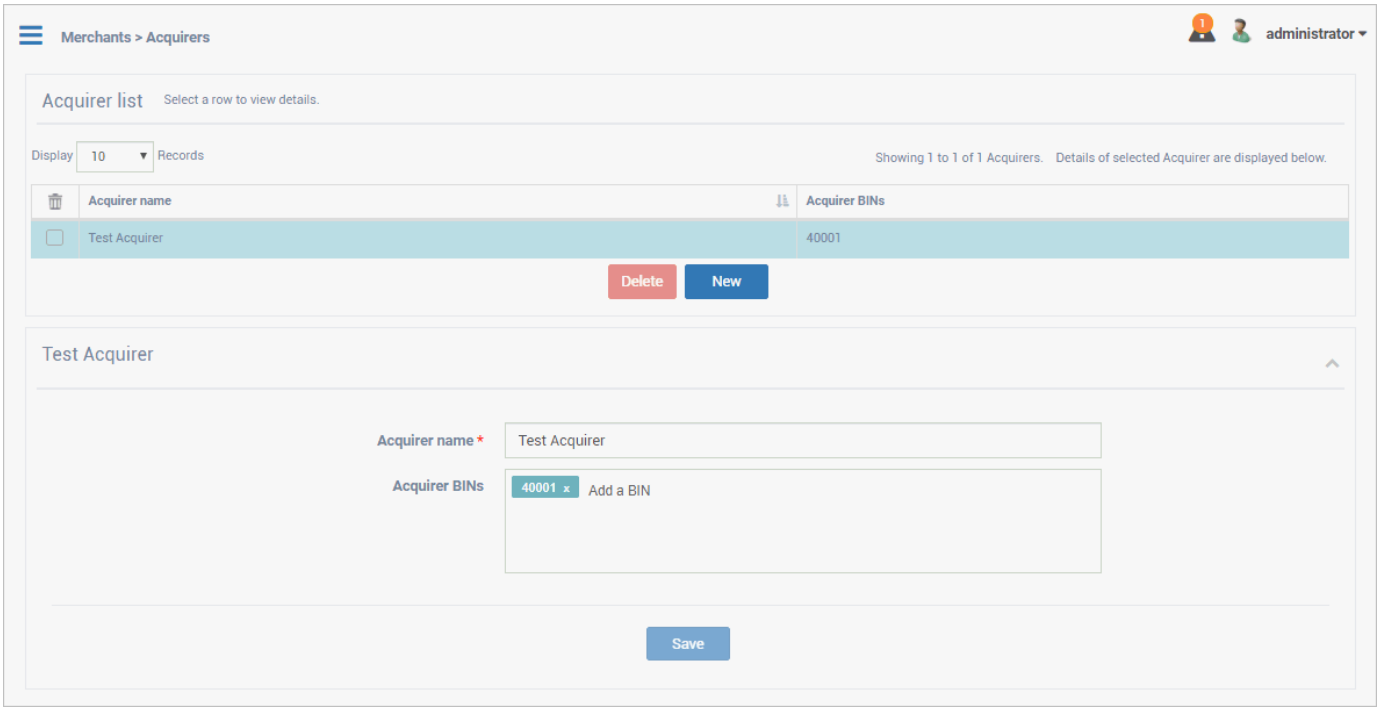
User Access

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.



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 ***Save*** 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.

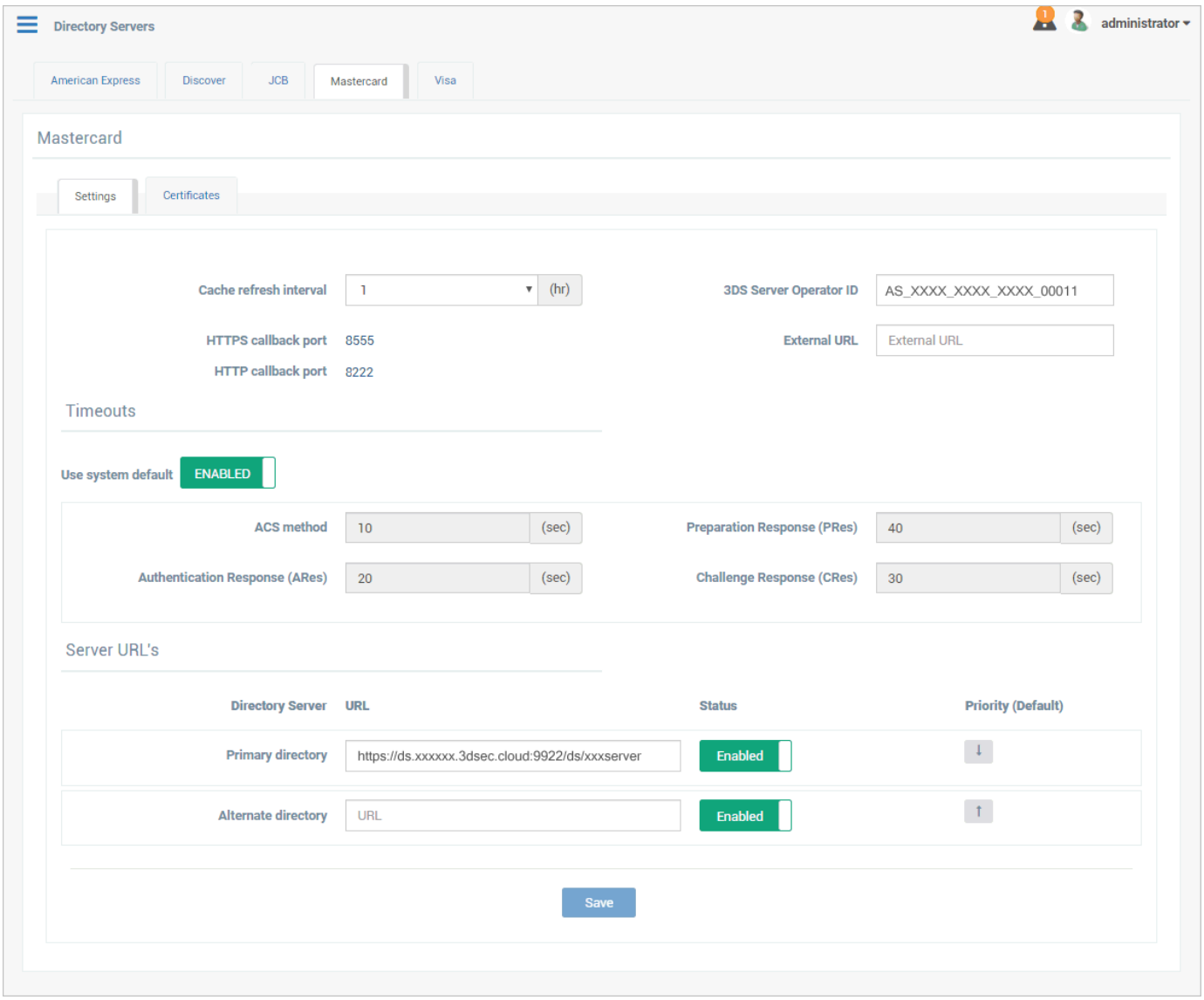
Delete an acquirer

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.



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** - 3DS Servers are required to make a call to each registered DS every 24 hours at a minimum, and once per hour at a maximum to refresh their cache. The cache contains 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. (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.

Timeouts

The following settings can be edited:

- **Use system default** - toggle that **Enables** or **Disables** the use of system default 3DS2 message timeout settings:
- **Enabled** - uses the timeout settings defined in 3D Secure 2 settings
- **Disabled** - allows the user to edit the timeout settings individually per DS:
    - **ACS Method**
    - **Preparation response (PRes)**
    - **Authentication response (ARes)**
    - **Challenge response (CRes)**

Server URLs

In the **Server URLs** section, the DS URLs can be input for the addresses of the card scheme DS.

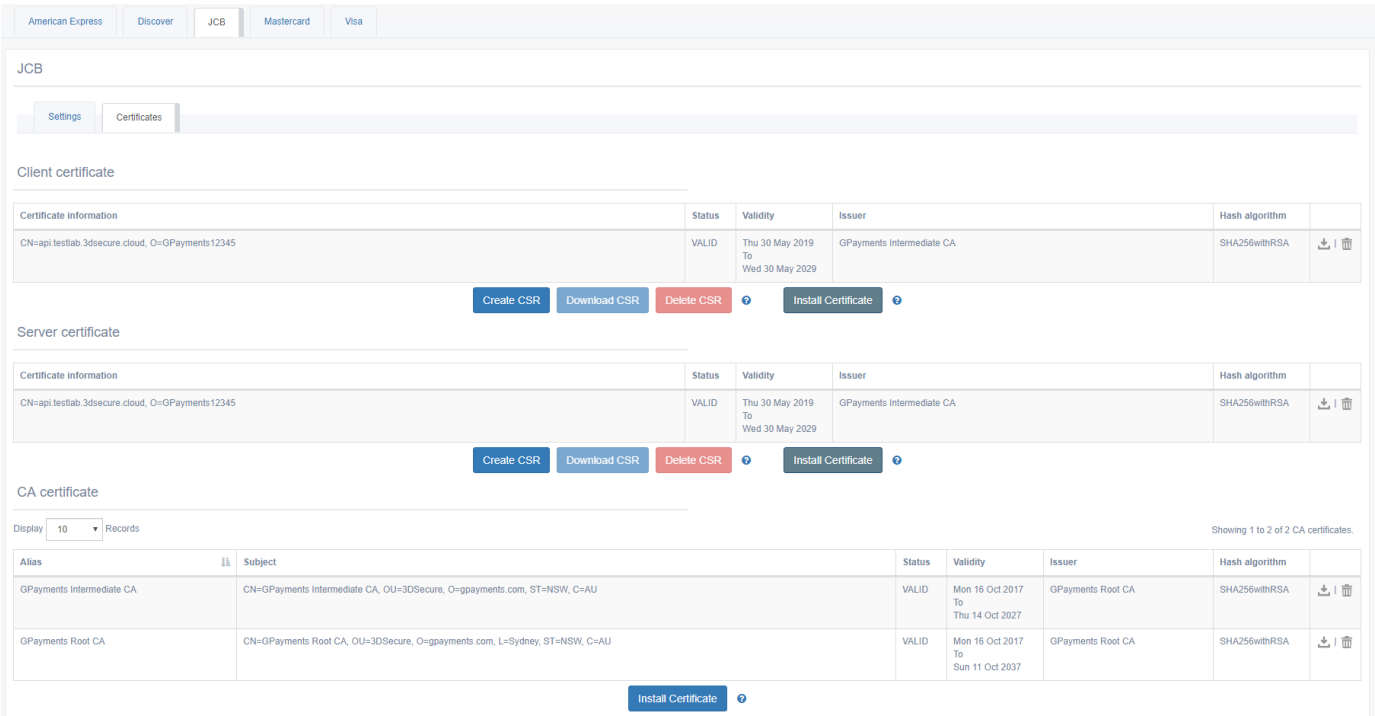
The **Default** URL is used for AReq and PReq message communications with the DS provider.

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.



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

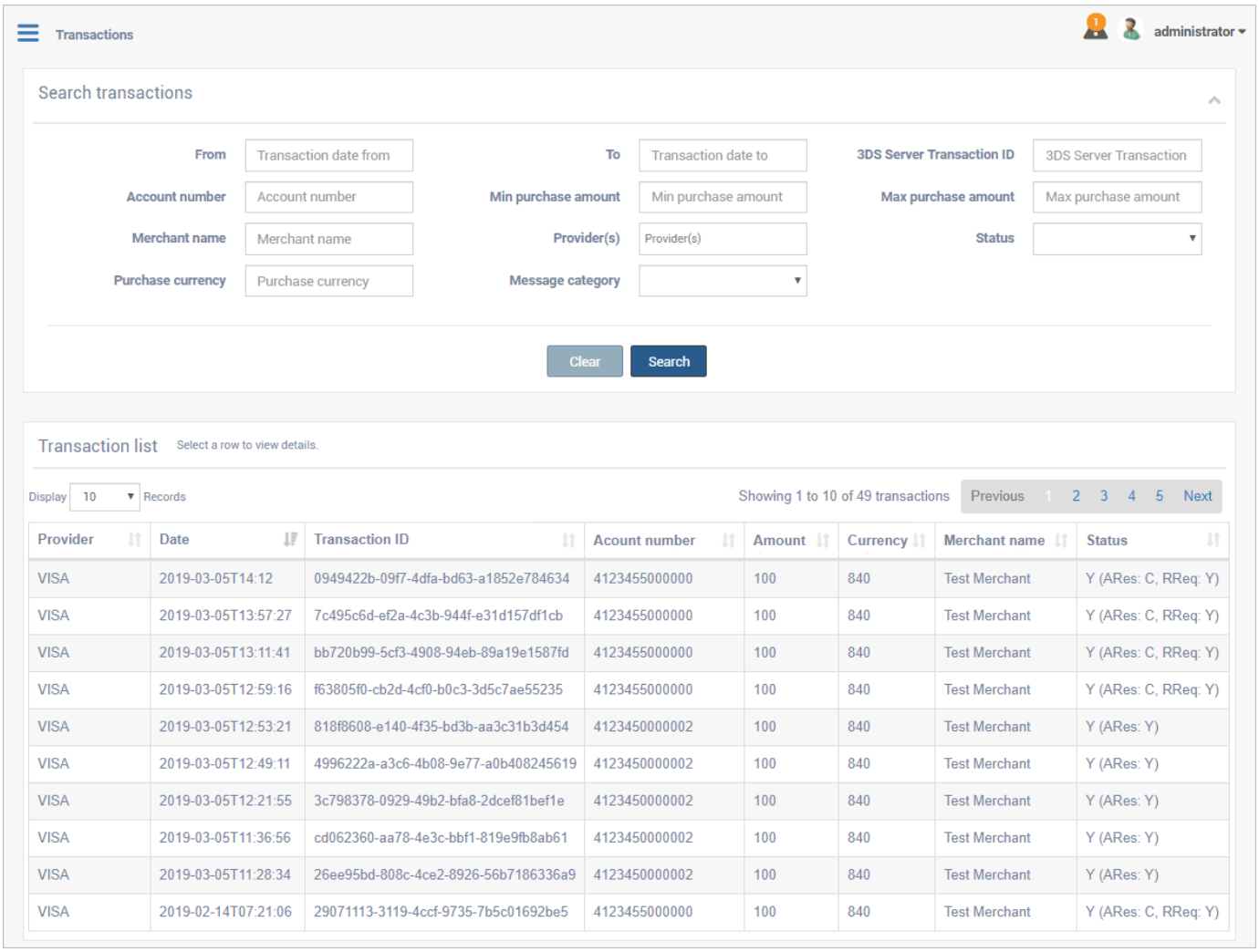
```
openssl pkcs12 -export -out certifcate.p12 -inkey privateKey.key -in
signedcertificate.crt -certfile ca-chain.crt
```

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



Provider	Date	Transaction ID	Acount number	Amount	Currency	Merchant name	Status
VISA	2019-03-05T14:12	0949422b-09f7-4dfa-bd63-a1852e784634	4123455000000	100	840	Test Merchant	Y (ARes: C, RReq: Y)
VISA	2019-03-05T13:57:27	7c495c6d-ef2a-4c3b-944f-e31d157df1cb	4123455000000	100	840	Test Merchant	Y (ARes: C, RReq: Y)
VISA	2019-03-05T13:11:41	bb720b99-5cf3-4908-94eb-89a19e1587fd	4123455000000	100	840	Test Merchant	Y (ARes: C, RReq: Y)
VISA	2019-03-05T12:59:16	f63805f0-cb2d-4cf0-b0c3-3d5c7ae55235	4123455000000	100	840	Test Merchant	Y (ARes: C, RReq: Y)
VISA	2019-03-05T12:53:21	818f8608-e140-4f35-bd3b-aa3c31b3d454	4123450000002	100	840	Test Merchant	Y (ARes: Y)
VISA	2019-03-05T12:49:11	4996222a-a3c6-4b08-9e77-a0b408245619	4123450000002	100	840	Test Merchant	Y (ARes: Y)
VISA	2019-03-05T12:21:55	3c798378-0929-49b2-bfa8-2dcef81bef1e	4123450000002	100	840	Test Merchant	Y (ARes: Y)
VISA	2019-03-05T11:36:56	cd062360-aa78-4e3c-bbf1-819e9fb8ab61	4123450000002	100	840	Test Merchant	Y (ARes: Y)
VISA	2019-03-05T11:28:34	26ee95bd-808c-4ce2-8926-56b7186336a9	4123450000002	100	840	Test Merchant	Y (ARes: Y)
VISA	2019-02-14T07:21:06	29071113-3119-4ccf-9735-7b5c01692be5	4123455000000	100	840	Test Merchant	Y (ARes: C, RReq: 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** - account number used in the transaction, may be represented by PAN or token.
- **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.

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
- **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/CRes/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

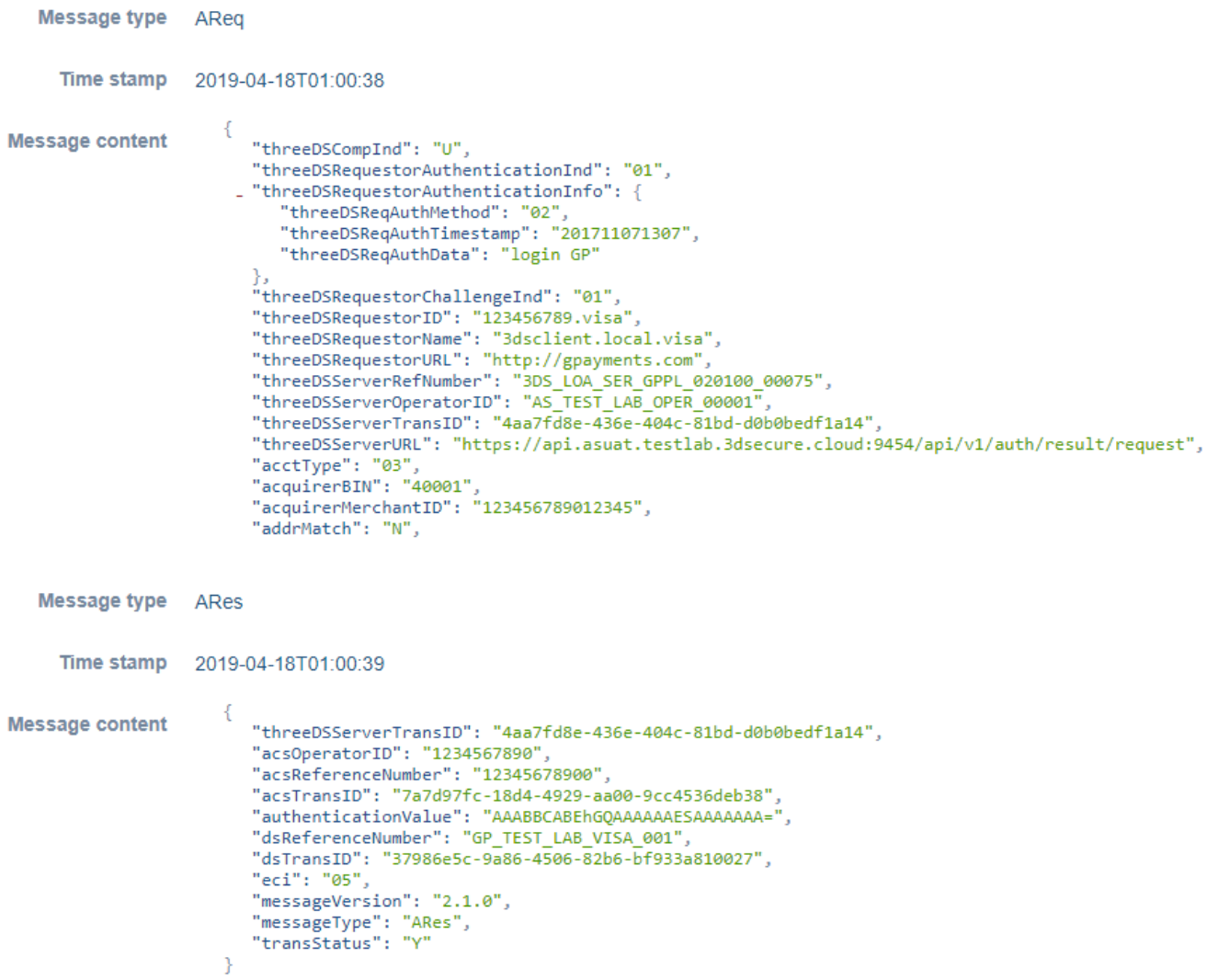
- **Authentication value** - payment system-specific value provided as part of the ACS registration for each supported DS. Authentication Value may be used to provide proof of authentication, e.g. during authorisation or clearing
- **Authentication type** - indicates the type of authentication method 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** - 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

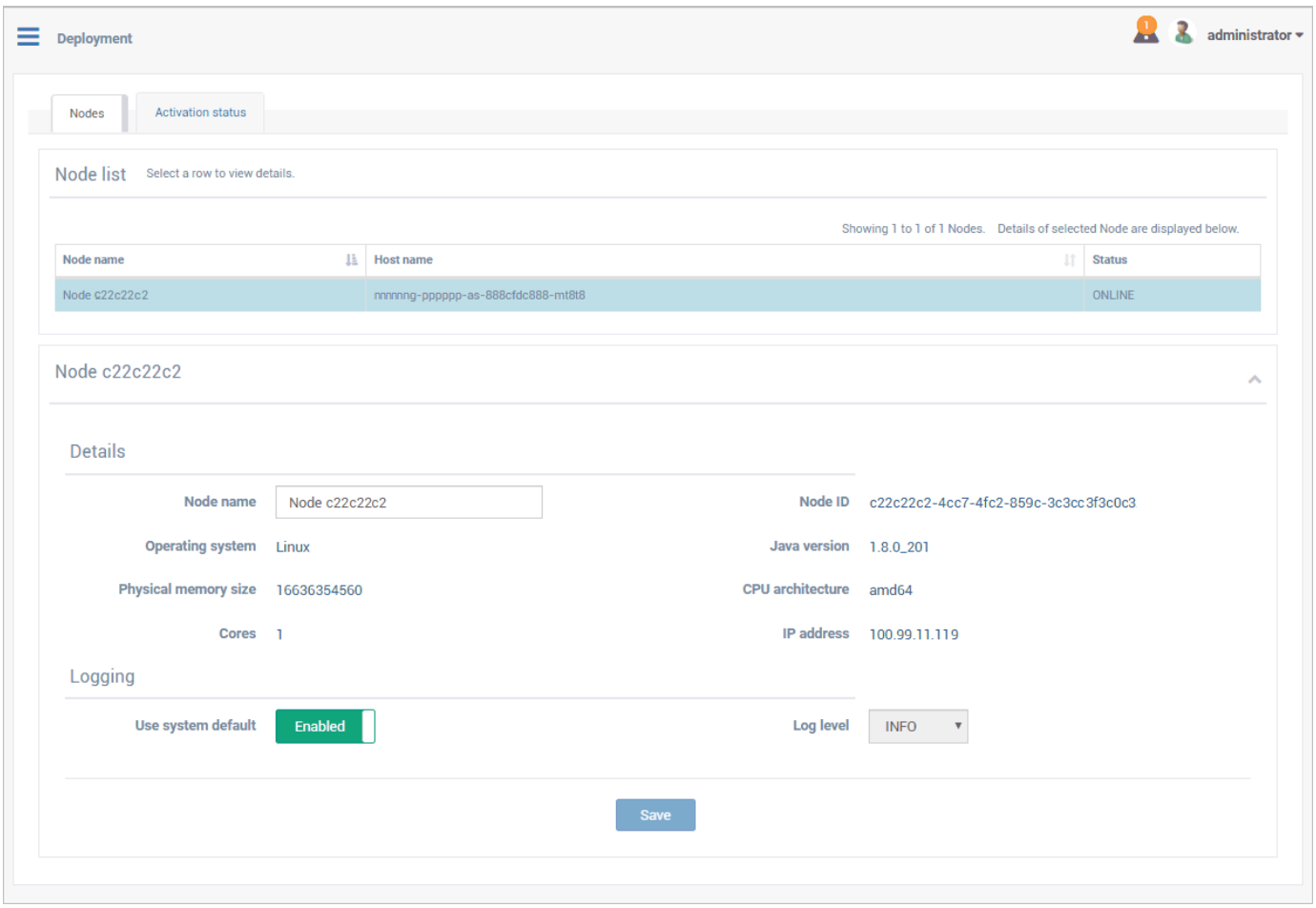
ActiveServer can be run in either a **single node** or **multi node** setup. The details for these nodes can be viewed on the **Deployment** page under the **Nodes** tab. This is also where you can set the logging level for a particular node, if it requires a different setting to the system wide log 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 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.

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:

- 3D Secure 2
- System
- Security

3D Secure 2

The **3D Secure 2** tab has 2 sections:

Settings

- **External URL** - externally accessible URL, used for authentication callbacks and product activation.
- **API URL** - URL used to generate client certificates for the APIs and receive API requests. If it is not provided by default it will use the domain name in the External URL for client certificate generation. Please note this URL does not have to be publicly accessible.
- **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.

Timeouts

- **Preparation Response (PRes)** - timeout interval for the PRes message
- **Authentication Response (ARes)** - timeout interval for the ARes message

- **Challenge Response (CRes)** - timeout interval for the CRes message

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. Please 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.

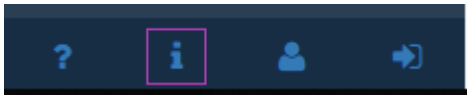
Version 1.0.4

This feature was added in the version 1.0.4 release.

System

- **Log level** - verbosity of the console output and system logs. Possible values in least verbose to most verbose order: **ERROR** > **INFO** > **DEBUG**.

View ActiveServer Information

The **About ActiveServer** page can be accessed from the icon at the bottom left corner of the page. 

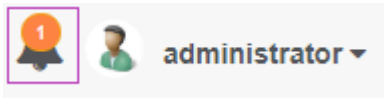
About ActiveServer displays basic information about the software deployment, which includes:

- **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.
- **Update available** - indicates if a newer version of ActiveServer is available for update.

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

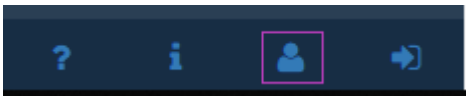
User notifications are shown to all users when an event will impact their account. The following table indicates the possible user notifications and how to resolve them.

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.

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.

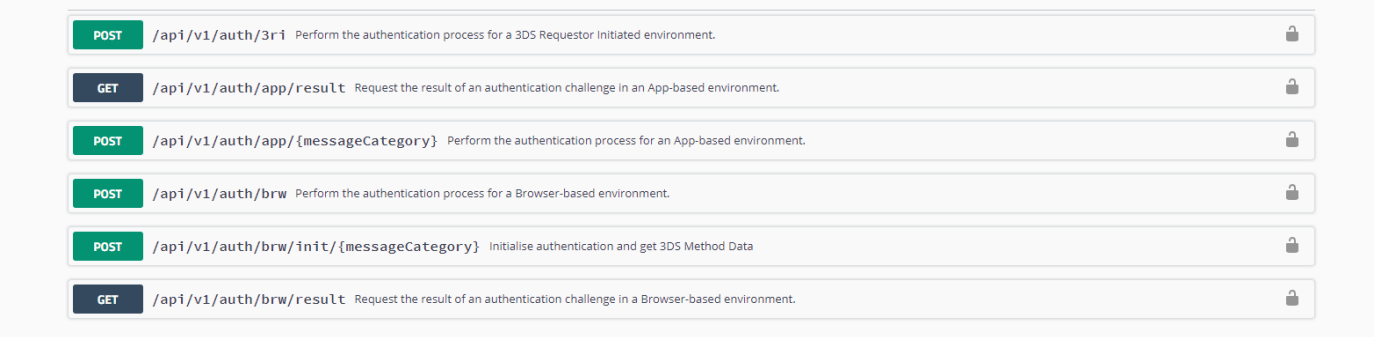
- **New password (required)** - new password for the user, must conform to the Password history check rules.
    - ***Requirements*** *- Between 8 and 100 characters, must contain at least one letter and one number*.
- **Confirm new password** - new password for the user, must match **New password** field.

API document overview

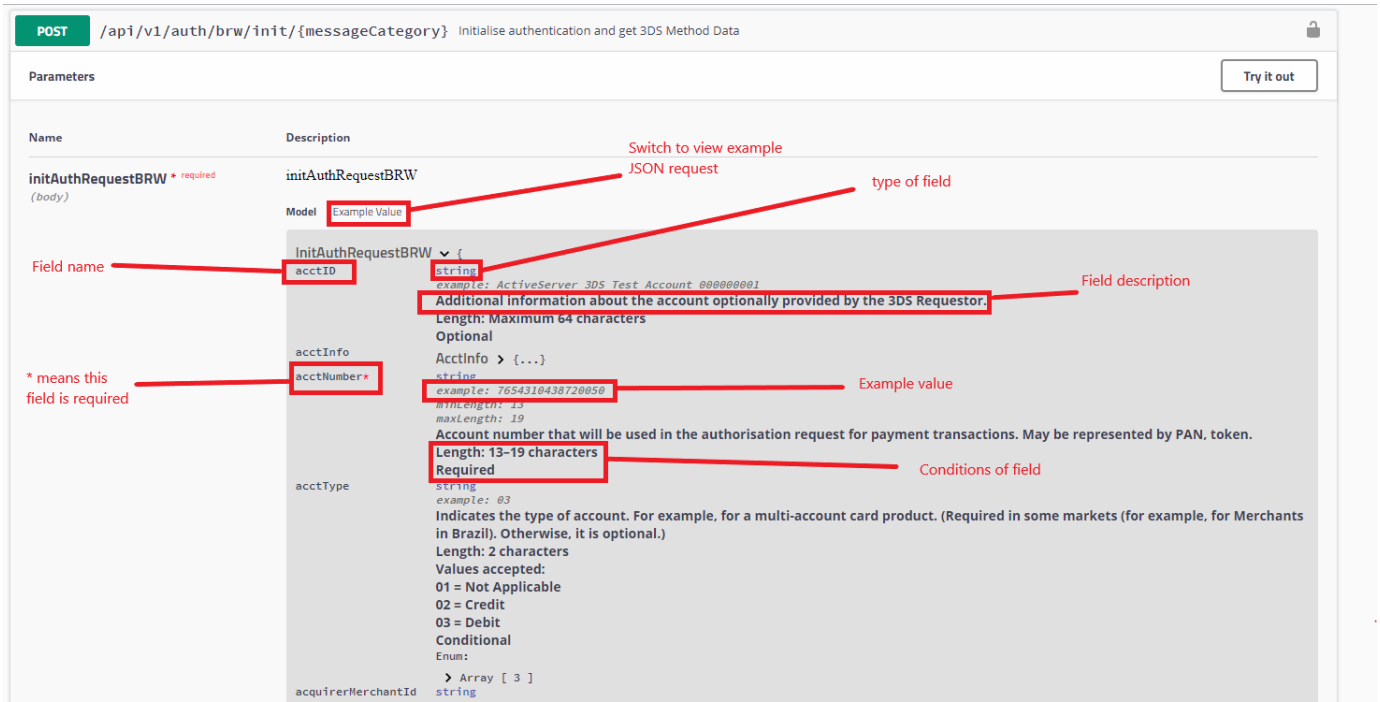
Getting started

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` .



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.



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.

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 issued by the ActiveServer.

1. Make sure the instance is activated by following this guide.
2. Download the client certificate for the Authentication API from the merchant page in the dashboard. This certificate is in `.p12` format.
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.
4. You can follow the Integration guide to setup a 3DS requestor to call the authentication API with the client certificate.

> **Warning**
>
> You will not see the `DOWNLOAD` button if your instance is not activated.

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`**.

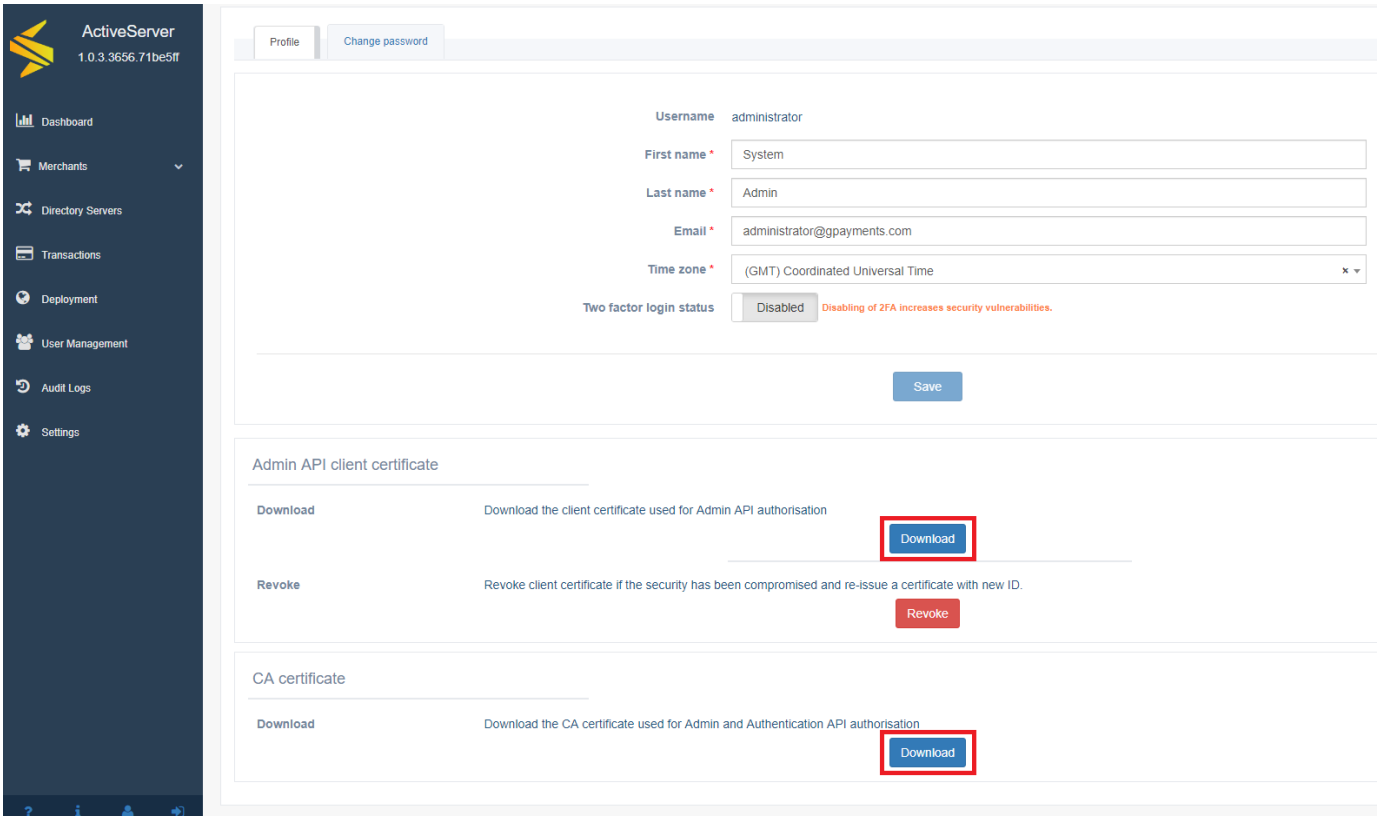
The latest Administration APIs are available from here.

API Authentication

API Authentication is performed similar to the Authentication API with a client certificate.

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. Please refer to the HTTP client you are using on how to do this or you can follow the API quick start for testing.



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. Open up a terminal.
4. 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
```

5. 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
```

6. 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.pem
```

7. 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"}]%
```

Error codes

This section provides details of errors that could occur during the running of ActiveServer.

3DS Error Codes (101 ~ 404)

Code	Name	Description
101	MESSAGE_RECEIVED_INVALID	Received message is invalid.
102	MESSAGE_VERSION_NUMBER_NOT_SUPPORTED	Unsupported message version number.
103	SENT_MESSAGES_LIMIT_EXCEEDED	Message sent exceeds the limit.
201	REQUIRED_DATA_ELEMENT_MISSING	A message element required as defined according the specification is missing.
202	CRITICAL_MESSAGE_EXTENSION_NOT_RECOGNISED	Message extension that is critical is not present.
203	FORMAT_OF_ONE_OR_MORE_DATA_ELEMENTS_IS_INVALID_ACCORDING_TO_THE_SPECIFICATION	Data element is not in the required format or value is invalid as defined according the specification.
204	DUPLICATE_DATA_ELEMENT	Found duplicate data element.
301	TRANSACTION_ID_NOT_RECOGNISED	Transaction ID received is not valid for the receiving component.
302	DATA_DECRYPTION_FAILURE	Encryption of data has failed.
303	ACCESS_DENIED_INVALID_ENDPOINT	Endpoint for the API request is invalid. Check the requesting url.
304	ISO_CODE_INVALID	ISO code is invalid.
305	TRANSACTION_DATA_NOT_VALID	Transaction data is invalid.

Code	Name	Description
306	MERCHANT_CATEGORY_CODE_MCC_NOT_VALID_FOR_PAYMENT_SYSTEM	Merchant category code is invalid.
307	SERIAL_NUMBER_NOT_VALID	Serial number is invalid.
402	TRANSACTION_TIMED_OUT	Transaction has timed out.
403	TRANSIENT_SYSTEM_FAILURE	System has failed for a short period.
404	PERMANENT_SYSTEM_FAILURE	System has failed permanently.
404	SYSTEM_CONNECTION_FAILURE	Failed to connect to the system.

Transaction Error Codes (1000 ~ 1026)

Code	Name	Description
1001	DIRECTORY_SERVER_NOT_FOUND	No directory server found for specified provider
1002	ERROR_SAVE_TRANSACTION	Error occurred while saving transaction
1003	ERROR_SAVE_TRANSACTION_MESSAGE	Error returned while saving transaction message
1004	UNHANDLED_EXCEPTION	Unhandled exception
1005	PAN_NOT_PARTICIPATING	Primary Account Number (PAN) is not participating.
1009	MERCHANT_INTERFACE_DISABLED	The interface is disabled for this merchant
1011	INVALID_LICENSE	Merchant has no valid license
1013	INVALID_TRANSACTION_ID	Transaction ID of 3DS server is not recognised
1014	INVALID_REQUESTOR_TRANSACTION_ID	Transaction ID of 3DS requestor is not recognised
1015	THREEDS_REQUESTOR_NOT_FOUND	Invalid 3DS Requestor ID/Merchant ID
1016	MISSING_REQUIRED_ELEMENT	Required element missing.

Code	Name	Description
1018	ELEMENT_NOT_DEFINED	Message element not a defined message.
1019	PROTOCOL_OLD	Protocol version is too old.
1020	ERROR_TRANSMISSION_DATA	Errors in data transmission
1021	PRIOR_TRANS_ID_NOT_FOUND	Error in setting requestor prior Transaction ID. Prior Transaction ID couldn't be found.
1022	INVALID_FORMAT	Format of one or more elements is invalid according to the specification.
1023	CARD_RANGE_IS_NOT_VALID	Card range provided is invalid.
1024	CACHE_UPDATE_IS_DISABLE	Cache update is disabled.
1025	CACHE_REFRESH_INTERVAL_IS_NOT_SET	Cache refresh interval is not set.
1026	MERCHANT_ID_THREEDS_REQUESTOR_ID_INVALID	Invalid acquirerMerchantID/threeDSRequestorID

General Error Codes (2000 ~ 2010)

Code	Name	Description
2000	NOT_FOUND	Resource not found.
2001	DUPLICATE_RECORD	Record already exists.
2002	VALIDATION_ERROR	Invalid inputs.
2003	INVALID_REQUEST	Invalid request.
2004	CONCURRENCY_FAILURE	Failed to update node.
2005	ACCESS_DENIED	Access is denied.
2006	METHOD_NOT_SUPPORTED	Request HTTP method is not supported.

Code	Name	Description
2007	INTERNAL_SERVER_ERROR	Internal server error.
2008	DATA_INTEGRITY_VIOLATION_ERROR	A specified value violated the integrity constraints.
2009	SESSION_TIMED_OUT	Session has timed out.

Security Error Codes (3001 ~ 3024)

Code	Name	Description
3002	NO_SUCH_ALGORITHM	No such algorithm.
3003	INVALID_CERT	The certificate's public key is not compatible with the corresponding private key.
3004	INVALID_CHAIN	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	No private key found.
3006	INVALID_CERTIFICATE_CONTENT	Invalid certificate content
3007	CERTIFICATE_IO_READ	Unable to read certificate.
3008	SUCH_PROVIDER_EXCEPTION	No such provider exception.
3009	NO_KEY	The certificate could not be installed because this object does not have an existing key.
3010	CERTIFICATE_CHAIN_BAD_FORMAT	Certificate chain has invalid format.
3011	MISMATCHED_PASSWORDS	Password fields do not match.
3012	IMPORT_CERTIFICATE	Please import client certificate.

Code	Name	Description
3013	IMPORT_NO_CERTIFICATE	There is no certificate to export.
3014	FAILED_TO_INITIALIZE	Failed to initialise.
3015	ENCRYPTION_FAIL	Failed to encrypt.
3016	DECRYPTION_FAIL	Failed to decrypt.
3017	INVALID_HSM_PROVIDER	The specified provider name for hardware encryption is not supported
3018	INVALID_PKCS11_CONFIG	Invalid PKCS11 config path
3019	FAILED_TO_INITIALIZE_PKCS11	Failed to initialise PKCS11.
3020	IMPORT_FAIL	Failed to import.
3021	NOT_SUPPORTED_IBM_PROVIDER	Only SUN provider is supported.
3022	UNABLE_TO_LOAD_KEYSTORE	Loading keystore failed.
3023	UNABLE_TO_LOAD_CERTIFICATE	Loading certificate failed.
3024	INVALID_KEY_SIZE	Key size is invalid.

User Error Codes (4000 ~ 4022)

Code	Name	Description
4000	DUPLICATE_EMAIL	E-mail already in use.
4001	LAST_ADMIN_DELETE_NOT_ALLOWED	You need to be at least a System Admin user to perform this action.
4002	ACCOUNT_IS_LOCKED	Your account is locked.
4003	ACCOUNT_IS_DISABLED	Your account is disabled.

Code	Name	Description
4004	ACCOUNT_WILL_BE_LOCKED	Your account will be locked after another wrong try. If you have been forgotten your password please click on "Lost your password"
4005	ACCOUNT_WAS_LOCKED	Password has been locked for 1 hour.
4006	ACCOUNT_IS_INACTIVE	Your account was not activated.
4007	PASSWORD_POLICY_MATCH	The password should be minimum eight characters, with at least one letter and one number.
4008	LOGIN_ALREADY_IN_USE	Username already in use.
4009	EMAIL_ALREADY_IN_USE	Email already in use.
4010	INVALID_TOTP_CODE	Invalid totp authentication code.
4011	EMAIL_SENDING_FAILED	Failed to send email.
4012	EMAIL_NOT_REGISTERED	Your email is not registered.
4014	FAILED_TO_CREATE_ACCOUNT	Failed to create the account.
4015	TWO_FA_MANDATORY	Using two factor login is mandatory.
4016	PASSWORD_EXPIRED	The password for user was expired.
4017	PASSWORD_EXPIRED_WARNING	The password for user is going to be expired on.
4018	PASSWORD_HISTORY_MATCHED	The password matched with the previous historical passwords.
4019	INVALID_TOKEN	An invalid token.
4020	INVALID_HSM_PIN	Invalid HSM Pin.
4021	INVALID_PASSWORD	Invalid password.
4022	EMAIL_INVALID_ACTIVATION	Account activation code is invalid.

Setup Error Code (5000)

Code	Name	Description
5000	SETUP_NOT_ALLOWED	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

Date	ActiveServer Version	Documentation Version	Change Details
16/08/19	1.1.0	1.1.0:1	• Added supported web browsers to the **Quickstart** guide • Updated the Manage DS Certificates guide with information on how certificates are used in **ActiveServer** as well as newly introduced CSR management features • Updated **Quickstart** guide with details on setting TLS version • Updated compatible database versions for PostgreSQL (add 8.4 and later) and DB2 (add 11.1 and later) • Added instructions to change session timeout setting • Updated the /api/v1/auth/app/{messageCategory} API example to reflect the `sdkEphemPubKey` attribute being an object and updated the sample JSON • Updated Manage DS Settings guide to reflect new optional PReq URL endpoint being added
16/07/19	1.0.5	1.0.5:2	• Added functionality to choose documentation version and language, as well as downloading the documentation as PDF
04/07/19	1.0.5	1.0.5:1	• Updated the Manage merchants guide with details of CA certificate download • Updated purchaseCurrency and purchaseAmount descriptions on /api/v1/auth/brw/init/ API request • Updated /api/v1/auth/3ri API request to require a *{messageCategory}* • Updated system labels for `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) • Updated the test card number used for challenge authentications

Date	ActiveServer Version	Documentation Version	Change Details
07/06/19	1.0.4	1.0.4:2	• Updated the **API document overview** to include field condition definitions and updated the **Front-end implementation** and **Step-by-step** guides to reflect latest changes to 3DS Requestor demo code on GitHub • Updated `BaseMerchantProvider` object used in /api/v1/admin/merchants to use `acqBinId` only
31/05/19	1.0.4	1.0.4:1	• Added a new Settings > HSM section, explaining how to use the HSM PIN change functionality • Updated **Quickstart** document with additional information for using Amazon S3 Keystore as the system keystore, including using IAM roles
30/05/19	1.0.3	1.0.3:2	• Updated **Quickstart** document with MS SQL sample properties • Updated compatible database versions for MySQL (add Amazon Aurora), Oracle (add 12c) and MS SQL (add 2008 R2, 2012, 2014, 2016)
27/05/19	1.0.3	1.0.3:1	• Updated the API document overview with details on how to download CA certificate chains and access Administration APIs using .X509 authentication • Updated the User profile guide with details of certificate download and revoke • Updated the Enrol API result description with details of the `enrolmentStatus`
24/05/19	1.0.2	1.0.2:1	• Added support for MSSQL 2017 • Updated **Quickstart** document with new default DS port numbers and updated supported Oracle DB version to **11g** • Updated Logging section to reflect new log file format (`as.yyyy-mm-dd.log`) • Added the Upgrade instance guide • Updated the **Activate instance** document, as MyAccount no longer requires the **External URL** during server registration
08/05/19	1.0.0	1.0.0:4	• Added data elements overview to the Activate instance guide

Date	ActiveServer Version	Documentation Version	Change Details
03/05/19	1.0.0	1.0.0:3	• Restructured the old **Authentication Integration** section into multiple pages along with adding more information to assist in explaining the integration process. There is now a high level overview set of documents, along with the step-by-step integration guides under the **Guides > Integration** heading
30/04/19	1.0.0	1.0.0:2	• Minor updates to fix broken links
18/04/19	1.0.0	1.0.0:1	• Initial release

Release notes

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
[#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

Change	Description
[#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

Change	Description
[#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
[#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

Change	Description
[#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

GPayments reserves all rights to the confidential information and intellectual property contained in this document. This document may contain information relating to the business, commercial, financial or technical activities of GPayments. This information is intended for the sole use of the recipient, as the disclosure of this information to a third party would expose GPayments to considerable disadvantage. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any process without prior written permission. This information is provided under an existing non-disclosure agreement with the recipient.

Copyright Statement

All third party product and service names and logos used in this document are trade names, service marks, trademarks, or registered trademarks of their respective owners.

The example companies, organisations, products, people and events used in screenshots in this document are fictitious. No association with any real company, organisation, product, person, or event is intended or should be inferred.

Disclaimer

GPayments Pty Ltd makes no, and does not intend to make any, representations regarding any of the products, protocols or standards contained in this document. GPayments Pty Ltd does not guarantee the content, completeness, accuracy or suitability of this information for any purpose. The information is provided "as is" without express or implied warranty and is subject to change without notice. GPayments Pty Ltd disclaims all warranties with regard to this information, including all implied warranties of merchantability and fitness for a particular purpose and any warranty against infringement. Any determinations and/or statements made by GPayments Pty

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.