フロントエンド実装(v2)

このセクションでは、サンプルコードを使用して加盟店アプリケーションのためのフロントエンドの実装方法を示します。

サンプルコードパッケージ内の以下のファイルはフロントエンド内の3DS2認証に必須です。必要な場合はディレクトリツリーで詳細を確認してください。

- `v2/process.html` - すべての認証シーケンスを実装しています。
- `v2/3ds-web-adapter.js` - 3DS2データをフロントエンドからバックエンドへ渡し、コールバックURLの為に必要な `iframe` を生成する**3DSクライアント**の重要なコンポーネントです。
- `notify_3ds_events.html` - **ActiveServer**の為に使用されるコールバックページで、認証において(ステップ 7とステップ 18(C))を開始する際に必要になります。このページはチェックアウト処理に認証イベントを渡します。

以下は認証処理と認証シーケンスに基づいたフロントエンド実装の詳細です。

> **重要**
>
> 以下のサンプルコードは開発用ガイドのための例であるため、商用環境で十分にテストされたものではありません。そのため商用環境向けにご利用いただく場合は、事前にレビューおよび修正の上、お客様の環境上適切であるかご確認ください。

処理1: 認証の初期化

認証を初期化する為に、フロントエンドが必要な事:

- `認証の初期化` メッセージを**3DSリクエスター**へ送信する (ステップ 1とステップ 2) 。
- 応答メッセージを受信しコールバック `iframe` をセットアップする (ステップ 5とステップ 6) 。

RBC

3DSリクエスターがブラウザ情報を提供する場合、`threeDSMethodAvailable` が `false` であれば `iframe` の設定はスキップできます。

ユーザーがチェックアウトページ内の**チェックアウトを続ける**ボタンをクリックすると、ブラウザは必要なデータをセッションストレージに保存し `process.html` ページに移動します。

```
//checkout.html
function checkout() {
    var apiVersion = getApiVersion();
    switch (apiVersion) {
      case "v2":
        goApiV1();
        break;
       ...
    }
}

function goApiV1() {
    var sessionData = genSessionData();
    sessionData.messageCategory = "pa";
    sessionStorage.setItem("sessionData", JSON.stringify(sessionData));
    window.location.href = "/v2/process";
}

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

注目

`sessionData` が含むもの:

- **`channel`** : `brw` または `3ri` 。上記の例では、ブラウザベースの認証を実行するために `brw` チャンネルを使用しました。
- **`messageCategory`** : `pa` -決済認証または `npa` -非決済認証のいずれか。上記の例では、 `pa` を使用しました。
- **`authData`** : JSON形式のすべての必要なデータ。データ構造については、APIドキュメントを参照してください。
- **`backButtonType`** : プロセスページの `back` ボタンタイプを示します。上記の例では `戻る` ボタンを `ショップに戻る` として作成します。

注意点：ページ間通信に `sessionStorage` を使用するのは、デモ用です。 3DSリクエスターの実際の実装では、既存のチェックアウトプロセスと統合するために、ページ間でパラメーターを転送するための最適なアプローチを選択しましょう。

`process.html` ページ内で、 `sessionData` 受信し3DS2処理を開始します。まず、認証を初期化する為の情報を**3ds-web-adapter**に送信します (ステップ 1) 。

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

3ds-web-adapter内の `brw()` 関数は以下のパラメータを含みます:

- `authData` : JSON形式のすべての必要なデータ。データ構造については、APIドキュメントを参照してください。
- `container` : 複数の `iframe` を生成する**3ds-web-adapter**の事前定義されたコンテナー。

- `callbackFn` ：認証結果を処理するためのコールバック関数。この関数は、3ds-web-adapterからのコールバックを処理するために、加盟店側で実装する必要があります。`callbackFn` は、 `type` と `data` の2つのパラメーターを受け取る必要があります。
  - `type` イベントの種類を示します。受け入れられる値：
    - `onAuthResult` 認証結果を示します
    - `on3RIResult` 3riの結果を示す
    - `onEnrolResult` Enrolの結果を示します
    - `onChallengeStart` チャレンジを開始することを示す
    - `onDecoupledAuthStart` デカップルド認証を開始することを示します。
    - `onError` エラーを示します
  - `data` 3ds-web-adapterから返されたデータ。データには、「auth」、「3ri」の結果、またはエラーメッセージのいずれかが含まれます。

  例として、 `process.html` ページの `_callbackFn（type、data）{...}` 関数を利用できます。
- `options` ：3DSリクエスターのオプションのパラメーター。次のオプションを使用できます。
  - チャレンジキャンセル - `options.cancelChallenge = true` を設定して、チャレンジが不要な場合にチャレンジをキャンセルするかどうかを指定します。
  - チャレンジキャンセル理由 - `options.cancelReason` を次のように設定することにより、DS/ACSに詳細情報を提供するためにチャレンジがキャンセルされた場合に提供するオプション。
    - `01` または `CReqNotSent` （大文字と小文字は区別されません）
    - `02` または `AuthResultNotDelivered` （大文字と小文字は区別されません）
  - チャレンジウィンドウサイズ - `options.challengeWindowSize` を次のように設定して、ACSから要求されるチャレンジウィンドウサイズを指定します。
    - `01` = 250 x400ピクセル
    - `02` = 390 x400ピクセル
    - `03` = 500 x600ピクセル
    - `04` = 600 x400ピクセル
    - `05` = フルスクリーン

- `transType` ：DSプロファイルを選択するためのパラメーター。 `transType = prod` を使用して本番ディレクトリサーバーを使用します。それ以外の場合は、TestLabディレクトリサーバーが使用されます。

次に**3ds-web-adapter**内の `brw()` メソッドから3DSリクエスターバックエンドに認証を初期化する為の情報を送信します(ステップ 2) 。

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

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

  if (options) {
    _options = options;
  }

  //iframeのためにランダムな番号を作成する
  iframeId = String(Math.floor(100000 + Math.random() * 900000));

  // trans-type=prod を使用し本番環境DSを利用する
  var initAuthUrl = "/v2/auth/init";

  //  本番環境DSを利用するためにtrans-type=prodを追加する。詳細はhttps://
docs.activeserver.cloudをご参照ください。
  if (transType === "prod") {
    initAuthUrl = initAuthUrl + "?trans-type=prod";
  }

  var initAuthData = {};
  initAuthData.acctNumber = authData.acctNumber;
  initAuthData.merchantId = authData.merchantId;
  if (options && options.challengeWindowSize) {
    initAuthData.challengeWindowSize = options.challengeWindowSize;
  }

  if (authData.skipAutoBrowserInfoCollect) {
    initAuthData.skipAutoBrowserInfoCollect = true;
  }

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

  //  /auth/initに認証を初期化するためにデータを送信する
  doPost(initAuthUrl, initAuthData, _onInitAuthSuccess, _onError)
}
```

`brw()` 関数は**バックエンド**へ/api/v2/auth/initPOSTリクエストを送信します。APIメッセージオブジェクトはJSON形式で送信されます。**バックエンド**がどのようにこのリクエストを処理するかを確認するにはこちらを参照してください。

3ds-web-adapterは成功した応答を処理する為に `_onInitAuthSuccess()` 関数を使用します(ステップ 5) 。

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

  if (!data.authUrl) {
    _onError(data);
    return;
  }

  serverTransId = data.threeDSServerTransID;

  if (_authData.skipAutoBrowserInfoCollect) {
    // ブラウザ情報収集処理がスキップされていれば、3DSメソッドは実行可能。
    if (data.threeDSServerCallbackUrl) {
      executeIframes(data)
    } else {
      // 3DSメソッドが利用不可の場合、認証処理へ移行する。
      _doAuth(data.threeDSRequestorTransID, _authData.browserInfoCollected)
    }
  } else {
    // 通常通りiframesを実行し、デフォルトのブラウザ情報収集処理を利用する。
    if (data.threeDSServerCallbackUrl) {
      executeIframes(data)
    } else {
      _onError(data);
    }
  }
}
```

3ds-web-adapterは2つの隠し `iframe` をチェックアウトページに挿入します (ステップ 6)。1つ目の `iframe` は(ステップ 7.1 ~ 7.4) 用で、 `threeDSServerCallbackUrl` を使用してブラウザの情報がACSとActiveServerにより収集出来るようにします。

2つ目はオプションのモニタリング `iframe` で、ブラウザ情報を収集する間もしくは3DSメソッドの処理中に何かしらのエラーが発生した時に、 `InitAuthTimedOut` イベントを**ActiveServer**受信できるようになります。このイベントのタイムアウトは**15秒**です。

> **注目**
>
> シーケンス図のステップ 1からステップ 7はこの処理で実装されています。

処理 2: 認証の実行

認証を実行するためにはフロントエンドはブラウザ情報の収集、3DSメソッドデータの収集（収集が可能な場合のみ）を終える必要があります。これらの2つの処理が完了した後、`notify_3ds_events.html` は `3ds-web-adapter` に認証を続けるよう通知します。この処理内で、何らかの理由によりデータ収集が失敗したか、もしくは完了出来なかった場合、InitAuth処理内でセットアップした別のモニタリング `iframe` によって `InitAuthTimedOut` のイベントが `3ds-web-adapter` に通知され、認証処理は終了されます。

以下は認証を実行するためのステップです。

- `notify_3ds_events.html` を実装もしくは提供されたものを再利用し、データ収集についてのイベントを受信します。
- `_on3DSMethodSkipped` もしくは `_on3DSMethodFinished` イベントが通知された後、 `認証の実行` メッセージを**3DSリクエスター**に送信します (ステップ 8とステップ 9) 。
- 認証結果をフリクションレスフローもしくはチャレンジフローで処理します。(ステップ 13に続き、ステップ 14(F)またはステップ 14(C)) または、デカップルド・フロー(Step. 14(D))。

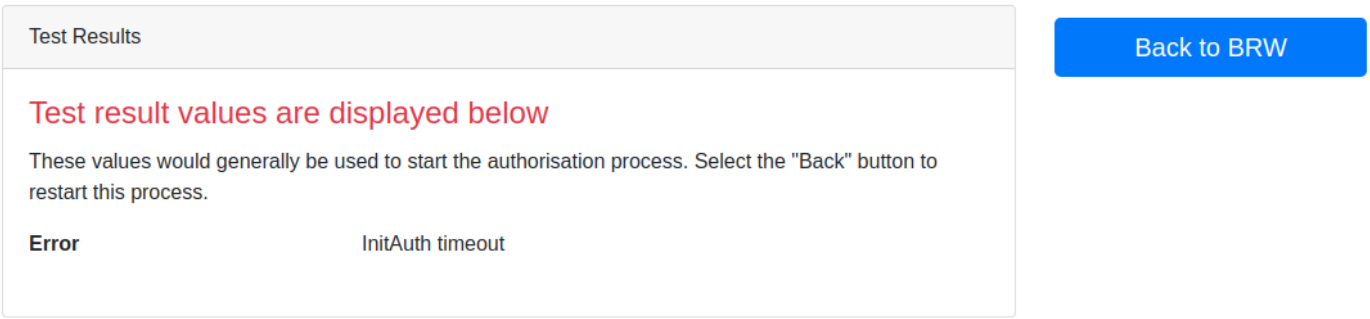
`notify_3ds_events.html` は認証処理を開始するのに使用されます (ステップ 8) 。 **3DSリクエスター**は `notify_3ds_events.html` に `transId` 、 `callbackName` とオプションの `param` 変数を提供します。バックエンド実装を確認するにはこちらを参照してください。


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

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

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script>
  //チェックアウトページに通知を送り、以降の処理を実行する。

  var callbackFn = parent[$('#notifyCallback').val()];
  //callbackFnは3ds-notifyハンドラーメソッドにより定義されています。
  if (typeof callbackFn === 'function') {
    callbackFn($('#transId').val(), $('#param').val());
  }
</script>

</body>
</html>
```

`callbackName` に応じて**3ds-web-adapter**内の異なるメソッドを呼んでいる事が分かります(ステップ 8) 。 `callbackName` の値は `_onThreeDSMethodFinished` 、 `_onThreeDSMethodSkipped` 、 `_onAuthResult` 、もしくは `_onInitAuthTimedOut` です。たとえば、 `_onInitAuthTimedOut` イベントは、3DSメソッドがタイムアウトしたときに使用できます。それぞれのメソッドの説明は以下に記載します。

イベント	説明
`_onThreeDSMethodFinished`	3DSメソッドがACSにより終了し、 `_doAuth()` を呼び出す時である事を通知します

イベント	説明
`_onThreeDSMethodSkipped`	3DSメソッドが(利用出来ない、もしくは別の理由により）スキップされ、`_doAuth()` を呼び出す時である事を通知します3DSメソッドがスキップされたか否かに関わらず、3DSサーバーのブラウザ情報収集は3DSメソッドよりも前に実行される事に注意してください。
`_onAuthResult`	このイベントは認証結果が**ActiveServer**から取得可能になったことを通知します。フリクションレスフローとチャレンジフローで使用されます (ステップ 17(F) と 19(C)) 。
`_onInitAuthTimedOut`	3DSメソッド中もしくはブラウザ情報収集中にエラーが発生した事を通知します。デモ内ではこのイベントが発生した場合、このデモでは、エラーメッセージが出力され、認証プロセスが続行されます。 詳細については、 `3ds-web-adapter.js` の `_onInitAuthTimedOut()` 関数を確認してください。



ここステップ 8で**ActiveServer**により返される `callbackName` は `_onThreeDSMethodFinished` もしくは `_onThreeDSMethodSkipped` のどちらかです。**3ds-web-adapter**はこれらのイベントを受信すると、POSTリクエストを送信し認証を実行する為に `_doAuth()` を呼び出します (ステップ 9) 。

そしてバックエンドはAPI /api/v2/auth/brw の呼び出しを作成します。**バックエンド**がどのようにそのリクエストを処理するか確認するにはこちらを参照してください。

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

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

  //最初に3DSメソッド用のiframeを取り除く
  $("#3ds_" + iframeId).remove();

  var authData = _authData;
  if (!authData.skipAutoBrowserInfoCollect) {
    // skipAutoBrowserInfoCollectが使われなかった場合、通常通りブラウザ情報を収集する。
    authData.browserInfo = param;
  }

  // 認証処理では不要となるためskipAutoBrowserInfoCollectを除外する。
  delete authData.skipAutoBrowserInfoCollect;
  authData.threeDSRequestorTransID = transId;
  authData.threeDSServerTransID = serverTransId;
  console.log("authData: ", authData);
  doPost("/v2/auth", authData, _onDoAuthSuccess, _onError);
}
```

注目

ActiveServer から返却された `param` 変数にはブラウザ情報が含まれており、それらは **`auth`** APIメッセージ内の `authData` に設定する必要があります。

RBC

3DSリクエスターでブラウザ情報収集処理を完了している場合は、 `/auth/brw` 呼び出し時に `browserInfoCollected` に設定する必要があります。

デモ用のコードでは、ブラウザ情報はフロントエンドjavascript (3ds-web-adapter)に設定しておりますが、セキュリティ上の観点を考慮すると、本番環境ではバックエンドにこの処理を実装する必要があります。

3ds-web-adapterは成功した応答を処理する為に `_onDoAuthSuccess()` 関数を使用します(ステップ 13) 。

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

  if (data.transStatus) {
    if (data.transStatus === "C") {
      // 3Dリクエスターはチャレンジフローに移行するか否かを選択できます。
      if (_options.cancelChallenge) {
          if (_options.cancelReason) {
            var sendData = {};
            sendData.threeDSServerTransID = serverTransId;
            sendData.status = _options.cancelReason;
            doPost("/v2/auth/challenge/status", sendData, _onCancelSuccess,
                _onCancelError)
          } else {
            var returnData = _cancelMessage();
            _callbackFn("onAuthResult", returnData);
          }
        } else {
          startChallenge(data);
        }

      } else if (data.transStatus === "D") {
        _startDecoupledAuth(data, getBrwResult);
      else {
        _callbackFn("onAuthResult", data);
      }
    } else {
      _onError(data);
    }
}
```

返された `transStatus` に基づいて異なるフローを実行します。

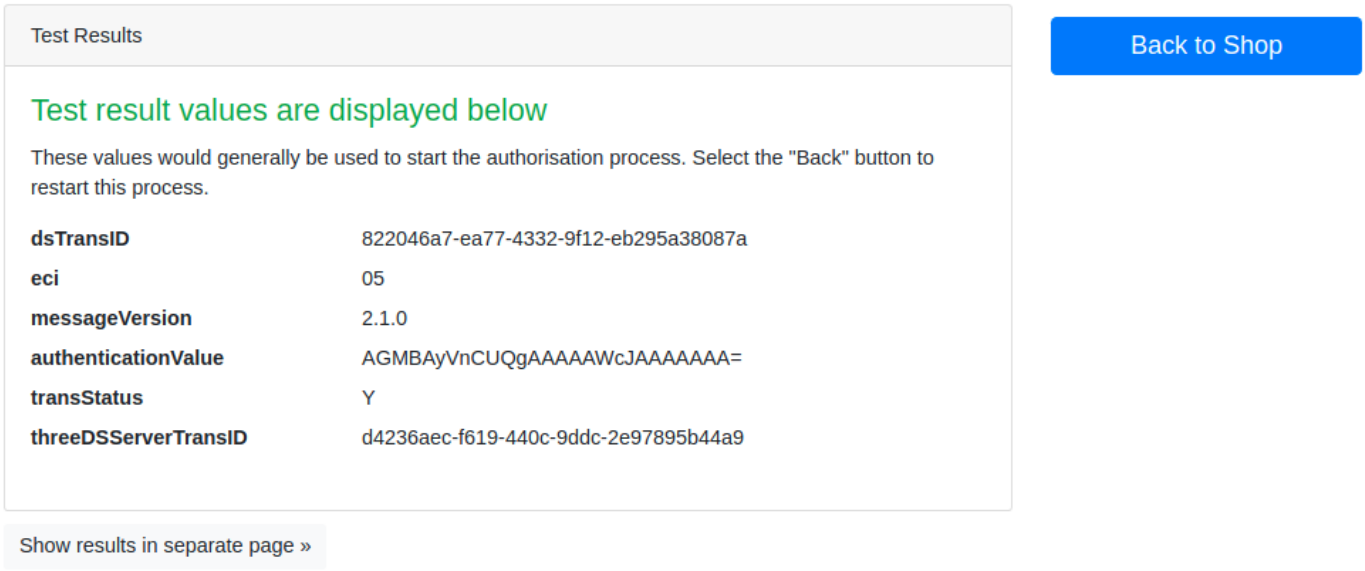
> **注目**
>
> transStatusは `Y` 、 `C` 、 `N` 、 `D` 、 `U` 、 `A` とです。
>
> - `Y` : 認証 / 口座確認に成功
> - `C` : チャレンジが必要
> - `N` : 未認証 / 口座未確認
> - `U` : 認証 / 口座確認を実行できなかった
> - `A` : 処理の試行が実施された
> - `R` : 未認証 / 口座未確認または取引拒否
>
> 詳細はAPIドキュメントを参照してください。

フリクションレス認証結果

もし `transStatus` が `C` または `D` ではない場合（つまりフリクションレスの場合）、**3ds-web-adapter**はフリクションレスフローへ移行し(ステップ 14(F)) 、 `_callbackFn("onAuthResult", data)` を呼び結果を表示します (11行目)。

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

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

`showResult(data)` 関数は `process.html` ページ内に結果を表示します:



フリクションレス認証処理はこの時点で終了になります。そして加盟店チェックアウト処理は、認証結果情報を使用して通常のオーソリ処理に移動できます。

> **情報**
>
> 認証結果は別のページにも表示されます。これは処理 3で説明されます。

チャレンジ処理の続行

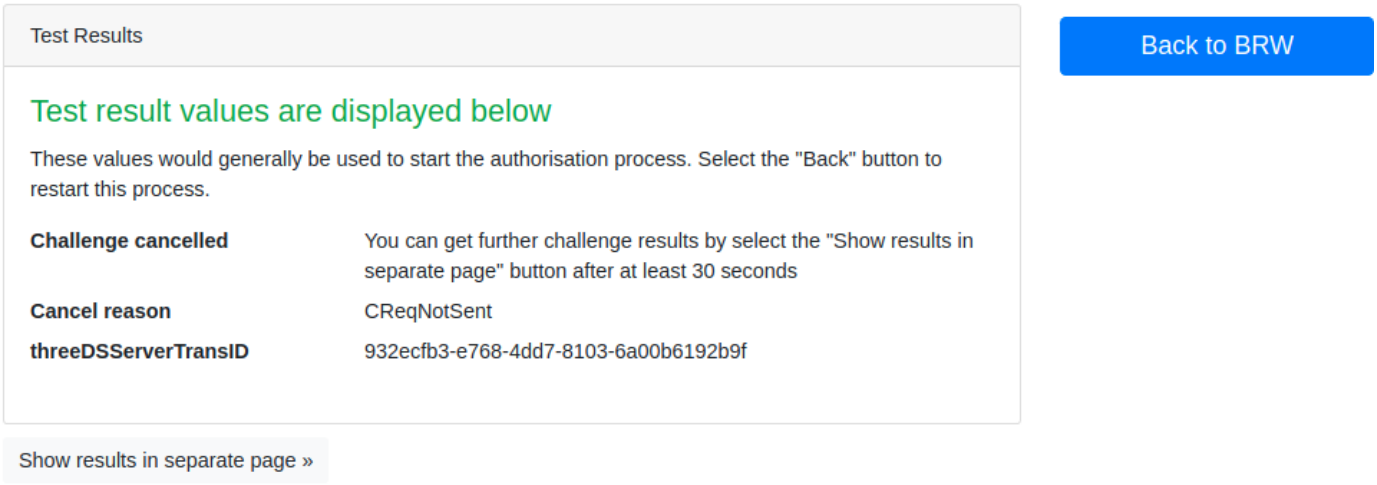
`transStatus` が `C` の場合、ACSがチャレンジを要求したことを示します。 3DS2認証を継続してチャレンジプロセスに進むか、チャレンジが不要な場合は認証プロセスを終了するかを決定するのは、3DSリクエスター次第です。このデモでは、 `options.cancelChallenge` パラメーターを使用して、チャレンジフローに関する3DSリクエスターの意思決定を示します。この機能は、BRWテストページで概説されています。

> **備考**
>
> デモの目的で、キャンセルチャレンジプロセスはフロントエンドの `javascript ( 3ds-web-adapter )` で実装されています。セキュリティ上の理由から、このプロセスは本番環境ではバックエンドで実装する必要がある場合があります。

3ds-web-adapterは、 `options.cancelChallenge` パラメーターをチェックします。 `options.cancelChallenge = true` の場合、**3ds-web-adapter**はチャレンジをキャンセルします。オプションで、指定されたキャンセルの理由に応じて、 `options.cancelReason` を設定して**ActiveServer**に送信することもできます。指定された理由は、**3ds-web-adapter**によって**バッ**

クエンドから `/auth/challenge/status` に送信されます。 **バックエンド**がこのリクエストを処理する方法を確認するには、こちらを参照してください。

チャレンジ結果のキャンセル画面は、次のスクリーンショットのようになります。



`options.cancelChallenge = true` が存在しない場合（または値が `false` に設定されている場合）、**3ds-web-adapter**は `startChallenge()` を呼び出し、チャレンジウィンドウに `src` を `challengeUrl` に設定した `iframe` を挿入します(ステップ. 14（C）)

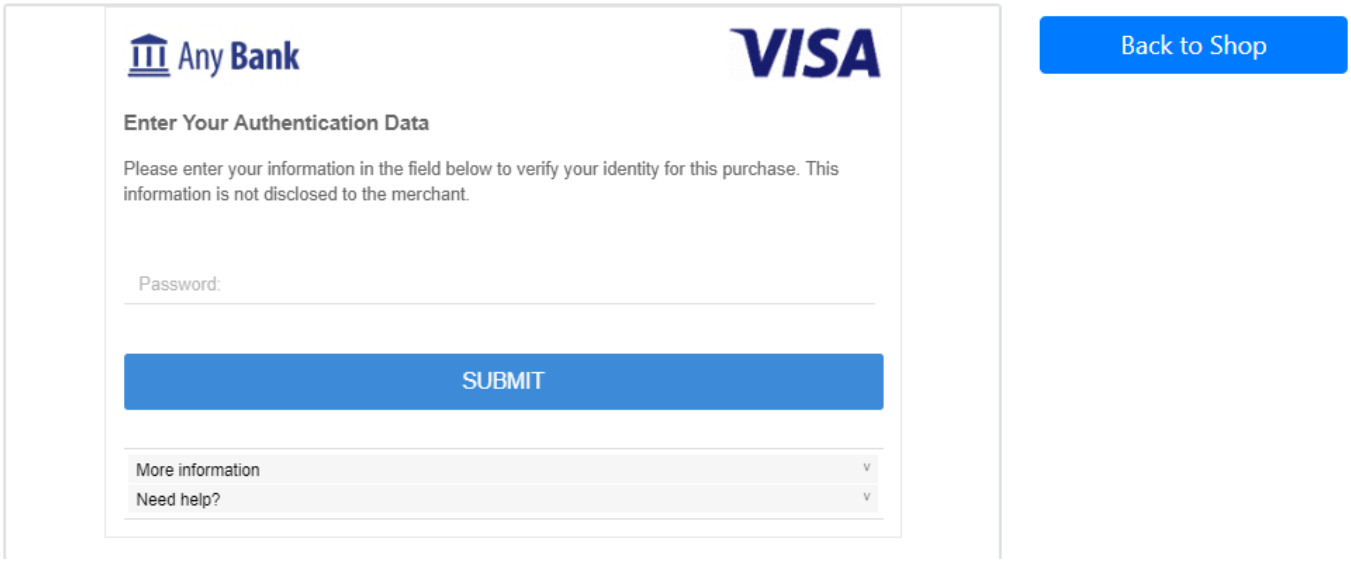

```
//3ds-web-adapter.js
function startChallenge(data) {

    if (data.challengeUrl) {
      if (_options.threeDSSessionData) {
        data.challengeUrl = appendTDSSessionData(data.challengeUrl);
    }
        challengeResultReady = false;
        _callbackFn("onChallengeStart");
    //iframeを生成
    $('<iframe src="' + data.challengeUrl
        + '" width="100%" height="100%" style="border:0" id="' + "cha_"
        + iframeId
        + '"></iframe>')
    .appendTo(iframeContainer);

    if (data.resultMonUrl) {
      console.log("Start polling for challenge result");
      _doPolling(data.resultMonUrl, getChallengeAuthResult);
    } else {
      console.log(
          "No resultMonUrl provided, challenge timout monitoring is skipped.");
    }

  } else {
    _onError({"Error": "Invalid Challenge Callback Url"});
  }


}
```

注目

チャレンジシナリオを試したい場合は、こちらのガイドに従ってください。

`iframe` の `class` 属性が `width="100%"` と `height="100%"` に設定される事が分かります。これは `iframe` がACSに提供される内容に従ってサイズ変更をする為に必要です。また、リクエスターは、ACSにチャレンジページを特定のサイズで表示するように要求することもできます。この機能の概要は、BRWテストページに記載されています。

`iframe` が設定されると、次のステップ(ステップ15(C) - 23(C)が自動的に実行されます。チャレンジ画面は、以下のスクリーンショットのようになります。



そしてカード会員はワンタイムパスワードのような認証データを入力してチャレンジフォームを送信する事が出来ます。ACSは取引とカード会員を認証してチャレンジ結果を返します。

!!! note "注釈" チャレンジiframeを設置するとき、3ds-web-adapterは `resultMonUrl` とともに `_doPolling()` 関数も追加する必要があります。これは、チャレンジプロセス時にACS側でタイムアウトが発生した場合に、認証結果が確実に取得されるようにするためです。
`_doPolling()` 関数と `resultMonUrl` の詳細については、次のセクションで説明します。

チャレンジフローが終了すると、**ActiveServer**はコールバックフォームを `iframe` に返します(ステップ23(C)。 `iframe` は、コールバックイベント `_onAuthResult` を**3DSリクエスター**バックエンドの `3ds-notify` エントリポイント(ステップ24(C))に自動的に転送します。次に、**3DSリクエスター**は、前回のブラウザー情報収集と3DSメソッドの処理と同様に `notify_3ds_events.html` ページをレンダリングします(ステップ25(C)。

チャレンジフローが終了した後、 **ActiveServer**は**3DSリクエスター**バックエンド内の `/3ds-notify` エントリーポイントを `iframe` を通して呼び出し、 `_onAuthResult` が呼ばれます。**3DSリクエスター**は以前 (ステップ 19(C))と同様に `notify_3ds_events.html` ページをレンダリングします。**3ds-web-adapter**は認証結果を取得しそれを `process.html` ページに表示する為に `_onAuthResult()` 関数を呼び出します。

情報

本番環境ではACSはカード会員から得られた情報から複雑なリスクベースの認証を実行します。同様に、認証メソッド（例えばワンタイムパスワード、生体認証など）がカード会員のイシュアーにより決定され、実行されます。

デカップルド認証処理を続行

`transStatus` が `D` の場合、カード会員とACSの間でデカップルド認証が実行されます。**ActiveServer**は、デカップルド認証が完了すると、ACSから認証結果を取得します。デカップルド認証結果を取得するために、**ActiveServer**は(ステップ13)で `transStatus = D` とともに `resultMonUrl` を提供します。 **3ds-web-adapter**が `resultMonUrl` を取得すると、`resultMonUrl` をポーリングし続けて、デカップルド認証結果の準備ができているかどうかを確認する必要があります(ステップ15(D)。

> **注釈**
>
> ポーリング間隔を決定するのは3DSリクエスター次第です。推奨される間隔は**2秒毎**になります。

`transStatus` が `D` のときに `_startDecoupledAuth()` 関数を呼び出すことによって行われる結果のポーリング。

```
//3ds-web-adapter.js
function _startDecoupledAuth(data, authReadyCallback) {
  if (data.resultMonUrl) {
    _callbackFn("onDecoupledAuthStart", data);
    _doPolling(data.resultMonUrl, authReadyCallback)
  } else {
    _onError({"Error": "Invalid Result Mon Url"});
  }
}
```

`_doPolling()` 関数はGETリクエストを `resultMonUrl` に送信して、結果の可用性を確認します((ステップ. 16(D))。 `resultMonUrl` から返された応答(Step.17(D))には、 `AuthResultNotReady` または `AuthResultReady` のいずれかの `event` が含まれています。 `AuthResultNotReady` が返されると、ポーリングプロセスは続行されます。 `AuthResultReady` が返されると、`authReadyCallback` 関数が呼び出され、**ActiveServer**から認証結果が取得されます。

```
//3ds-web-adapter.js
function _doPolling(url, authReadyCallback) {
  console.log("call mon url: ", url);
  $.get(url)
  .done(function (data) {
    console.log('returns:', data);

    if (!data.event) {
      _onError({"Error": "Invalid mon url result"});
    }

    if (data.event === "AuthResultNotReady") {
      console.log("AuthResultNotReady, retry in 2 seconds");

      setTimeout(function () {
        _doPolling(url, authReadyCallback)
      }, 2000);
    } else if (data.event === "AuthResultReady") {
      console.log('AuthResultReady');
      authReadyCallback(serverTransId, _callbackFn);
    } else {
      _onError({"Error": "Invalid mon url result event type"});
    }
  })
  .fail(function (error) {
    callbackFn("onError", error.responseJSON);
  });
}
```

処理3: 認証結果の取得

チャレンジ処理が終了した後(ステップ. 25(C))、デカップルド認証結果を取得可能な場合、(ステップ. 18(D))、もしくはフリクションレス処理結果を別ページに表示する必要がある場合、オーソリ処理に必要な最終認証結果を取得するために、認証結果を要求する必要があります。

> **⚠ なぜ別の認証結果要求が必要なのか？**
>
> 処理 2の後で利用可能な認証の結果を既に持っているのに、なぜ結果をもう一度要求する必要があるのか不思議に思うかもしれません。
>
> これは、ステップ. 13で認証結果がページに転送されているためです。認証結果ページは、チェックアウトページとは別のページとして表示されるのが一般的です。この場合、**3ds-web-adapter**は結果を3DSリクエスターまたは結果ページに転送できますが、認証結果はクライアント側コードによって再転送されるため、推奨されるデータフローではありません。一般的に安全ではないと見なされます。
>
> 3DSリクエスターのサーバー側には、常に元のソース(**ActiveServer**)から結果を取得する独自のメカニズムが必要です。3DSリクエスターのクライアント側が認証結果を提供するのではなく、3DSリクエスターのサーバー側が認証結果を取得し結果ページを提供することでデータがより安全であることが保証できます。したがって、このステップで認証結果を再要求する必要があります。

このデモでは、 `process.html` 内の処理 2の最後の `Show results in separate page` を選択して別のページに結果を表示する事も出来ます。

認証結果を取得し別ページに表示する為に、フロントエンドが必要な事は:

- `認証結果の取得` メッセージを**3DSリクエスター**に送信する(ステップ 15(F)、ステップ26(C)もしくはステップ 20(C))。
- 画面に結果を表示する (ステップ 17(F)、ステップ. 21(D)、22(D)もしくは ステップ 22(C))。

まず、 `process.html` ページ内で `serverTransId` を `sessionStorage` 内に保存し、 `result.html` ページに移動します。

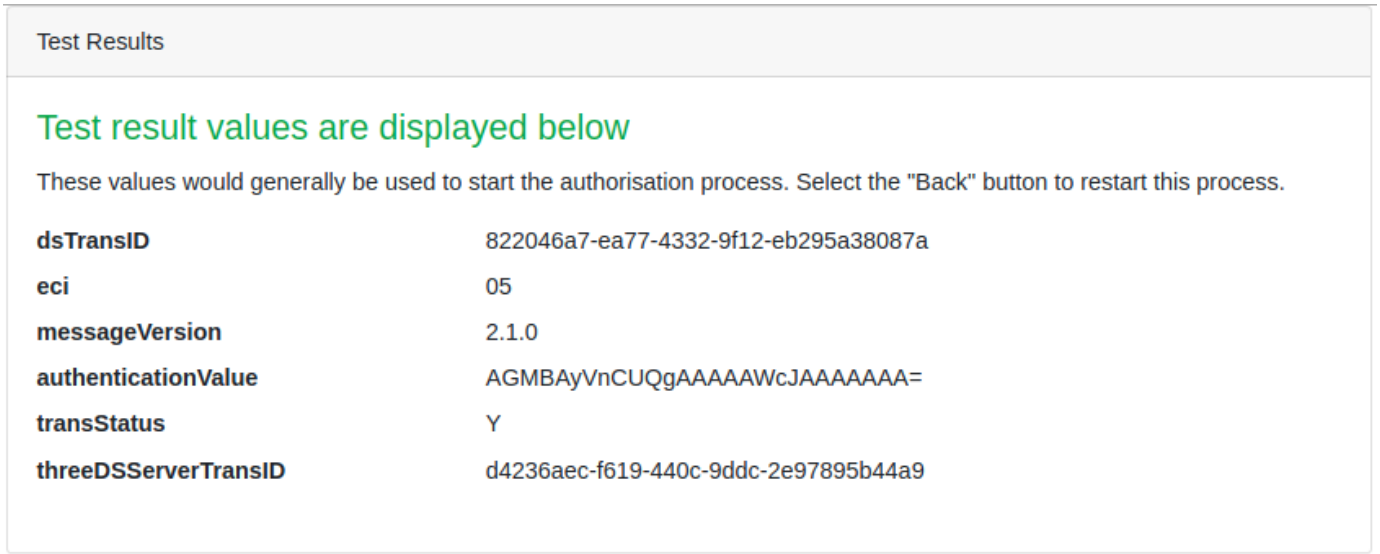
```
//process.html
function openResultInNewWindow() {
if (serverTransId) {
  var url = '/v2/result';

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

`v2/result.html` ページ内で、認証結果を取得する為に **3ds-web-adapter**内の `getBrwResult()` メソッドを呼び出します。 `getBrwResult()` メソッドは `認証結果の取得` メッセージをバックエンドへ送信します。バックエンドはこのリクエストを受信し、/api/v2/auth/brw/resultメッセー

ジを**ActiveServer**に送信します。**バックエンド**がどのようにこのリクエストを処理するか確認するには、こちらを 参照してください。コールバック関数 `showData()` はページに結果を表示します。

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

getBrwResult(serverTransId, showData);

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

新しい結果画面は以下のスクリーンショットのようになります:



✓ 成功

この文書ではフロントエンド統合について説明しました。認証が完了した後、チェックアウト処理は取引ステータス、ECIとCAVVを使用してオーソリ処理を続け、取引を完了出来ます。

次のチャプター

次へボタンを選択し3DSリクエスターの **バックエンド実装**について確認できます。