MyID

MyID REST API Browser Sample

Lutterworth Hall, St Mary's Road, Lutterworth, Leicestershire, LE17 4PS, UK
www.intercede.com | info@intercede.com | @intercedemyid | +44 (0)1455 558111

Document reference: TLK2023-01 March 2023

Copyright

Conventions Used in this Document

- Lists:
  - Numbered lists are used to show the steps involved in completing a task when the order is important
  - Bulleted lists are used when the order is unimportant or to show alternatives

- **Bold** is used for menu items and for labels.

  For example:
  - "Record a valid email address in **'From' email address**"
  - Select **Save** from the **File** menu

- *Italic* is used for emphasis and to indicate references to other sections within the current document:

  For example:
  - "Copy the file *before* starting the installation"
  - "See *Issuing a Card* for further information"

- ***Bold and italic*** are used to identify the titles of other documents.

  For example: "See the ***Release Notes*** for further information."

  Unless otherwise explicitly stated, all referenced documentation is available on the product media.

- A `fixed width` font is used where the identification of spaces is important, including filenames, example SQL queries and any entries made directly into configuration files or the database.

- **Notes** are used to provide further information, including any prerequisites or configuration additional to the standard specifications.

  For example:

  Note: This issue only occurs if updating from a previous version.

- Warnings are used to indicate where failure to follow a particular instruction may result in either loss of data or the need to manually configure elements of the system.

  For example:

Warning:	You must take a backup of your database before making any changes to it.

Contents

1 Introduction

The MyID REST SDK includes a set of JavaScript sample files that will let you authenticate from a web page to your MyID server using OAuth with PKCE mode security. Additional sample scripts show how to call a MyID REST function using the OAuth token and also how to communicate with the MyID Client Server to read a smart card.

The samples have been designed to avoid the need for any 'frameworks' such as node, REACT or even jQuery. Their clarity and simplicity illustrate the convenient flexibility offered by the MyID platform. It means that secure credential management functions can easily be slotted straight into your intranet pages by anyone with modest HTML and JavaScript coding skills.

1.1 Change history

Version	Description
TLK2023-01	First release.

2 Sample Files

The demonstration sample comprises the following files:

`demoOne.html`	*The single page for this demo
`demo.css`	The style sheet
`logo.png`	Logo graphic
`settings.js`	Demo configuration settings – you will need to edit this
`rest-helper.js`	*A set of MyID REST client utilities
`auth-helper.js`	*The entry point to MyID Authentication utilities
`auth/*.js`	A set of 5 short OAuth/PKCE utility files

To get the demo running, you will need to edit settings.js to match your MyID installation server.

When you modify or adapt the sample scripts, you will probably want to change parts of the files marked with an asterisk.

2.1 OAuth Client Scripts

The MyID REST service is protected by OAuth bearer token authentication. The Operator client includes all of the necessary components to perform the authentication within its REACT/REDUX framework. However, adding your own extensions is very challenging to achieve without making upgrades prohibitively complex. To work around this problem, you can either (a) add links to your own web pages from the OC menu, and have the OC pass its OAuth token to you or (b) perform the authentication token request directly from your own web pages.

At present, the OC enhancement to enable option (a) is still under development, but option (b) can be used with any version of MyID from 11.6 onward.

Those of you familiar with the OAuth protocol will appreciate that the cryptographic messaging needed can be a little daunting, but with the samples provided this problem is reduced to a few script references and couple function calls in your web page. In the sample *demoOne.html* there is one function call on each of the 'Sign In' and 'Read Card' buttons. The included scripts called from the *signIn()* function in *auth-helper.js* do the rest for you, subject to a couple of configuration settings described later.

2.2 Reading a smart card

The *rest-helper.js* file includes a function *getCardSN*(), which opens a connect to the client server and instructs it to prompt for a smart card and read the serial number. When this succeeds, it calls back into a function *showCardSN(data)* that you can alter to perform any operations you wish with the card information that it returns. There is a sample implementation of this function also included in the script file.

```
function getCardSN() {
	websocket = new WebSocket(settings.wsLocation);
	websocket.onopen = function (event) {
		var payload = '{"Session":"","Type":"Applet","Method":"SelectCard"}';
		websocket.send(payload);
	};

	websocket.onmessage = evt => {
		var data = JSON.parse(evt.data);
		showCardSN(data);
		websocket.close(1000);
	};

	websocket.onclose = function (event) {
		if (event.code > 1000) {
			showStatus("Unable to connect to the local device service");
		}
	};
}
```

2.3 Calling MyID REST API methods

The actual javaScript code to access the REST API is really simple once you have got the authentication token. For example:

```
function getPersonDetails(id) {
      var url = settings.apiLocation + 'people/' + id;
      fetch(url, restGET)
      .then (response => response.json())
      .then (info     => showPersonDetails(info))
      .catch(err      => showStatus(err))
}
```

Is all you need to do to retrieve the attributes of a user record. It uses the *restGET* http header variable that the authentication processor sets up to authorize the asynchronous request to the REST service. When that returns, the response is converted into easily manipulated JSON with a single call, which is then passed to a function like this that you can write to present the data on the page:

```
function showPersonDetails(info) {
	setInner("dn",      info.account.dn);
	setInner("address", info.address.line1);
	setInner("empid",   info.employeeId);
	setInner("dob",     info.personal.dateOfBirth.substr(0, 10));
}
```

Example HTML page:

```
<html>
<head>
	<link rel="stylesheet" type="text/css" href="demo.css" />
	<script
src="https://react.domain31.local/myid/OperatorClient/appSettings.js"></script>
	<script src="settings.js"   ></script>
	<script src="auth/sha256.js"></script>
	<script src="auth/pkce.js"  ></script>
	<script src="auth/cuid.js"  ></script>
	<script src="auth/oauth2.js"></script>
	<script src="auth/popup.js" ></script>
	<script src="auth-helper.js"></script>
	<script src="rest-helper.js"></script>
</head>

<body>
	<table>
	<tr><td rowspan="2"><img src="logo.png" /></td>
	<td><h1>MyID REST API Demo</h1></td><td></td>
   </tr>
	<tr>
		<td>
			<button id="logonButton" onclick="signIn();">Sign In
</button>  
			<button id="readButton"  onclick="getCardSN();" disabled>Read
Card</button>  
		</td>
		<td width="50%">
			<div style="float:right">    <span
id="status"></span>  |  
				<span id="operatorName"></span>
     </div>
		</td>
	</tr>
	</table>

	<hr><br>
	<div><em>Instructions:</em></div>
	<div>Sign in with your card and then click 'Read Card' to pick a card to
inspect</div>
	<br><hr><br>

	<h1>Card Information</h1>

	<table>
		<thead><td width="200px">Property</td> <td>Value</td></thead>
		<tr><td>Serial number</td>
			<td id="cardsn"  ></td>
			<td rowspan="10"><img id="photo" width="240px"/></td>
		</tr>
		<tr><td>FASC-N</td>                <td id="fascn">
	</td>		</tr>
		<tr><td>Card type</td>             <td id="cardtype">          </td>
	</tr>
		<tr><td>Status</td>                <td id="cardstatus">        </td>
	</tr>
		<tr><td>Profile</td>               <td id="profile">
	</td>		</tr>
		<tr><td>Issued</td>                <td id="validfrom">         </td>
	</tr>
		<tr><td>Expires</td>               <td id="validuntil">        </td>
	</tr>
		<tr><td>Owner</td>                 <td id="owner">
	</td>		</tr>
```

```
                <tr><td>DN</td>                              <td id="dn">
        </td>           </tr>
                <tr><td>Employee #</td>      <td id="empid">                      </td>
        </tr>
        </table>
</body>
</html>
```

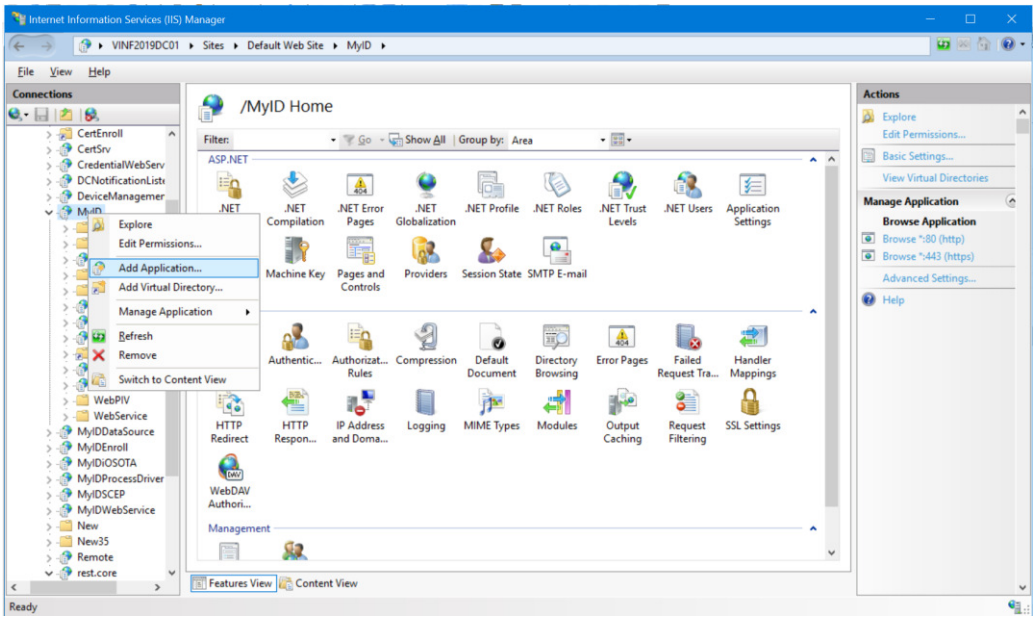
3 Installation and Configuration

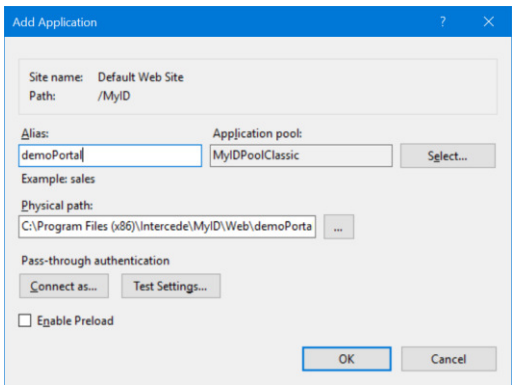
For obvious security reasons, care must be taken when installing and configuring MyID to accept any custom web pages that need to interact with the MyID Authentication Service (MAS). The strict rules that browsers and web servers apply to prevent cross-origin resource sharing (CORS) means that any of your files handling the request must be within the same web domain as MyID, and that and redirection locations are explicitly permitted in the MyID *web-auth/appsettings.json* file.

Let's assume that you are creating your own web portal to let cardholders examine the status of a smart card.

The first step it to create a location for your landing/sign-in page within the MyID website. So, in windows explorer, go to the *c:\Program Files\Intercede\MyID\web* folder and create a new folder *demoPortal*. Copy the demo script folder contents to the new folder.

Now in IIS, add this folder as a web application. This gives you full control over that site, including the ability to use different server technologies such as ASP within your portal.





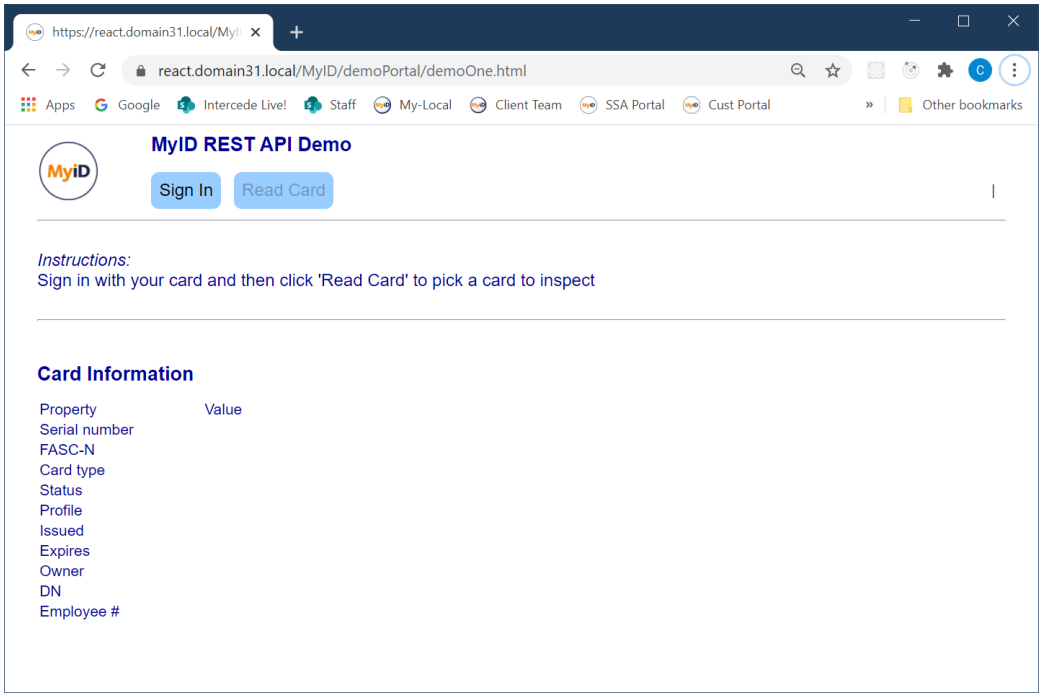
Now you need to grant your new web app permission to use the MAS. We shall declare it as a new 'Client' in the MAS *appsetting.json* file (setting your paths accordingly):

```
"Clients": [
   {
   "ClientId":            "myid.demo",
   "ClientName":          "MyID API Demo",
   "AllowedGrantTypes":   ["authorization_code"],
   "RequireClientSecret": false,
   "RequirePkce":         true,
   "AllowedScopes":       ["myid.rest.basic"],
   "AllowAccessTokensViaBrowser":    true,
   "RequireConsent":      false,
   "RedirectUris":        ["https://your-myid-server.com/MyID/demoPortal/demoOne.html"]
   },
...
```

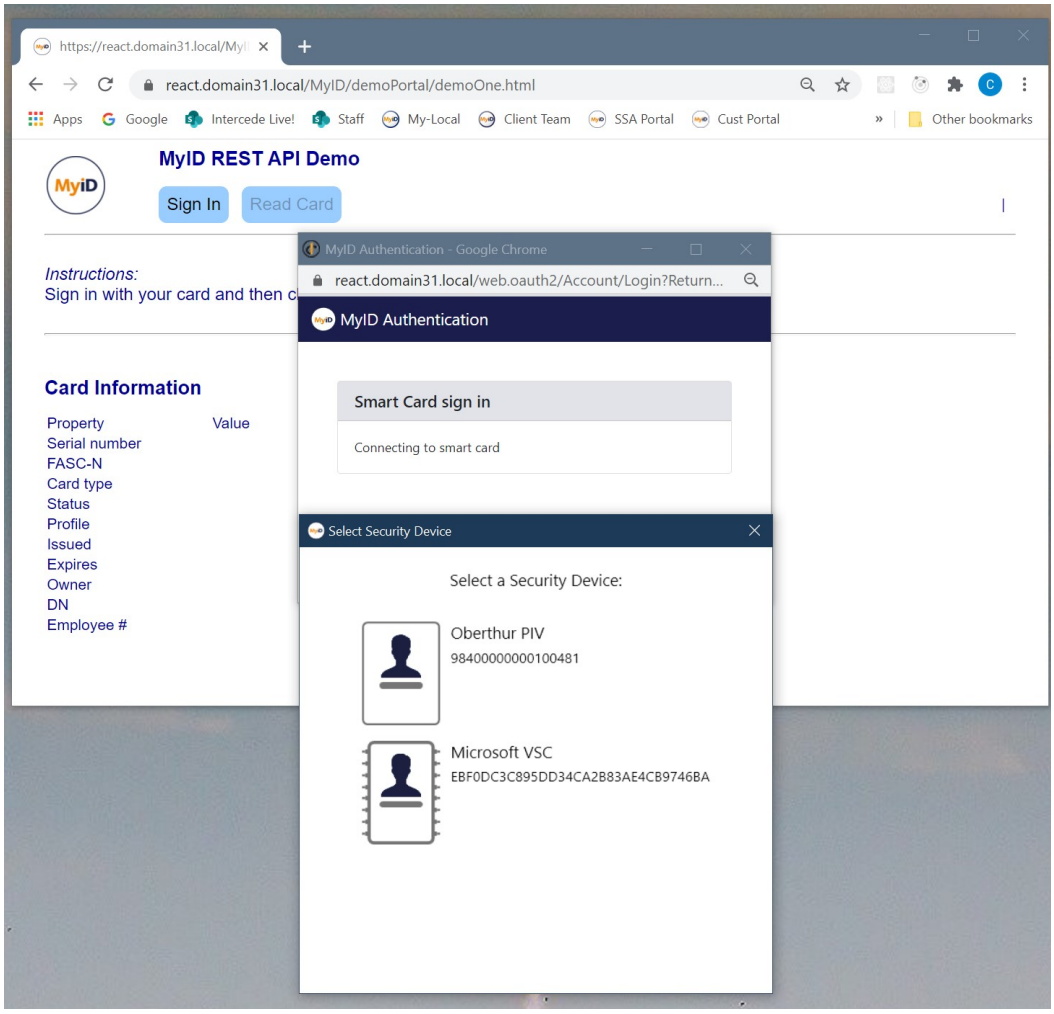
You need to restart IIS for the new client to be recognized.

4 Running the Demo

Now, assuming that the rest of your MyID client and server installations are installed and working, you should be able to point your browser at the demoOne.html web location and see the following screen:



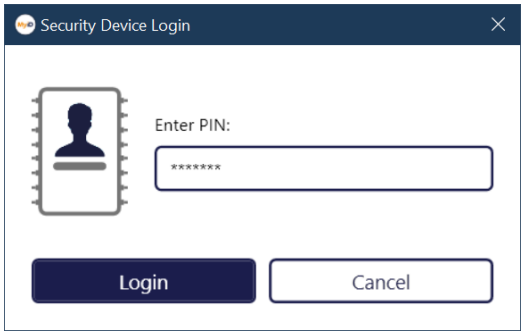
From here, click ‘Sign In’ and you should get the MyID authentication screen and a card selector displayed:

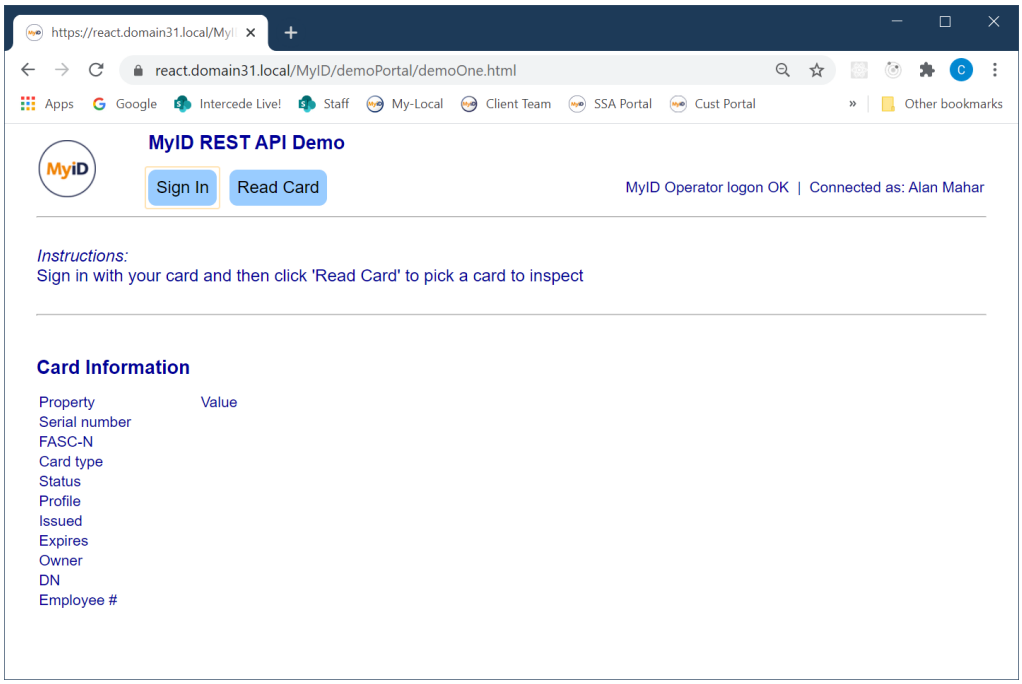


If you get an ‘Unauthorized_client’ error, check that the ClientID matches the value you set in settings.js. Also check that the RedirectUris list contains an entry for your html file location with the correct path.

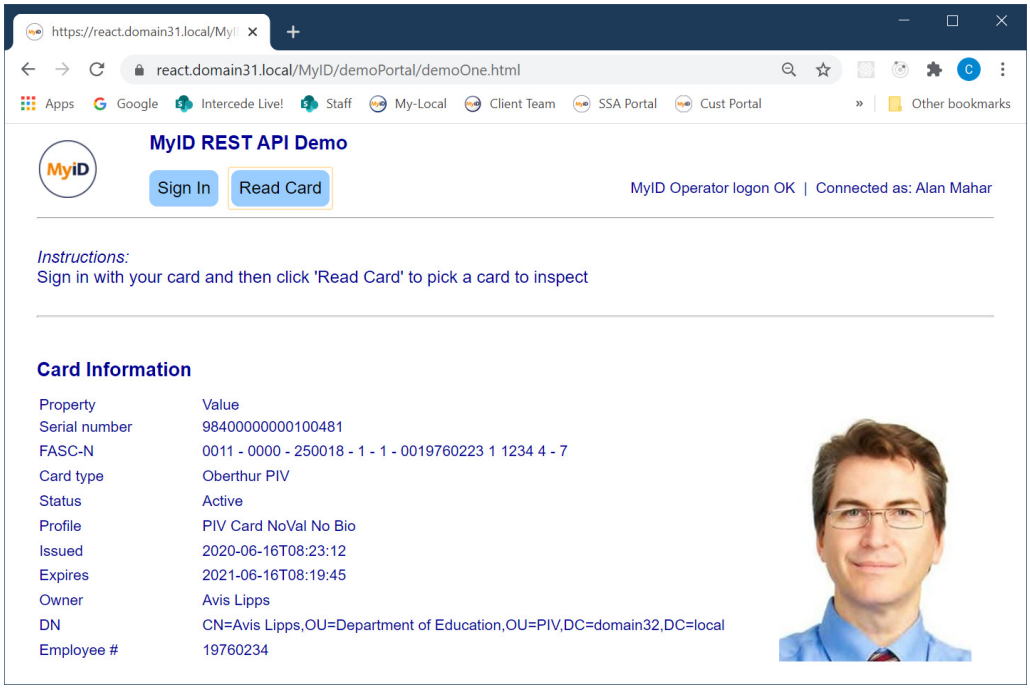
If you get an “HTTP Error 500.30 ANCM In-Process Start Failure” in the authentication window, check the *appsettings.json* file for syntax errors – including duplicate entries for ClientID, stray/missing commas etc.

Choose the card, enter the PIN and you should be logged into the MyID server:





Now click on Read Card and choose the card that you want to inspect:



The card serial number is read, and a series of REST calls are made to retrieve the card and user data from MyID.

5 Developer documentation

The MyID REST API has a full supporting set of 'swagger' documentation, including the ability to use MyID as a live test-rig for your calls.

The documentation portal is disabled by default, but can be turned on as follows:

Edit the *appSettings.json* file in you MyID server *rest.core* folder, setting these values to true:

```
"SwaggerApiDocumentation":  {
          "GenerateDocumentation":       true,
          "DocumentErrorCodes":          true,
          "AllowTestFromDocumentation":  true
},
```

Restart the MyID web service (*iisreset*). Now you can navigate to *https://<myid.server>/rest.core/swagger/index.html* and examine the full MyID REST API documentation.

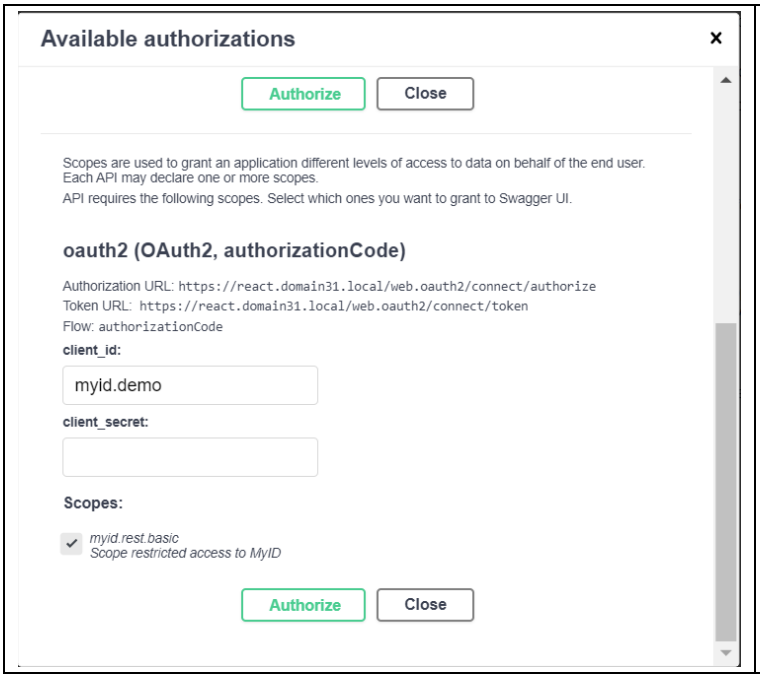
5.1 Exercising the API

To use the documentation portal as a test rig, you need to enable it for authentication as follows;

Add the documentation redirect page to the safe redirect URI locations for the myid.demo client ID that you defined earlier in the MyID server *web-auth* folder:

```
"RedirectUris":   [
                   "https://<myid.server>/MyID/demoPortal/demoOne.html",
                   "https://<myid.server>/rest.core/swagger/oauth2-redirect.html"
                  ]
```

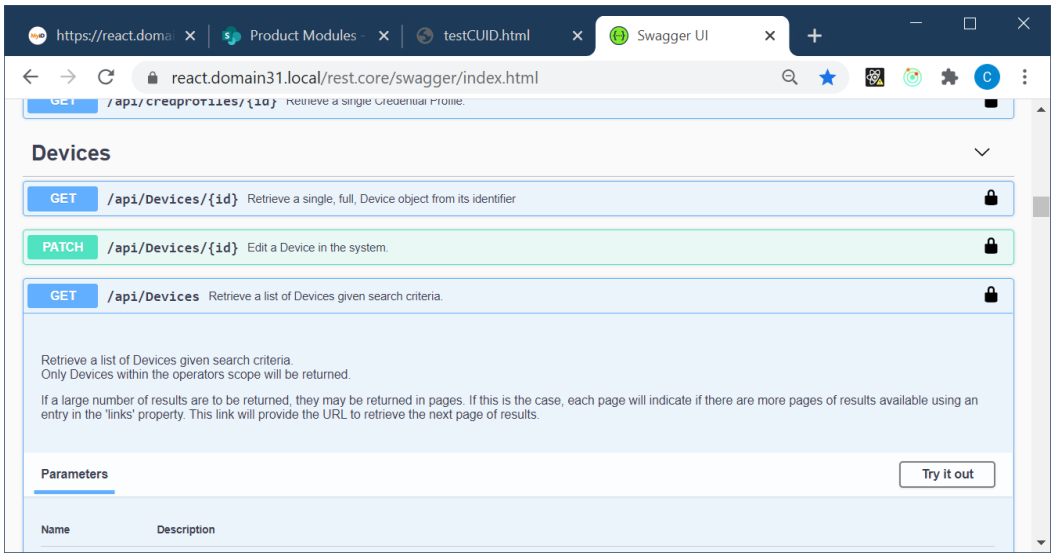
Now you can click the Authorize button on the main documentation page. Scroll down to the second section (authorizationCode):



Enter the client ID you set earlier and check the 'scopes' box. You can provide a unique *client_secret*, but this is an optional security enhancement.

Click *Authorize* and you will be prompted through a MyID authentication as usual.

Once authorized, you can use the 'Try it out' buttons for each end point in the documentation



The use of this feature is very intuitive. After clicking the Try It Out button, just fill in the values you want to send and click the Execute bar at the bottom of that section. The results are shown inline.

If you are working in a more advanced development environment such as Visual Studio, you can click the *swagger.json* link at the very top of the page (below the MyID Core API header. This returns an OpenAPI JSON definition of the MyID REST API, which you can then import into your IDE to automatically create supporting object representations in your chosen programming language.

6 Architecture

Architecture