# SSO with Google

## Part 1: How to implement Shopper Login and API Access Service (SLAS) in Salesforce Commerce Cloud (SSO with Google)

***In Part 1, we will cover the SLAS Admin configuration stage. We will configure the Client and the IDP in this section.***

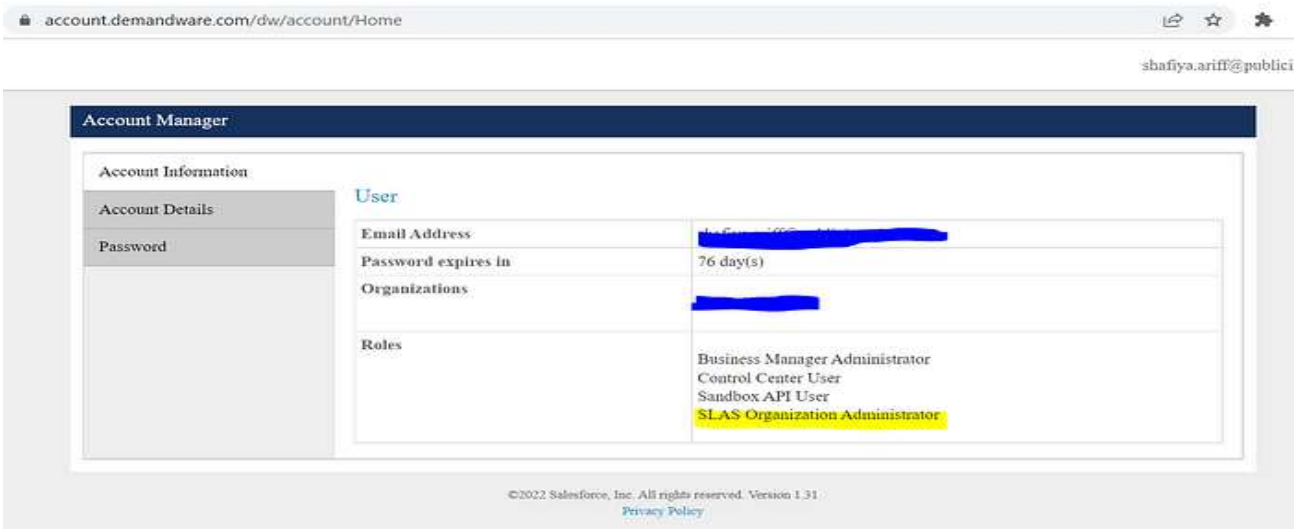
***In Part 2, we will implement SLAS in the RefArch Storefront.***

What is SLAS?

The Shopper Login and API Access Service (SLAS) enables secure access to the Shopper APIs of the B2C Commerce API and the OCAPI. It is a highly scalable solution for authentication and authorization that enables customers to level up security and trust in their commerce applications.

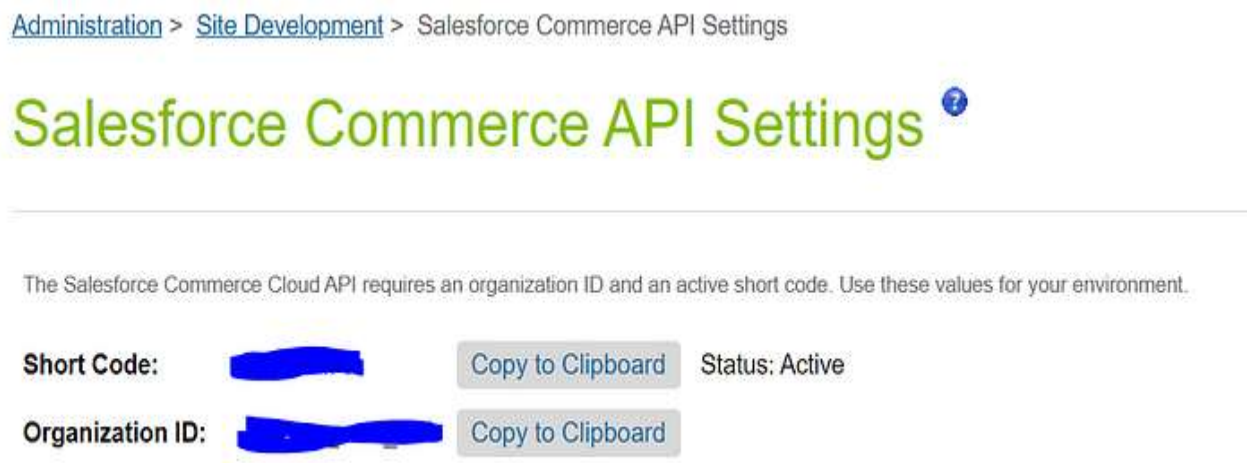
In this article, we will implement **Single Sign-On** with **Google** as an Identity Provider. This use case describes what happens when a shopper logs in using a third-party identity provider (Google)before adding a product to their basket.

Before we get started, we need to know about the **Shopper Login (SLAS) Admin**. The SLAS Admin API is used for administration tasks, not for requesting access to the Shopper APIs. It is used for setting up clients and IDPs. To use the SLAS Admin API and its associated UI, your Account Manager account must have the **SLAS Organization Administrator** role. If you don't have admin rights, connect with the Administrator of your team and get the role assigned.

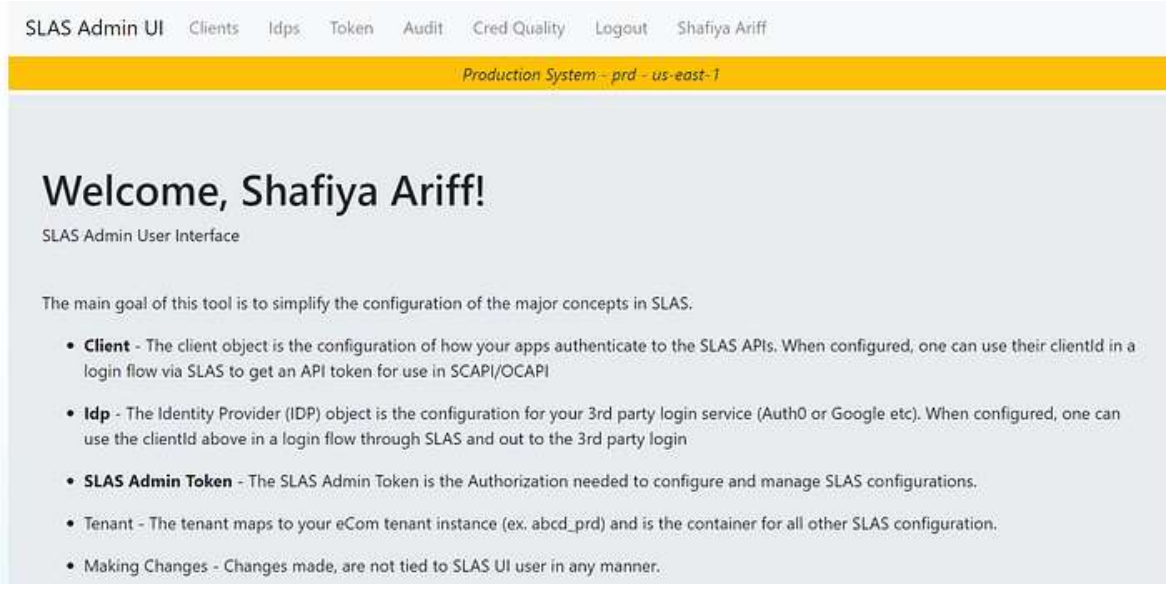


Once the role is assigned, you can access the SLAS Admin UI. The URL — **Error! Hyperlink reference not valid.**n/v1/ui

The '**shortCode**' can be found in your Business Manager. Go to **Administration > Site Development> Salesforce Commerce API Settings.**

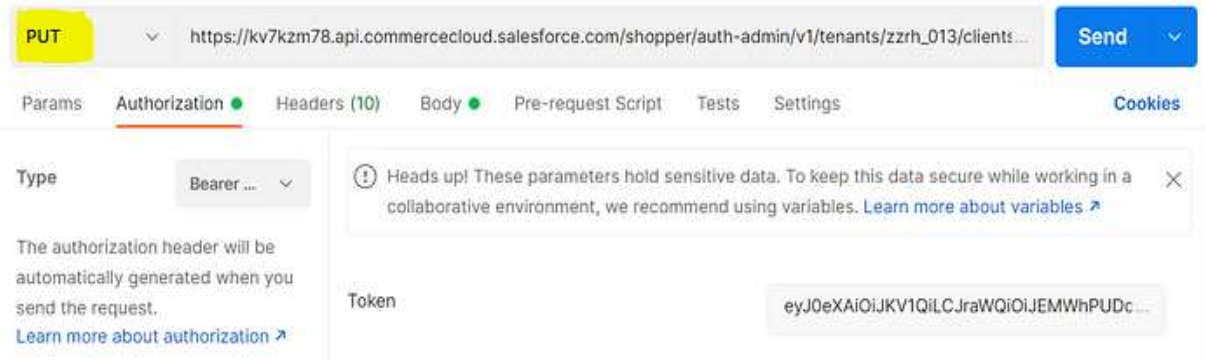



Replace **Error! Hyperlink reference not valid.** with the Short Code from your BM and you have your Admin UI!

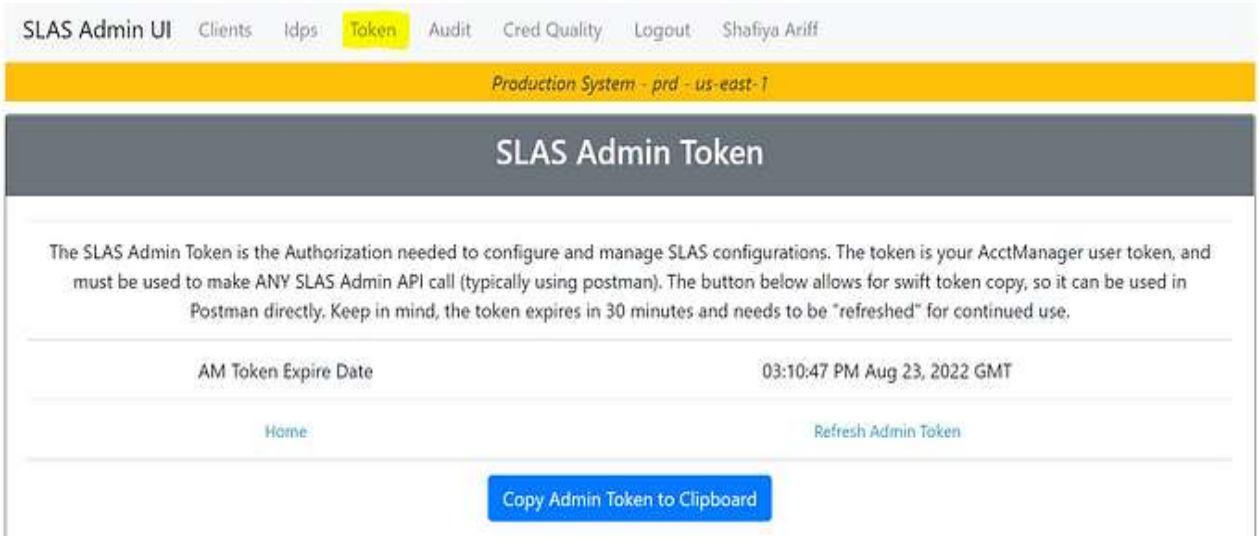



Now we need to create a Client. You can use Postman to create a private SLAS client.

The request URL — https://{shortCode}.api.commercecloud.salesforce.com/shopper/auth-admin/v1/tenants/{tenant_id}/clients/{client_id}



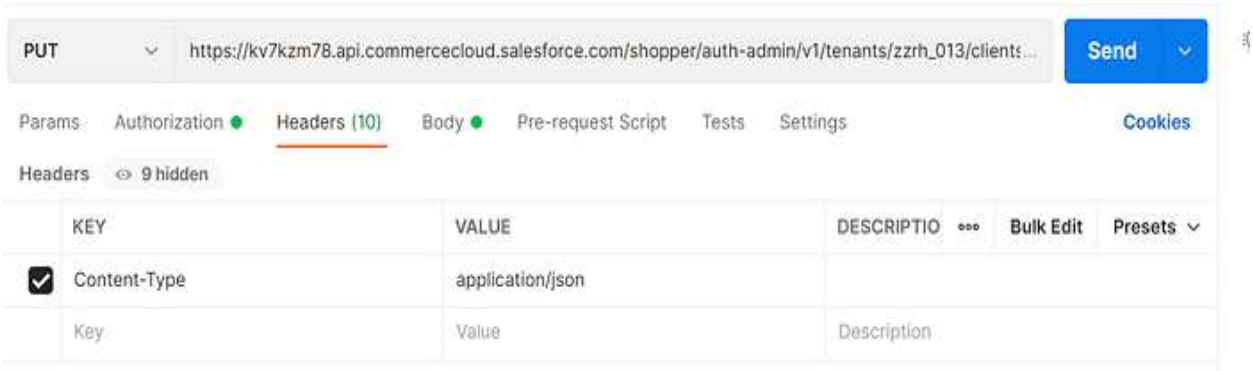

The tenant_id can be your sandbox number and the client_id can be your sandbox UUID.To get the Bearer Token for Authorization, click on the '**Token**' tab in the SLAS Admin UI. Generate a token and paste it as shown in the screenshot above.




The request body –

```
{
  "clientId": "<CLIENT_ID>",   //replace with your client id (slas client id)
  "secret": "<CLIENT_SECRET>", //you will get secret when the client is created at first time in slas
  "isPrivateClient": true,
  "name": "Test Client",
  "channels":
    ["RefArch",
    "RefArchGlobal",           //add channels as per your requirement
    "SiteGenesis"],
  "scopes": [
    "sfcc.shopper-baskets-orders.rw",
    "sfcc.shopper-categories",
    "sfcc.shopper-customers.login",
    "sfcc.shopper-customers.register",
    "sfcc.shopper-gift-certificates",
    "sfcc.shopper-myaccount.rw",
    "sfcc.shopper-product-search",
    "sfcc.shopper-productlists",
    "sfcc.shopper-products",
    "sfcc.shopper-promotions",
    "sfcc.shopper-stores"     ],
"redirectUri": ["https://abcd-015.sandbox.us01.dx.commercecloud.salesforce.com/on/demandware.store/Sites-RefArch-Site/en_US/Account-Show"] }
```

Header:



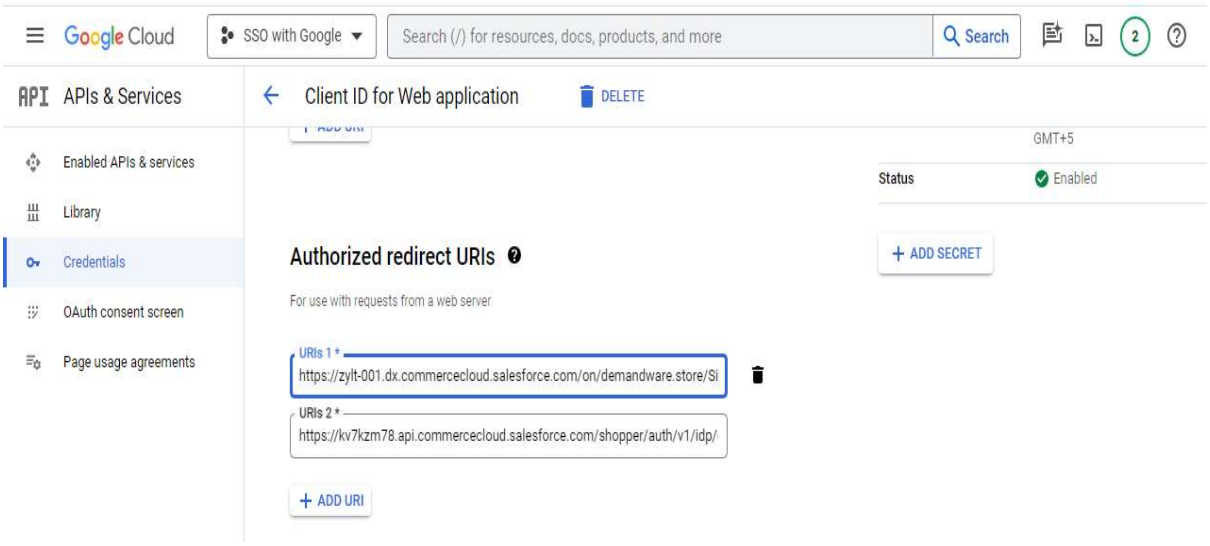

Notice that in the redirectUri I have given 'Account-Show'. This is for redirecting to Account Dashboard after google sign in.

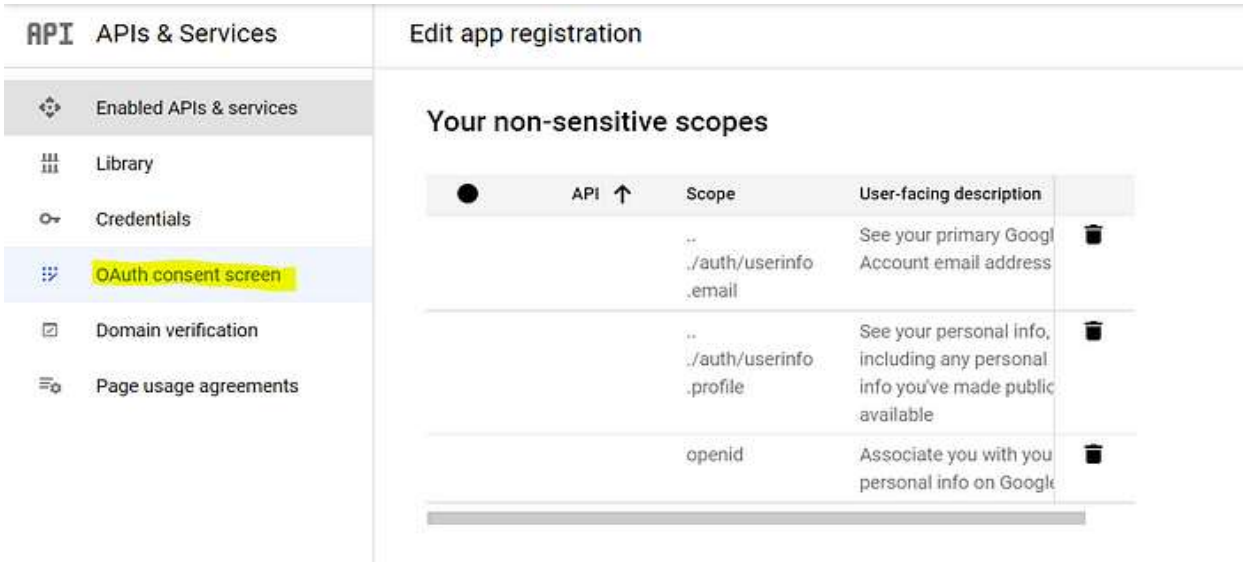
After clicking on send, if you get a 200 response back, you will be able to see that the Client has been created in the Admin UI. Now let's move on to creating an IDP!

For creating an IDP, we need to create a google oauth Client ID. If you don't have one already, you can follow this guide to create it. There are three things to take care of while creating the Client ID:

1. Generate a Client Secret and keep it secure for future use.
2. Make sure you are entering the same redirect URI as the SLAS Client URI under 'Authorized redirect URIs'.

3. You need to provide the scopes. This will later help us in retrieving the user information after signing in.






You can now go back to the SLAS Admin UI and click on idps and select '**Add Idp**'


You can enter the same details as above. Make sure you are adding the Google oauth Client ID and Client Secret which you created previously in Idp Client Id and Secret respectively. Idp Name **must** be 'google'.

We are done with the Admin configuration of SLAS!

Click here for Part 2 where we will start implementing it in our RefArch Storefront.

Thanks for reading :)

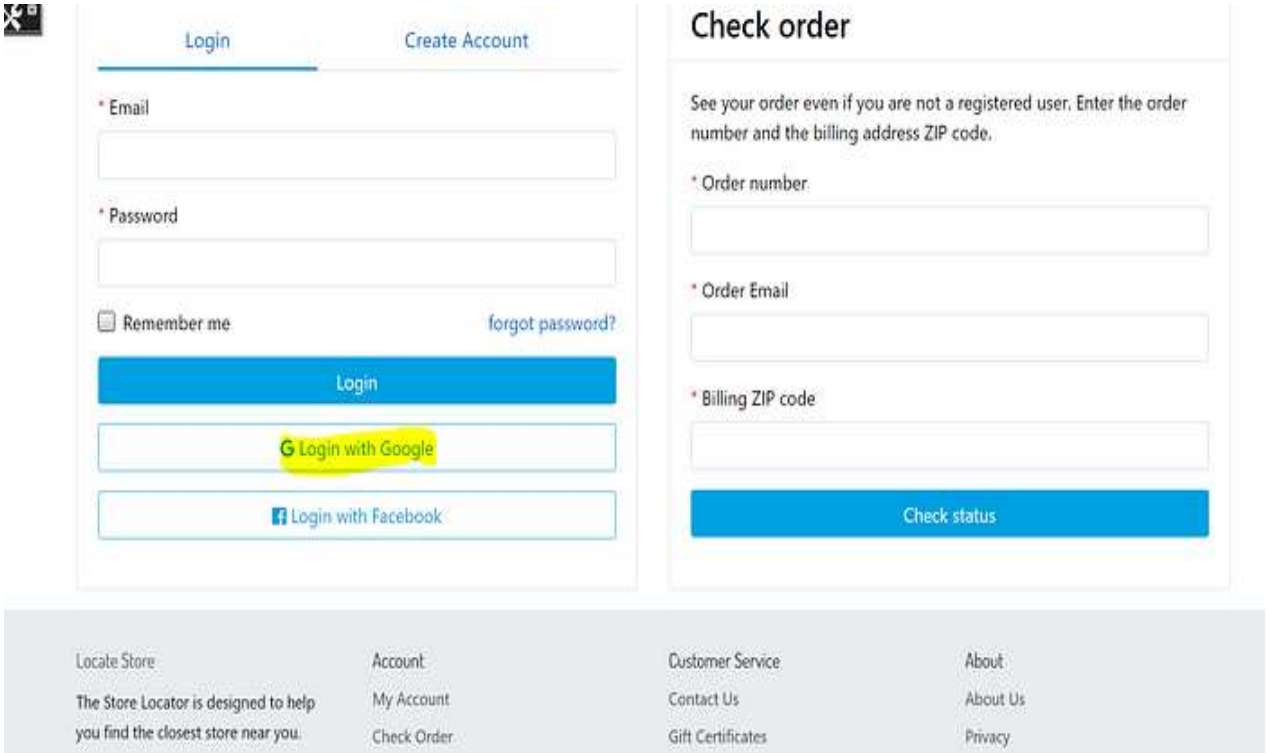
# Part 2: How to implement Shopper Login and API Access Service (SLAS) in Salesforce Commerce Cloud (SSO with Google)

In Part 1, we configured the SLAS Admin. In Part 2, we will implement it in our RefArch Storefront!

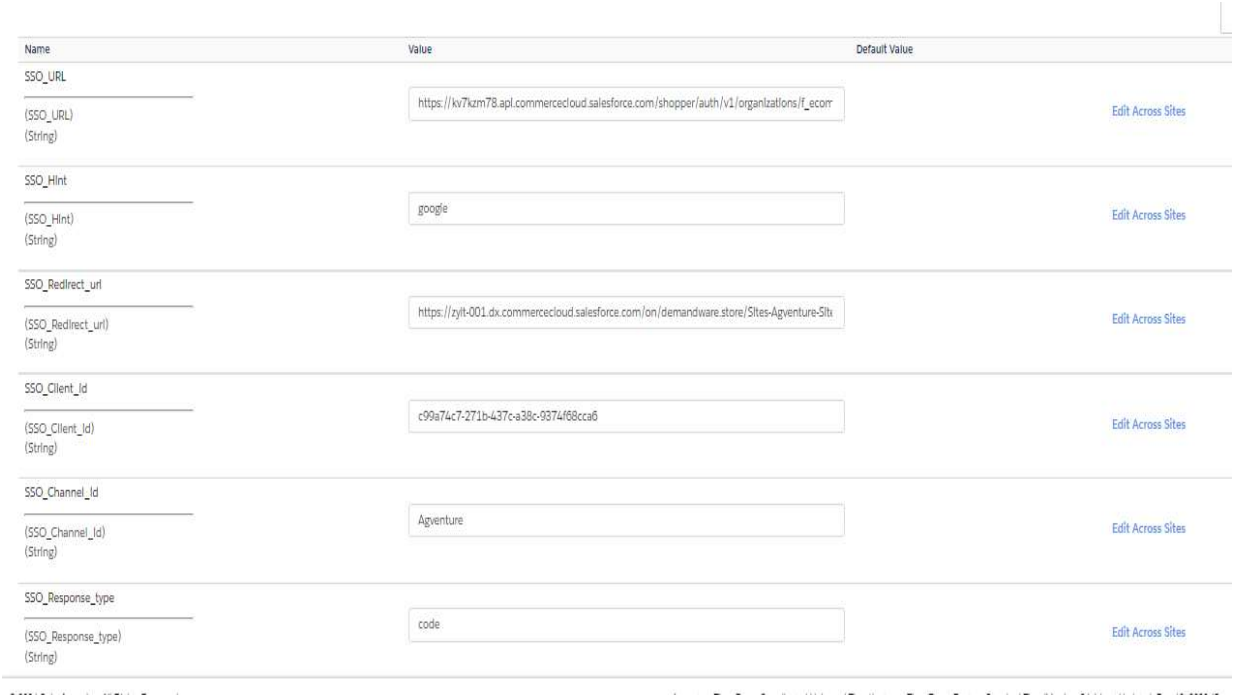
Let's understand the flow first.

1. Call the **/authorize** endpoint of the SLAS API. This API will redirect to the IDPs login page.
2. Once we enter our credentials and give consent, the API responds with an **authorization code** and redirects to the shopping website.
3. Now using this authorization code, we can call the **/token** endpoint of the SLAS API. This API responds with the access token, refresh token, customer Id, USID and IDP access token.

Let's start with the implementation!



The first step is to call the **/authorize** endpoint of the SLAS API when we click on the 'Login with Google' button. This endpoint requires a couple of query parameters for it to work. We can add these parameters as custom preferences in our BM.




Enter your Client ID and redirect URI which you used while setting up the Client in the SLAS Admin. Your SSO_URL would be —
https://{shortCode}.api.commercecloud.salesforce.com/shopper/auth/v1/organizations/{organization_id}/oauth2

You can find you {organization_id} in the same page that you found your sandbox {shortCode}.

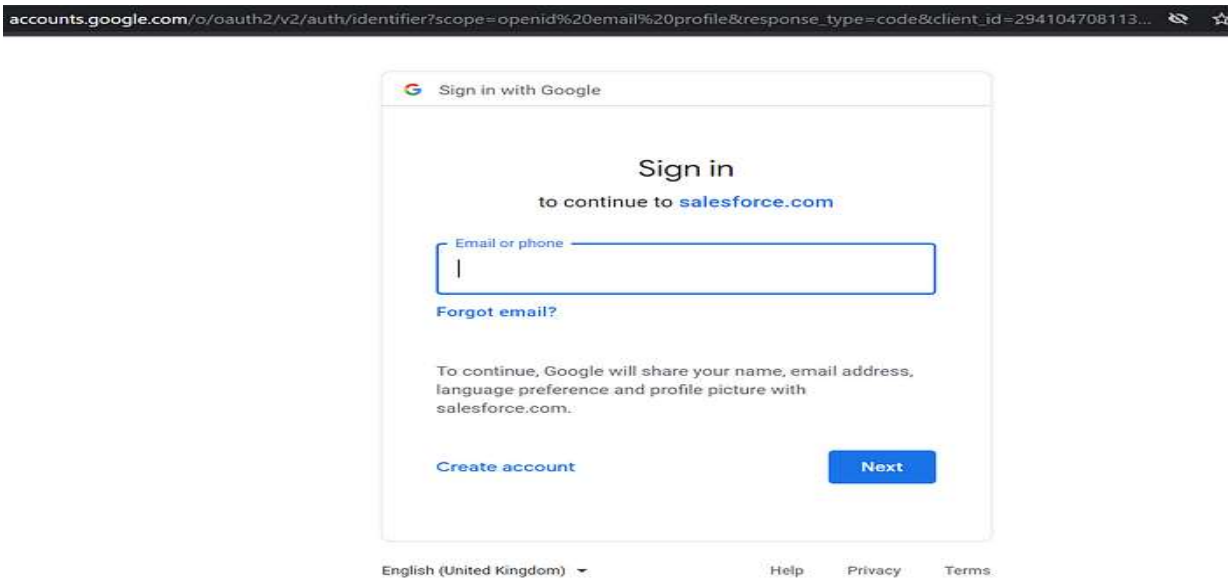
We can add some new code along with the old one in Login.js inside 'Login-Show' function. This will help in sending our SLAS /**authorize** endpoint as HREF to oauth.isml. (oauth.isml is rendering the 'Login with Google' button)

```
/**
 * Login-Show : This endpoint is called to load the login page
 * @name Base/Login-Show
 * @function
 * @memberof Login
 * @param {middleware} - consentTracking.consent
 * @param {middleware} - server.middleware.https
 * @param {middleware} - csrfProtection.generateToken
 * @param {renders} - isml
 * @param {serverfunction} - get
 */
server.get(
    'Show',
    consentTracking.consent,
    server.middleware.https,
    csrfProtection.generateToken,
    function (req, res, next) {
        var Site = require('dw/system/Site');

        var url = Site.current.getCustomPreferenceValue('SSO_URL');
        var uri = Site.current.getCustomPreferenceValue('SSO_redirect_uri');
        var client_id = Site.current.getCustomPreferenceValue('SSO_client_id');
        var channel_id = Site.current.getCustomPreferenceValue('SSO_channel_id');
        var hint = Site.current.getCustomPreferenceValue('SSO_hint');
        var grant = 'authorization_code'
        var response_type = Site.current.getCustomPreferenceValue('SSO_response_type');

        res.render('/account/login', {
            url: url+'/authorize?client_id='
                +client_id+'&redirect_uri='
                +uri+'&hint='
                +hint+'&response_type='
                +response_type+'&channel_id='
                +channel_id,
        });

        next();
    }
);
```

Now in oauth.isml, you can use ‘pdict.url’ to use as an href.

```
<form action="" class="login-oauth" method="post" name="login-oauth-form">

    <div class="form-group">
        <a href="${pdict.url}" class="btn btn-block btn-outline-primary oauth-google"
            role="button" aria-pressed="true">
            <i class="fa fa-google" aria-hidden="true"></i>
            ${Resource.msg('button.text.oauth.google', 'login', null)}
        </a>
    </div>
    <div class="form-group">
        <a href="${URLUtils.https('Login-OAuthLogin', 'oauthProvider', 'Facebook', 'oauthLoginTargetEndPoint', pdict.oAuthReentryEndpoint)}" clas
            role="button" aria-pressed="true">
                <i class="fa fa-facebook-official" aria-hidden="true"></i>
                ${Resource.msg('button.text.oauth.facebook', 'login', null)}
        </a>
    </div>
</form>
```

If all is configured well and you click on the 'Login with Google', you will be taken to the Google Sign In page!




Enter your Google credentials and sign in. If successfully signed it, it will return an authorization code appended to our redirect URL. Since we are redirecting to our Account.js controller (Account-Show), we will write our code there to retrieve this **auth_code** and generate an access token. But before that, we need to create new service with the SLAS '**/token**' endpoint API. Add the following new service in services.xml for establishing a connection with the API.

```
<?xml version="1.0" encoding="UTF-8"?>
<services xmlns="http://www.demandware.com/xml/impex/services/2014-09-26">
    <service-credential service-credential-id="http.mysite.sso.cred">
        <url>https://{shortCode}.api.commercecloud.salesforce.com/shopper/auth/v1/organizations/{organization_id}/oauth2</url>
        <user-id>{SLAS_CLIENT_ID}</user-id>
        <password encrypted="true" encryption-type="common.export">{SLAS_CLIENT_SECRET}</password>
    </service-credential>
    <service-profile service-profile-id="SSO_Service">
        <timeout-millis>1000000</timeout-millis>
        <rate-limit-enabled>false</rate-limit-enabled>
        <rate-limit-calls>0</rate-limit-calls>
        <rate-limit-millis>0</rate-limit-millis>
        <cb-enabled>false</cb-enabled>
        <cb-calls>0</cb-calls>
        <cb-millis>0</cb-millis>
    </service-profile>
    <service service-id="http.sso.token">
        <service-type>HTTPForm</service-type>
        <enabled>true</enabled>
        <log-prefix>SSO</log-prefix>
        <comm-log-enabled>true</comm-log-enabled>
        <force-prd-enabled>true</force-prd-enabled>
        <mock-mode-enabled>false</mock-mode-enabled>
        <profile-id>SSO_Service</profile-id>
        <credential-id>http.mysite.sso.cred</credential-id>
    </service>
</services>
```

Now create a new Service file — '**SSOServices.js**' from where we can make the service call. The

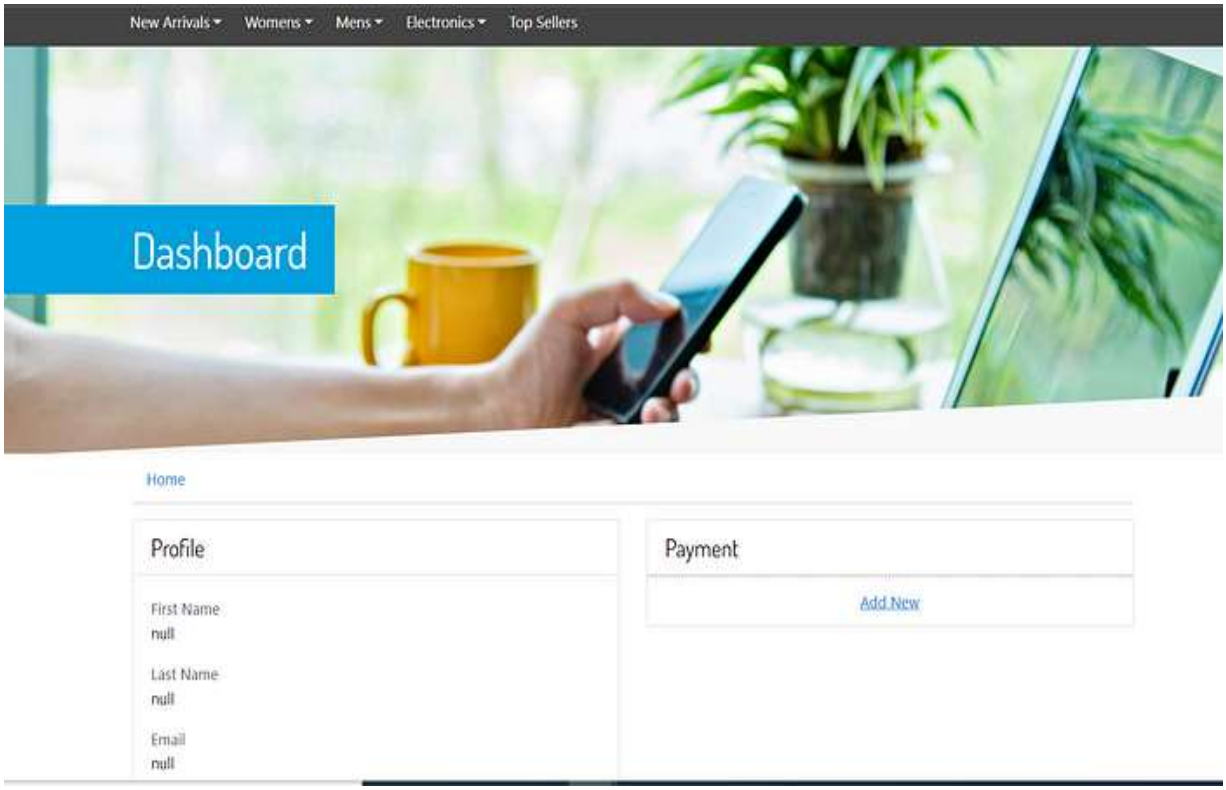
Authorization header should be a Base64 encoded version of the following string: <clientID> :<clientSecret>

```
var LocalServiceRegistry = require('dw/svc/LocalServiceRegistry');
var Logger = require('dw/system/Logger');
var Site = require('dw/system/Site');

/**
 * SSO
 * @returns {Object} SSO Service
 */
function initializeService() {
    return LocalServiceRegistry.createService('http.sso.token', {
        createRequest: function(service, args) {
            service.setURL(service.getURL() + '/token');
        },
        parseResponse: function (response) {
            return response;
        },
        filterLogMessage: function (msg) {
            return msg;
        },
    });
}


function getSSOAccessToken(auth_code, grant_type, refresh_token) {
    var authResult;
    var service = initializeService();
    service.setRequestMethod('POST');
    service.addHeader('Content-Type','application/x-www-form-urlencoded');
    service.addHeader('Authorization', 'BASIC <clientID>:<clientSecret>');
    service.addParam('client_id',Site.current.getCustomPreferenceValue('SSO_client_id'));
    service.addParam('redirect_uri',Site.current.getCustomPreferenceValue('SSO_redirect_uri'));
    service.addParam('channel_id',Site.current.getCustomPreferenceValue('SSO_channel_id'));
    service.addParam('code',auth_code);
    service.addParam('grant_type', grant_type);
    var responseObject;
    try {
        responseObject = service.call();
        if(responseObject.object &&
            responseObject.object.client &&
            responseObject.object.client.text) {
                authResult = JSON.parse(responseObject.object.client.text);
            }
    } catch (e) {
        Logger.error('Exception : ' + e);
    }

    return authResult;
}

module.exports = {
    getSSOAccessToken: getSSOAccessToken,
};
```

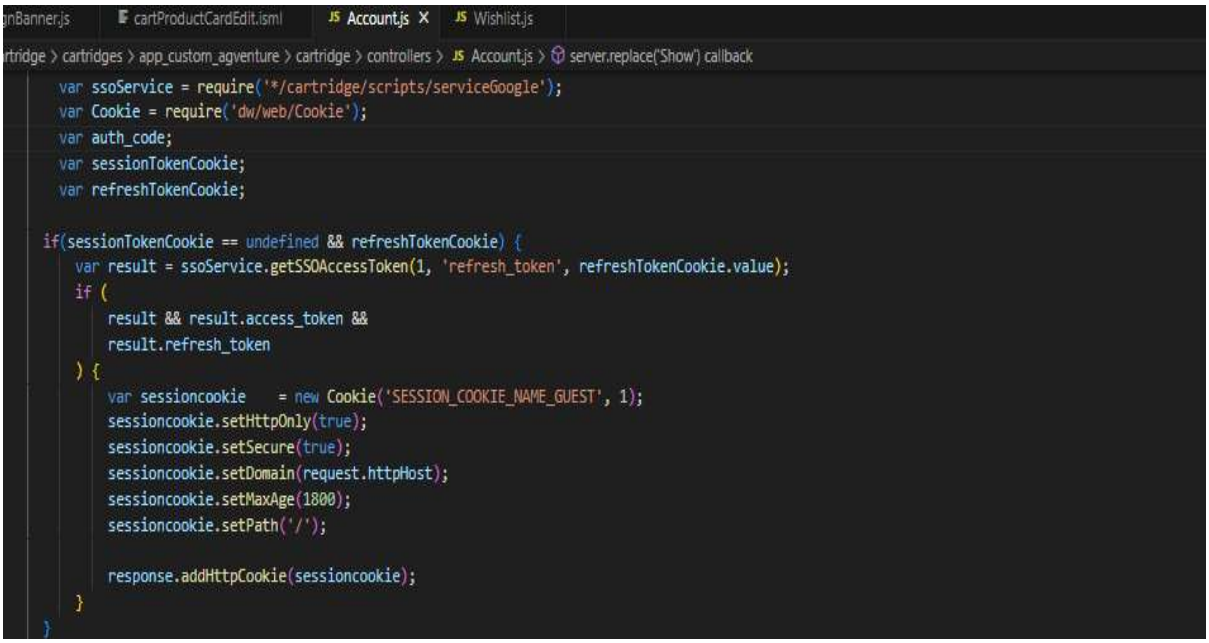
Now finally, we will call getSSOAccessToken () from Account.js file. It will return access token, refresh token, customer Id, USID and IDP access token.

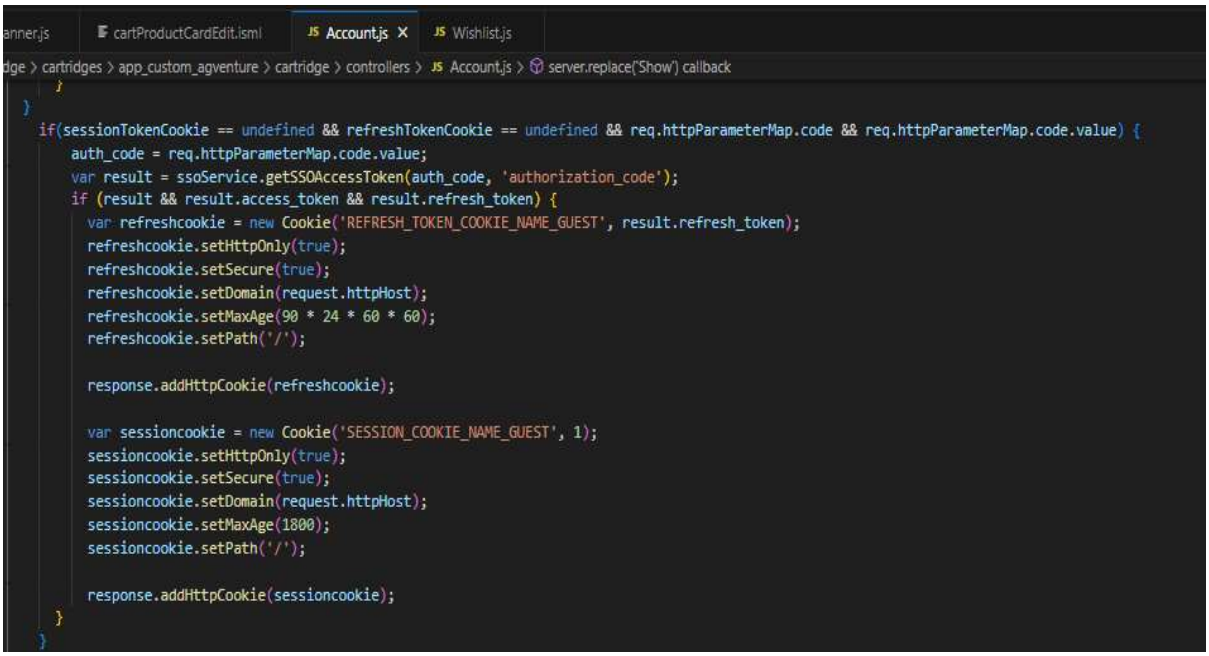
```
**
 * Account-Show : The Account-Show endpoint will render the shopper's account page. Once a shop
 * @name Base/Account-Show
 * @function
 * @memberof Account
 * @param {middleware} - server.middleware.https
 * @param {middleware} - consentTracking.consent
 * @param {category} - senstive
 * @param {renders} - isml
 * @param {serverfunction} - get
 */
server.get(
    'Show',
    server.middleware.https,
    consentTracking.consent,
    function (req, res, next) {
      var ssoService = require('../services/SSOServices');
      var auth_code;
      if(req.httpParameterMap.code && req.httpParameterMap.code.value) {
          auth_code = req.httpParameterMap.code.value;
          var result = ssoService.getSSOAccessToken(auth_code, 'authorization_code');
      }
    }
);
```

And, we are directed to the Account Dashboard!

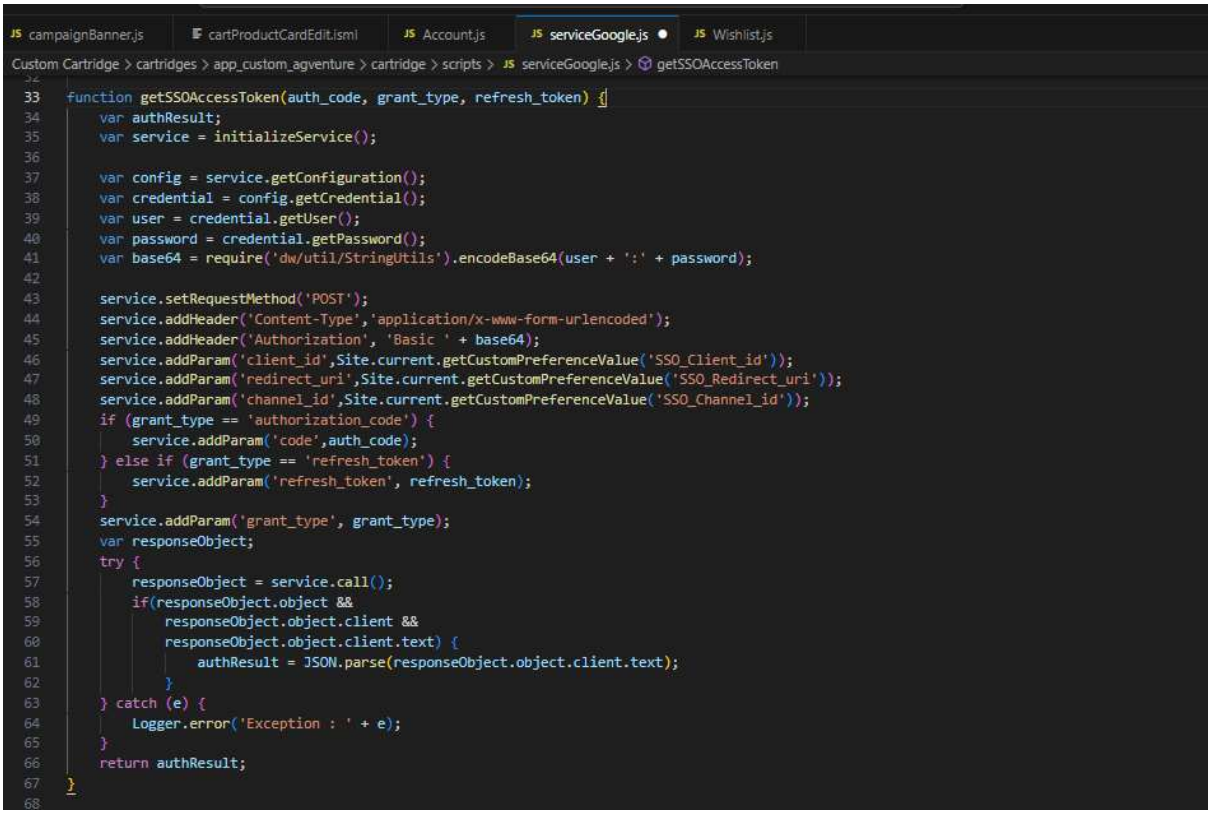


We will take care of the **null** details in a while. But before that we need to take care of the **access** and **refresh** tokens. Access tokens have an expiry time of 30 minutes whereas refresh token can be used for 90 days. We will add the refresh token in a cookie and use it to generate a new access token every time it expires. We will make the changes in the Account.js controller again.




The expiry time of the refresh token cookie has been kept as 90 days and 30 minutes for the access token. If the access token expires, it will again send a request to the '/token' API. But this time, the **grant_type** will be 'refresh_token' instead of 'authorization_code'. A quick few changes in SSOServices-getSSOAccessToken .



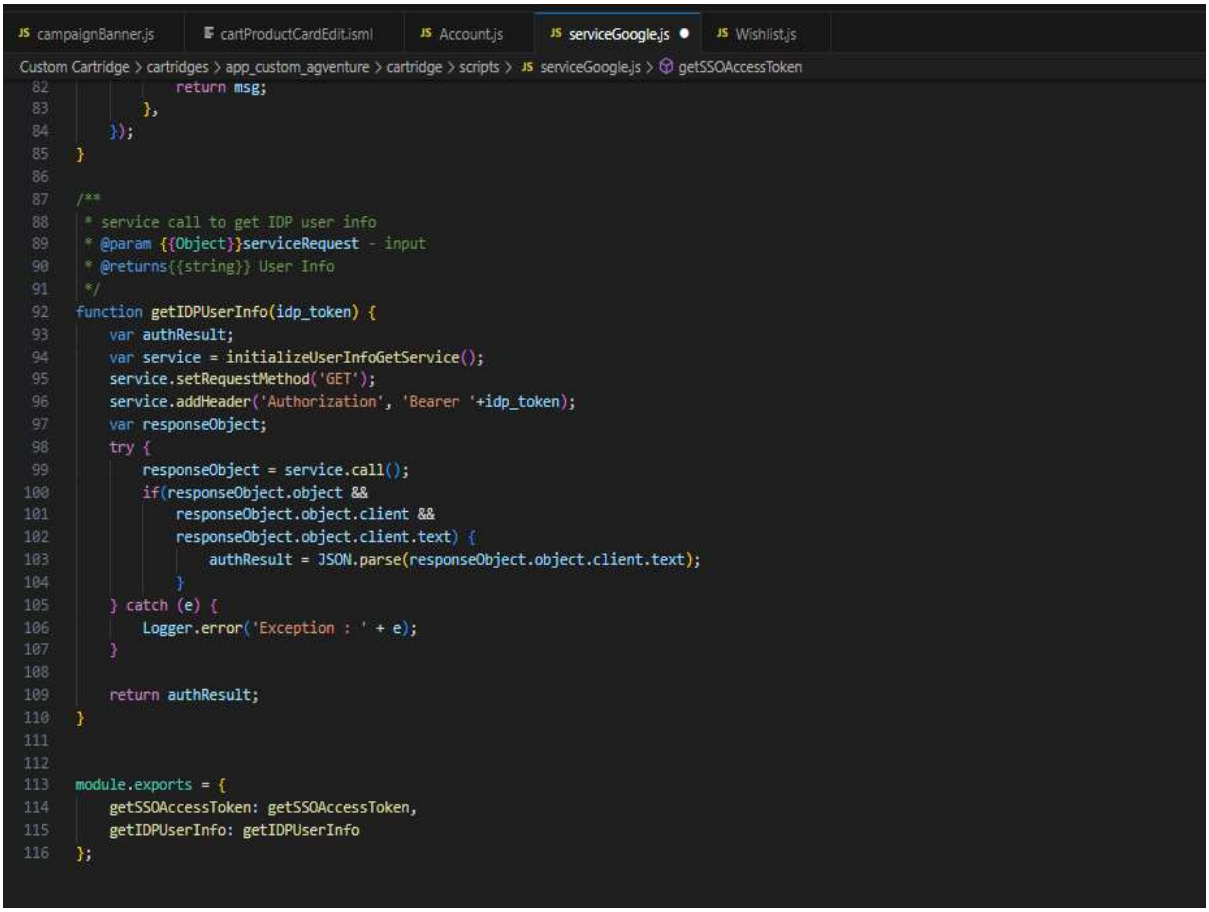
```
function getSSOAccessToken(auth_code, grant_type, refresh_token) {
    var authResult;
    var service = initializeService();

    var config = service.getConfiguration();
    var credential = config.getCredential();
    var user = credential.getUser();
    var password = credential.getPassword();
    var base64 = require('dw/util/StringUtils').encodeBase64(user + ':' + password);

    service.setRequestMethod('POST');
    service.addHeader('Content-Type','application/x-www-form-urlencoded');
    service.addHeader('Authorization', 'Basic ' + base64);
    service.addParam('client_id',Site.current.getCustomPreferenceValue('SSO_Client_id'));
    service.addParam('redirect_uri',Site.current.getCustomPreferenceValue('SSO_Redirect_uri'));
    service.addParam('channel_id',Site.current.getCustomPreferenceValue('SSO_Channel_id'));
    if (grant_type == 'authorization_code') {
        service.addParam('code',auth_code);
    } else if (grant_type == 'refresh_token') {
        service.addParam('refresh_token', refresh_token);
    }
    service.addParam('grant_type', grant_type);
    var responseObject;
    try {
        responseObject = service.call();
        if(responseObject.object &&
            responseObject.object.client &&
            responseObject.object.client.text) {
                authResult = JSON.parse(responseObject.object.client.text);
            }
    } catch (e) {
        Logger.error('Exception : ' + e);
    }
    return authResult;
}
```

Now that the cookies are handled, we can take care of getting the user information from our IDP. This info can be then displayed in our Account Dashboard.

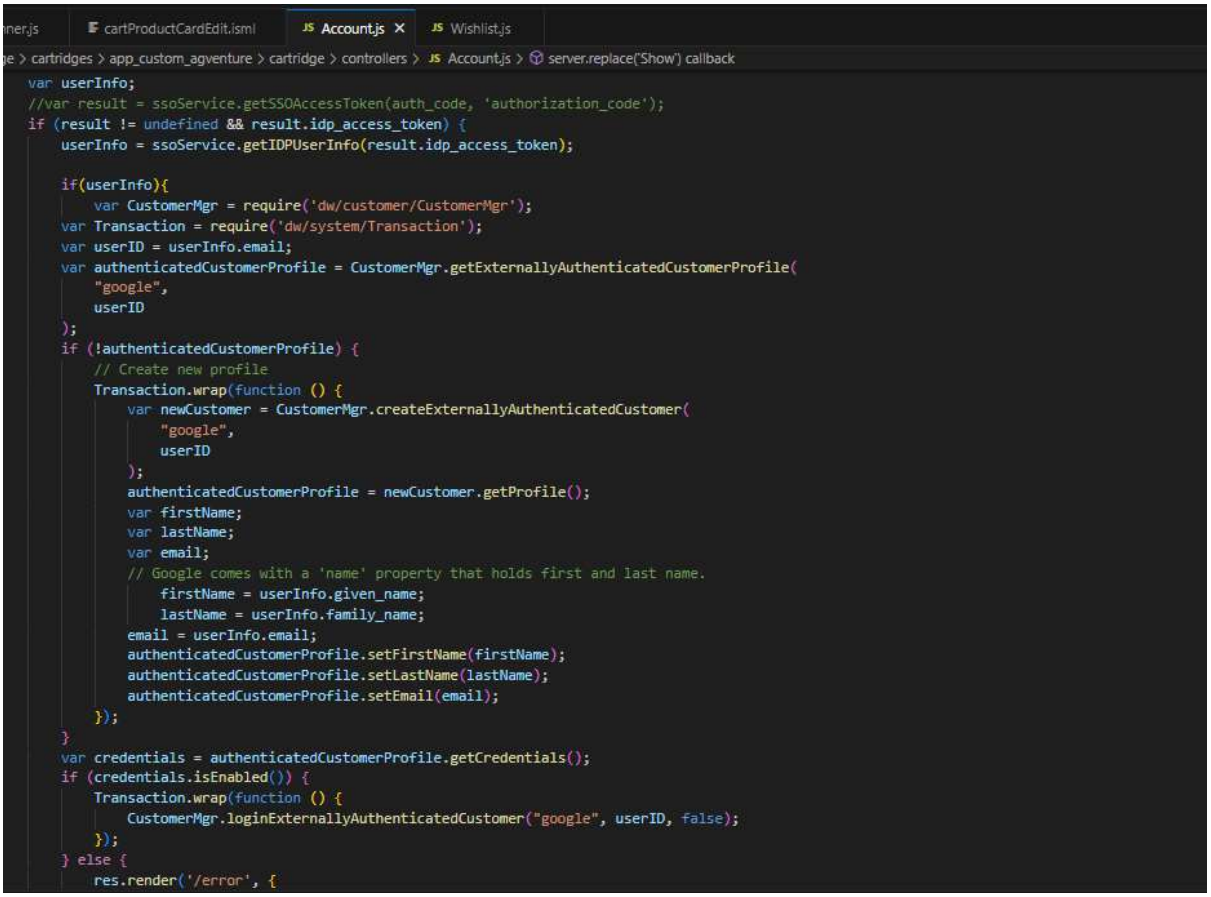
Let's start by creating a new service for this. This service will make a call to the API — https://www.googleapis.com/oauth2/v3/userinfo.

```
<?xml version="1.0" encoding="UTF-8"?>
<services xmlns="http://www.demandware.com/xml/impex/services/2014-09-26">
    <service-credential service-credential-id="SSO_userinfo">
        <url>https://www.googleapis.com/oauth2/v3/userinfo</url>
    </service-credential>
    <service-profile service-profile-id="SSO_userinfo">
        <timeout-millis>1000</timeout-millis>
        <rate-limit-enabled>false</rate-limit-enabled>
        <rate-limit-calls>0</rate-limit-calls>
        <rate-limit-millis>0</rate-limit-millis>
        <cb-enabled>true</cb-enabled>
        <cb-calls>0</cb-calls>
        <cb-millis>0</cb-millis>
    </service-profile>
    <service service-id="http.sso.userinfo">
        <service-type>HTTP</service-type>
        <enabled>true</enabled>
        <log-prefix>SSO</log-prefix>
        <comm-log-enabled>true</comm-log-enabled>
        <force-prd-enabled>true</force-prd-enabled>
        <mock-mode-enabled>false</mock-mode-enabled>
        <profile-id>SSO_userinfo</profile-id>
        <credential-id>SSO_userinfo</credential-id>
    </service>
</services>
```

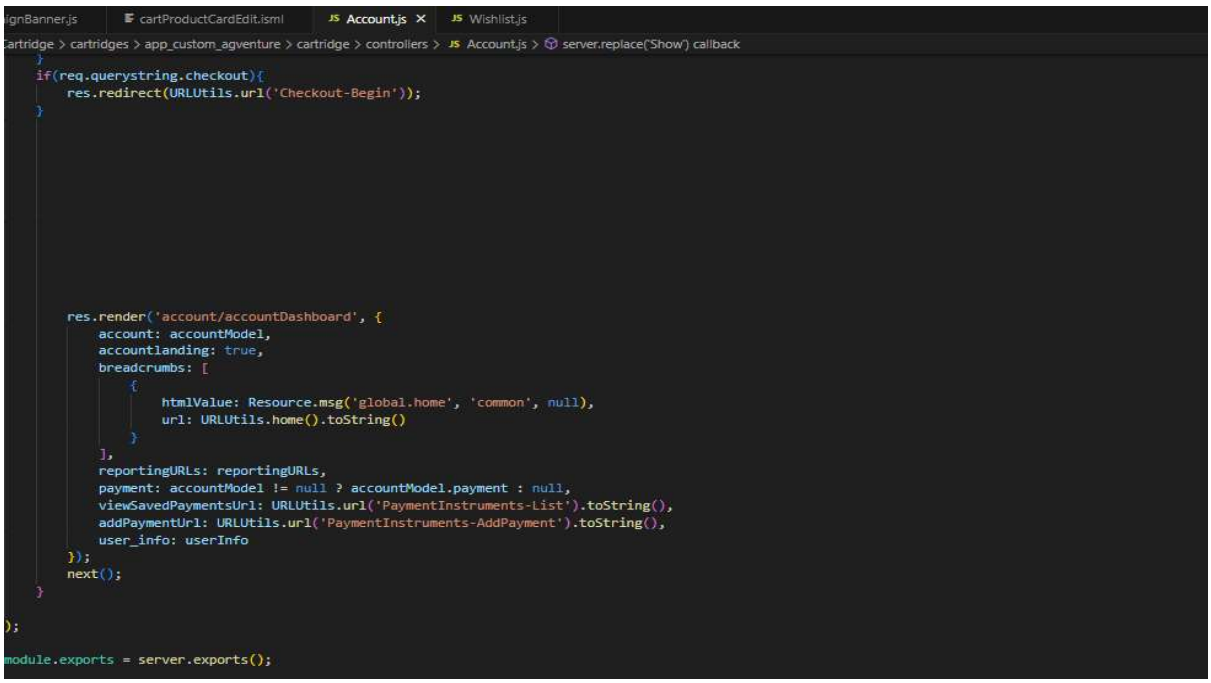
We can establish the connection in SSOServices.js file.

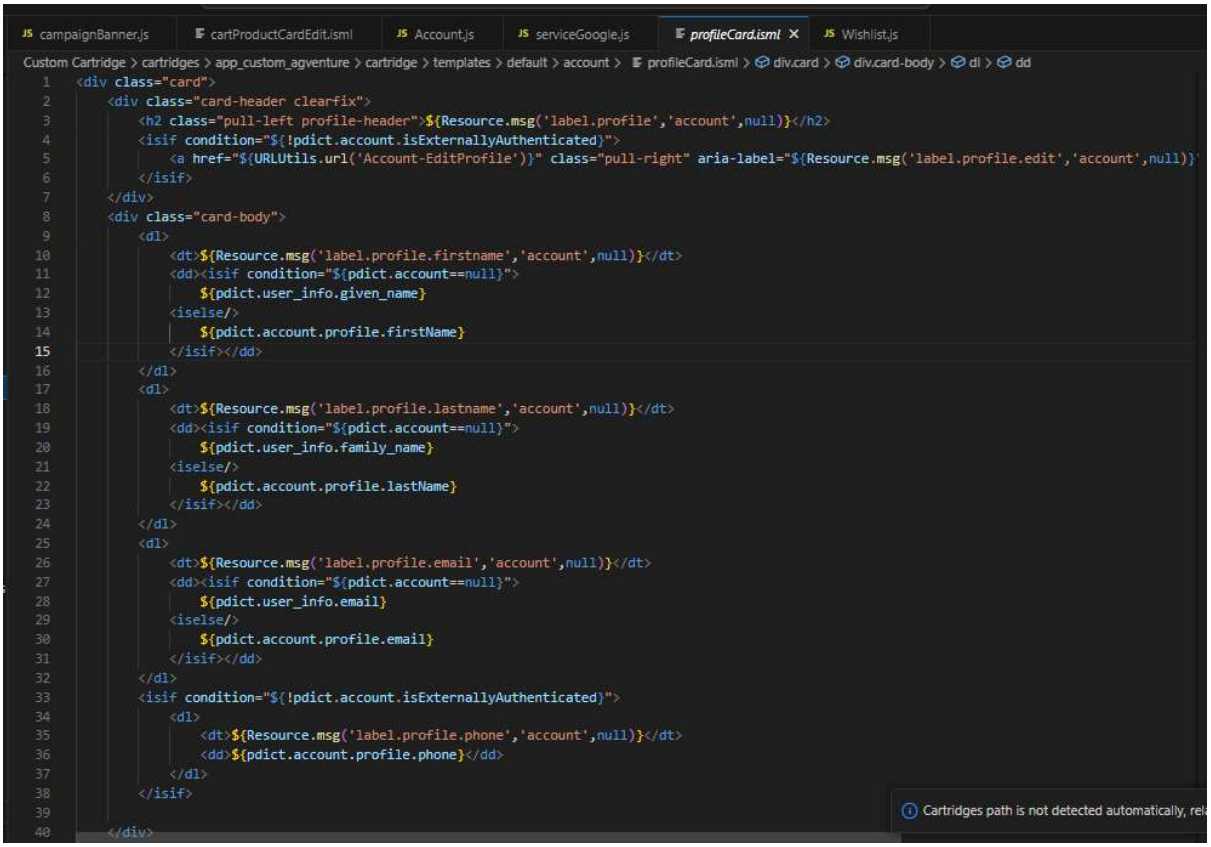

Remember how we got **idp_access_token** as response from the SLAS '/token' endpoint? We need this IDP token to get our user info. Let's call the service file from Account.js file. Create the customer after getting user information. If the user is logging in checkout page then redirect to Shipping form after creating the user in storefront.


ner.js
cartProductCardEdit.isml
Account.js
Wishlist.js
e > cartridges > app_custom_agventure > cartridge > controllers > Account.js > server.replace('Show') callback
var userInfo;
//var result = ssoService.getSSOAccessToken(auth_code, 'authorization_code');
if (result != undefined && result.idp_access_token) {
userInfo = ssoService.getIDPUserInfo(result.idp_access_token);
if(userInfo){
var CustomerMgr = require('dw/customer/CustomerMgr');
var Transaction = require('dw/system/Transaction');
var userID = userInfo.email;
var authenticatedCustomerProfile = CustomerMgr.getExternallyAuthenticatedCustomerProfile(
"google",
userID
);
if (!authenticatedCustomerProfile) {
// Create new profile
Transaction.wrap(function () {
var newCustomer = CustomerMgr.createExternallyAuthenticatedCustomer(
"google",
userID
);
authenticatedCustomerProfile = newCustomer.getProfile();
var firstName;
var lastName;
var email;
// Google comes with a 'name' property that holds first and last name.
firstName = userInfo.given_name;
lastName = userInfo.family_name;
email = userInfo.email;
authenticatedCustomerProfile.setFirstName(firstName);
authenticatedCustomerProfile.setLastName(lastName);
authenticatedCustomerProfile.setEmail(email);
});
}
var credentials = authenticatedCustomerProfile.getCredentials();
if (credentials.isEnabled()) {
Transaction.wrap(function () {
CustomerMgr.loginExternallyAuthenticatedCustomer("google", userID, false);
});
} else {
res.render('/error', {

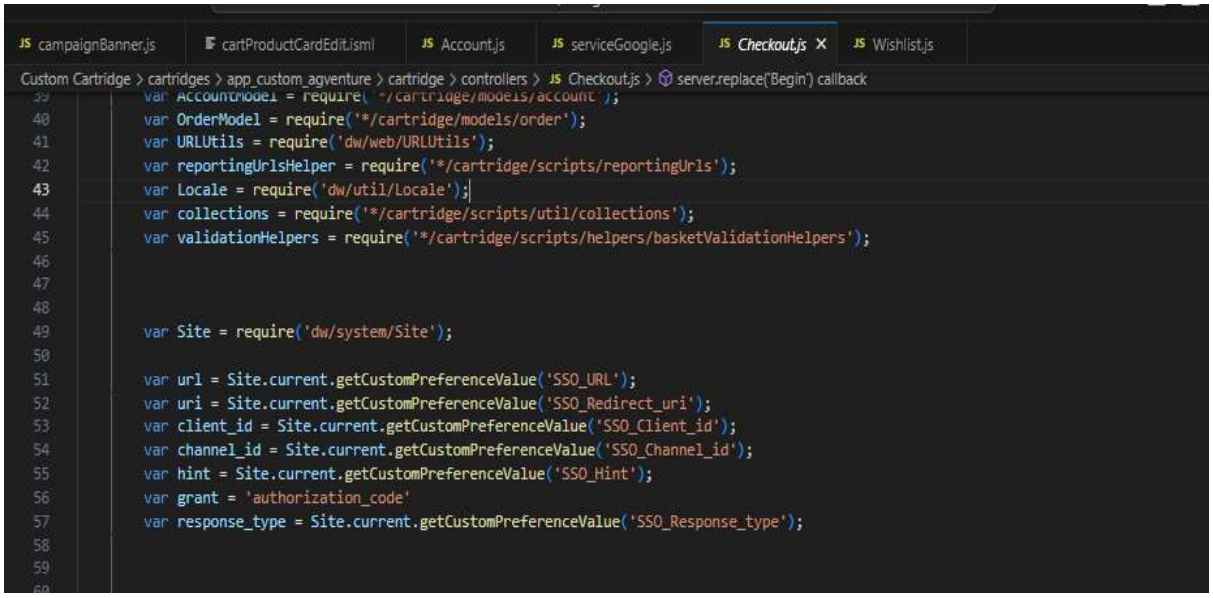

} else {
res.render('/error', {
message: Resource.msg('error.oauth.login.failure', 'login', null)
});
}
}else{
res.redirect(URLUtils.url('Login-Show'));
}
}
if(req.querystring.checkout){
res.redirect(URLUtils.url('Checkout-Begin'));
}

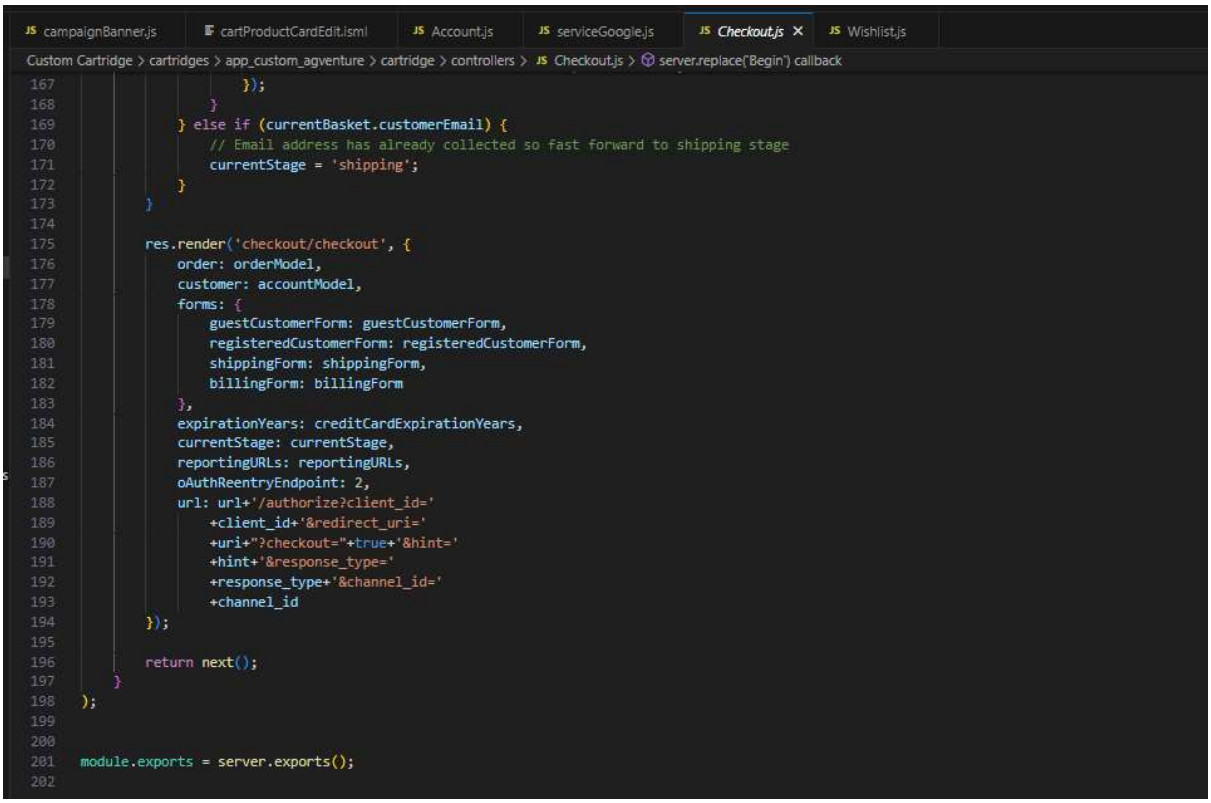

ignBanner.js
cartProductCardEdit.isml
Account.js
Wishlist.js
cartridge > cartridges > app_custom_agventure > cartridge > controllers > Account.js > server.replace('Show') callback
}
if(req.querystring.checkout){
res.redirect(URLUtils.url('Checkout-Begin'));
}
res.render('account/accountDashboard', {
account: accountModel,
accountlanding: true,
breadcrumbs: [
{
htmlValue: Resource.msg('global.home', 'common', null),
url: URLUtils.home().toString()
}
],
reportingURLs: reportingURLs,
payment: accountModel != null ? accountModel.payment : null,
viewSavedPaymentsUrl: URLUtils.url('PaymentInstruments-List').toString(),
addPaymentUrl: URLUtils.url('PaymentInstruments-AddPayment').toString(),
user_info: userInfo
});
next();
}
);
module.exports = server.exports();

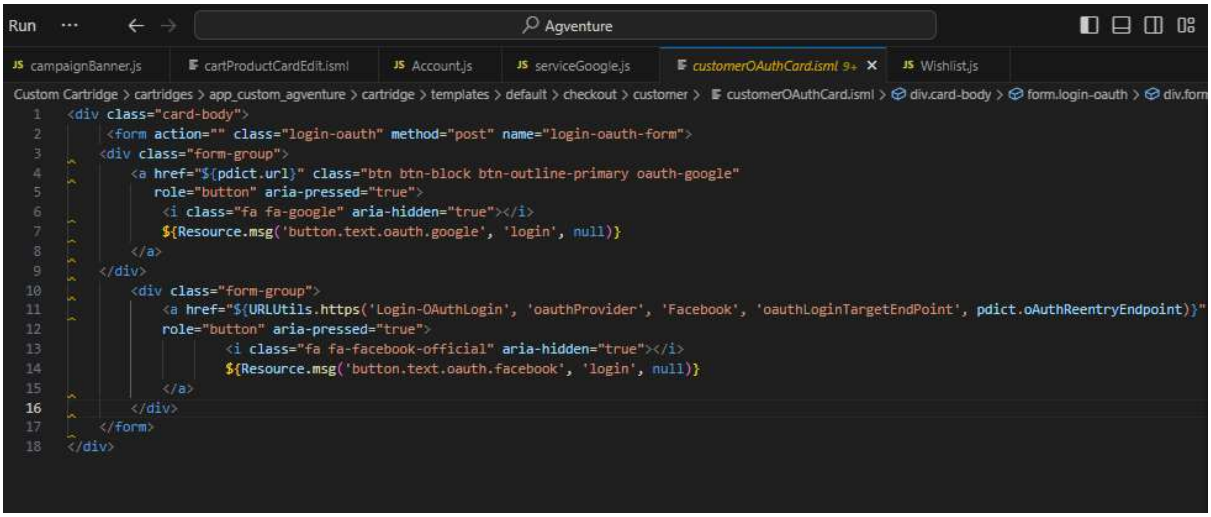
Now, we can just render this data in profileCard.isml.



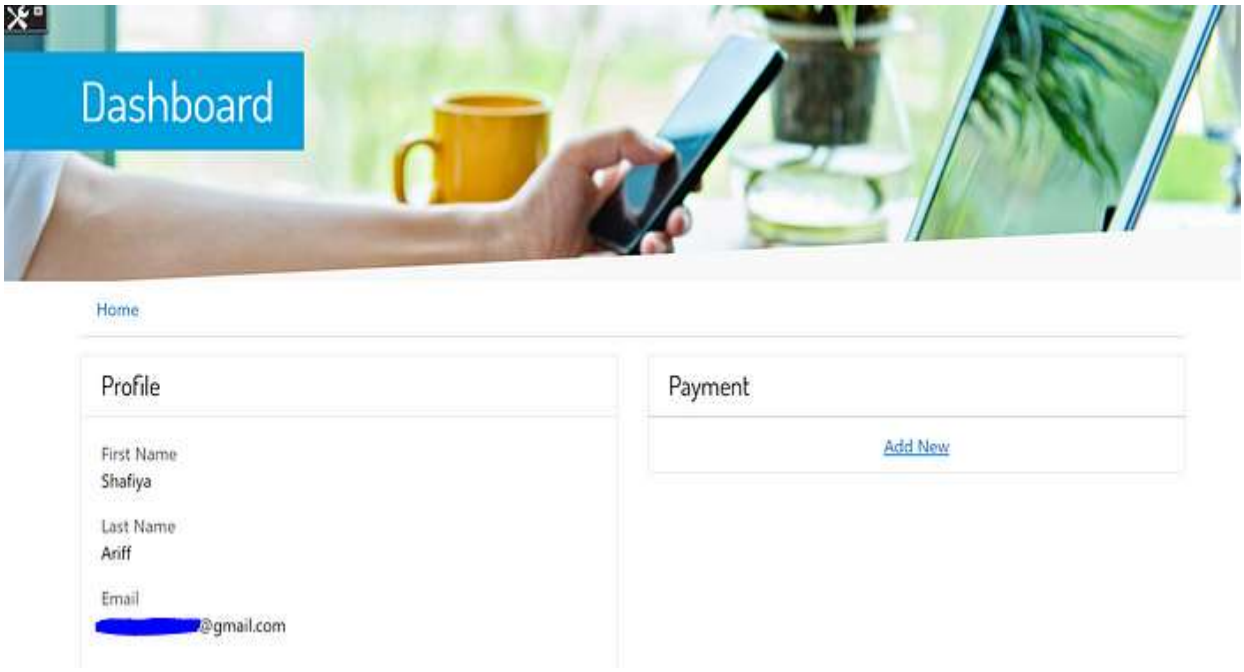

To add SSO with Google in checkout page, add following code.






And that's it! You are now signed it and can see your details in the Account Dashboard!




Thanks for reading :)

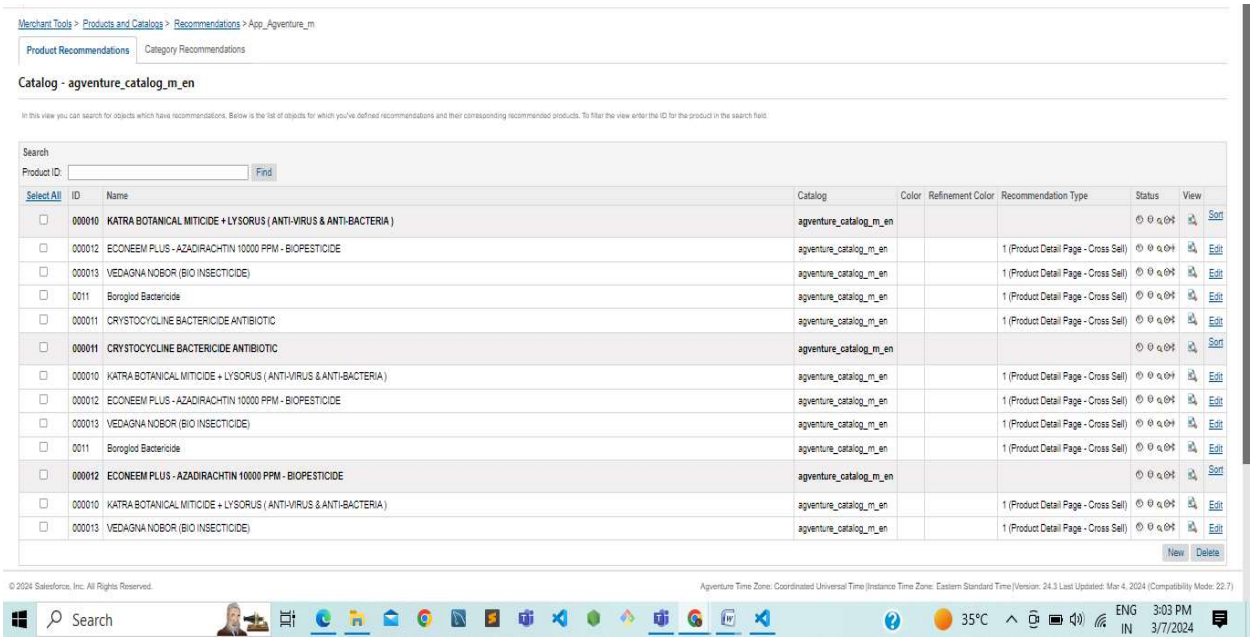
**For more knowledge visit following links**

**https://medium.com/@shafiya.ariff23/part-1-how-to-implement-shopper-login-and-api-access-service-slas-in-salesforce-commerce-cloud-18c8980ec1c0**

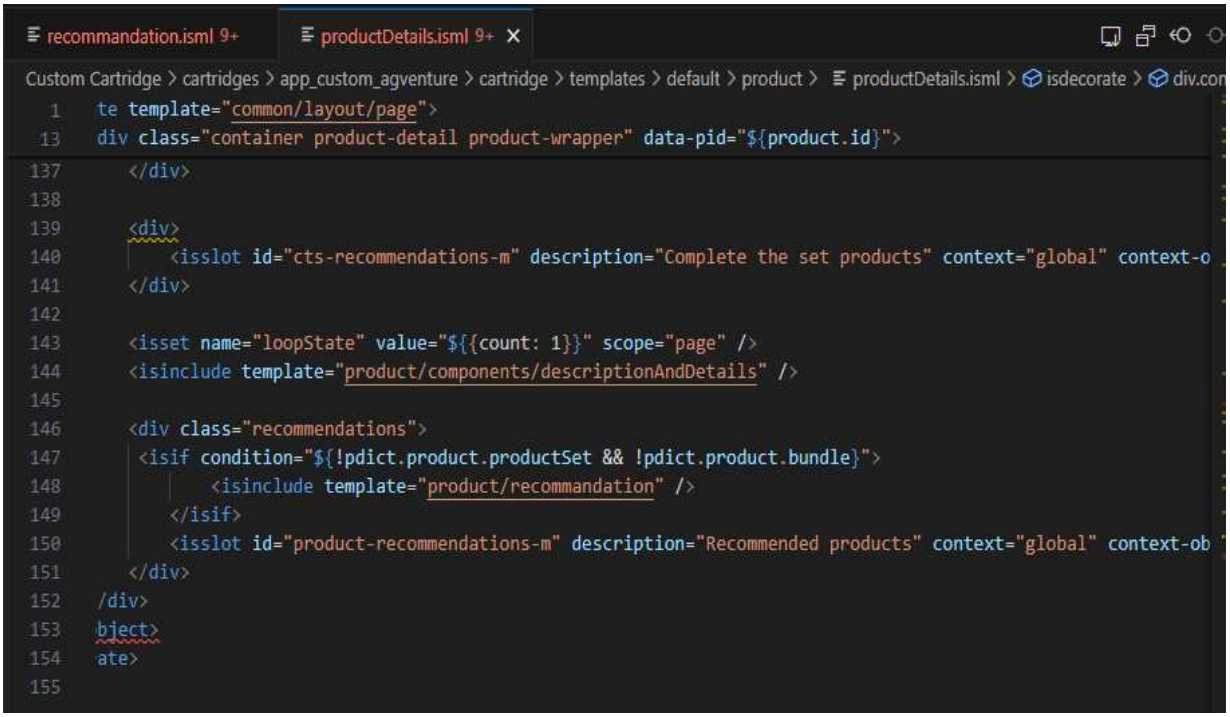
**https://medium.com/@shafiya.ariff23/part-2-how-to-implement-shopper-login-and-api-access-service-slas-in-salesforce-commerce-cloud-b88fc524804f**

# 1. Recommendation Products on PDP Page

- Add recommendation products for each product in business manager
- By navigating to Merchant Tools > Products and Catalogs > Recommendations > App_Agventure_m create new product recommendation




- Extract these products and display in pdp page
- To Display recommended products create new ISML file and add this file in ProductDetails.isml file



recommandation.isml 9+ X  productDetails.isml 9+

Custom Cartridge > cartridges > app_custom_agventure > cartridge > templates > default > product > recommandation.isml > isscript

```
<isscript>
    var ProductMgr = require('dw/catalog/ProductMgr');
        var Product = ProductMgr.getProduct(pdict.product.id);
        var recommendations = Product.getOrderableRecommendations(1).iterator();
        var recProducts = new dw.util.ArrayList();
        var maxRecs = 20, counter = 0;

        while( recommendations.hasNext() )
        {
            var recommendation = recommendations.next();
            var recommendedProduct = recommendation.getRecommendedItem();

            recProducts.add( recommendedProduct );

            if(++counter >= maxRecs)
            {
                break;
            }
        }
</isscript>
```

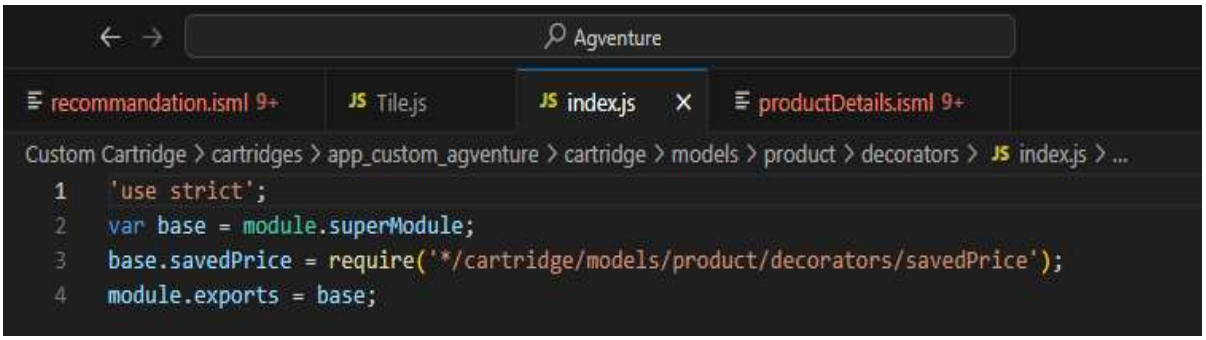
recommandation.isml 9+ X  productDetails.isml 9+

Custom Cartridge > cartridges > app_custom_agventure > cartridge > templates > default > product > recommandation.isml > isscript

```
<isif condition="${recProducts.size() > 0}">
 <isscript>
        var assets = require('*/cartridge/scripts/assets');
        assets.addJs('/js/recommend.js');
</isscript>
<h2 class="title d-none d-sm-block">${Resource.msg('pdp.recommend.heading','account',null)}</h2>
<div class="wrapper">
<ul class="carousel">
<isloop items="${recProducts}" var="product" status="st">
    <li class="card" id="${st.count}">
        <isinclude url="${URLUtils.url('Tile-Show', 'pid', product.ID, 'swatches', true, 'ratings', true)}" />
    </li>
    <isif condition="${st.last}">
        <isset name="count" value="${st.count}" scope="page" />
    </isif>
</isloop>
</ul>
<a href="#1"><i id="left" class="fa fa-arrow-circle-left fa-2x arrow1"></i></a>
<a href="${"#"+count}"><i id="right" class="fa fa-arrow-circle-right fa-2x arrow2"></i></a>
</div>
</isif>
```

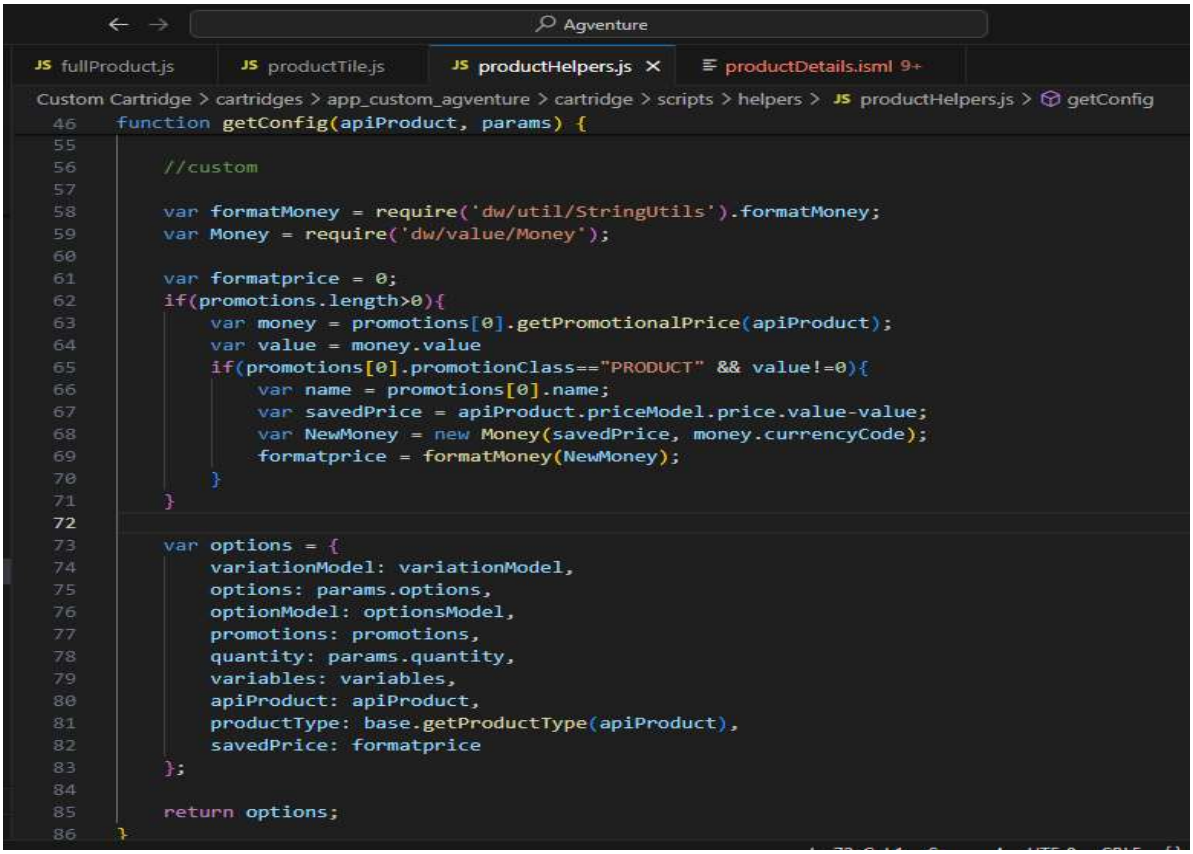
## 2. Displaying Promotion details on Product Tile and PDP Page

- Created new decorator file under model folder and implemented saved price

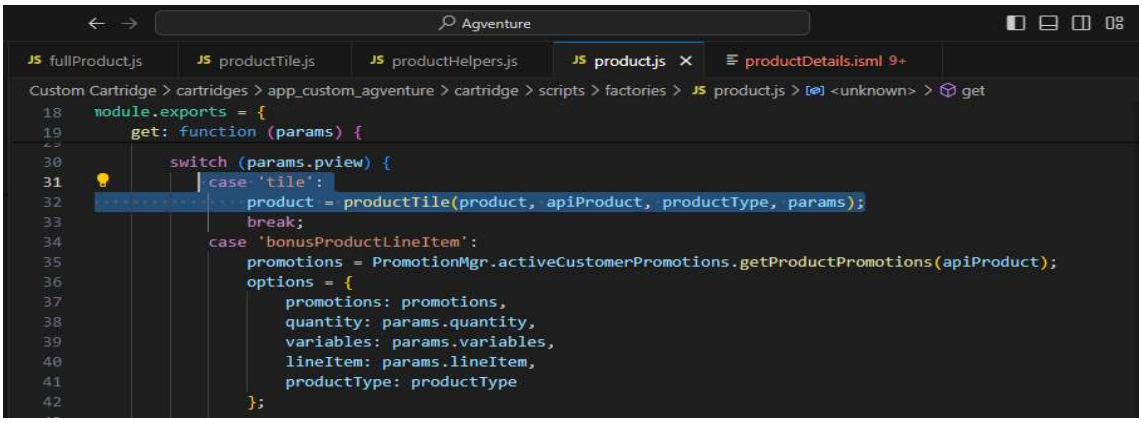


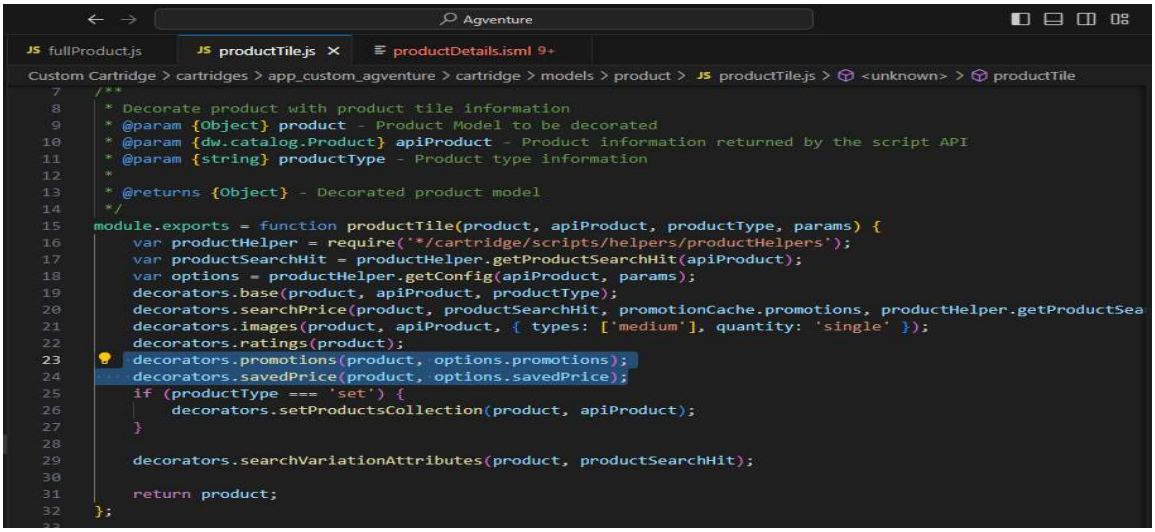


- To Create Decorator file we need to configure Saved price in product Helper file

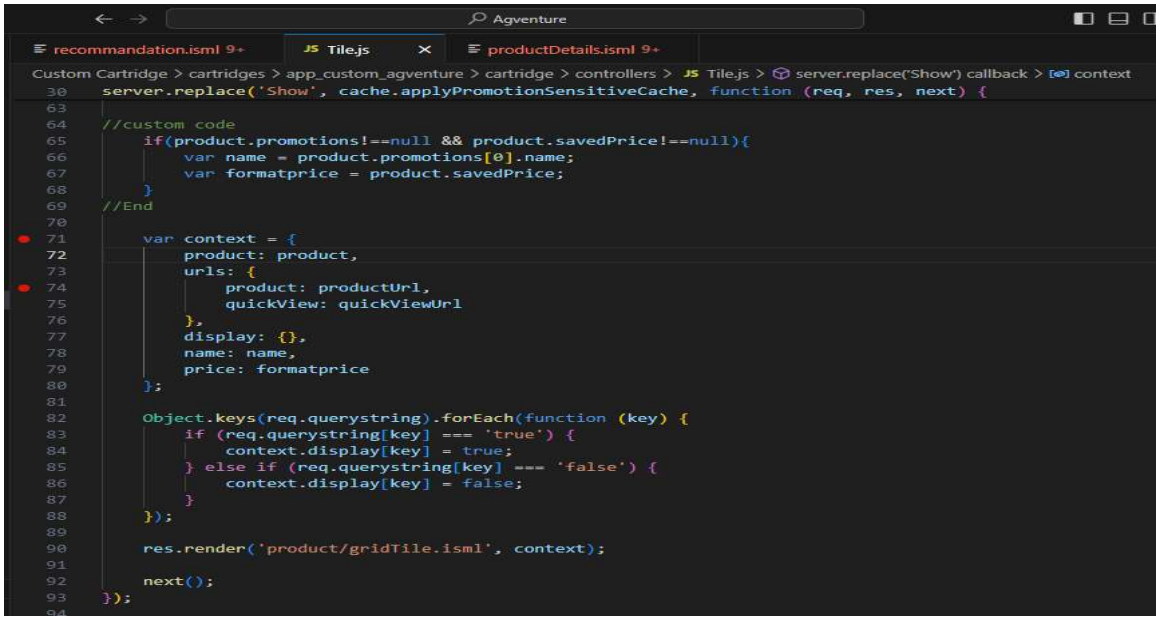


- Call the product object by using product and productTile files
- This product object will be called from Tile.js controller file,
  First we call the product file and from product file we will call the product Tile file

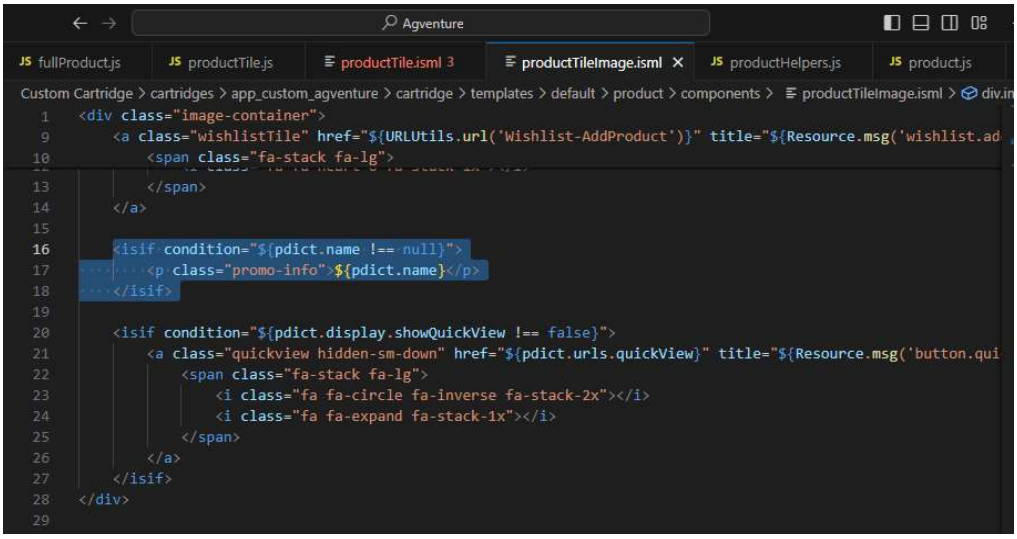


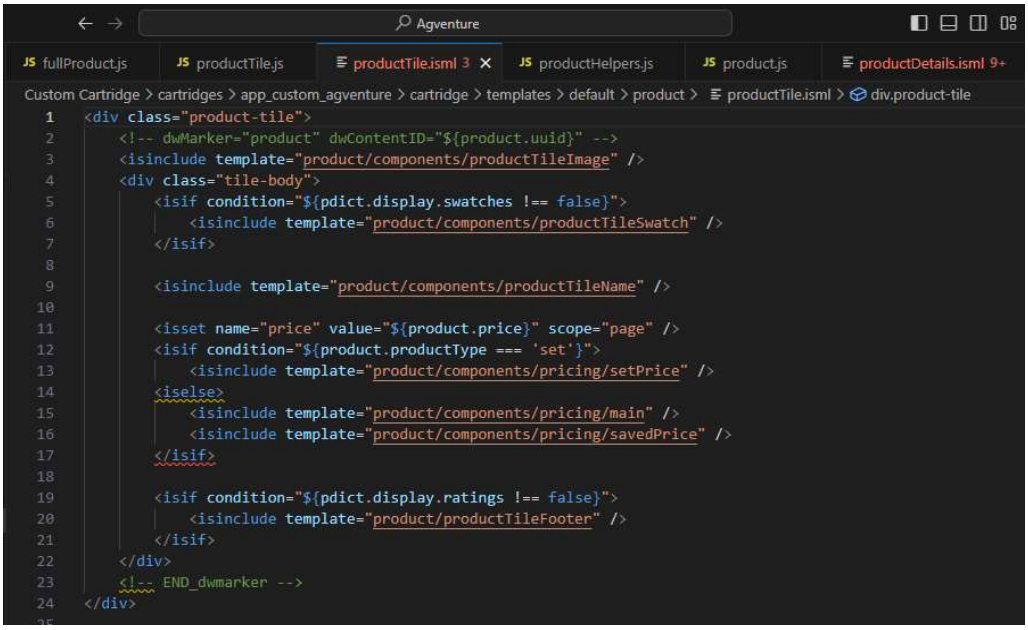

- Tile.js controller



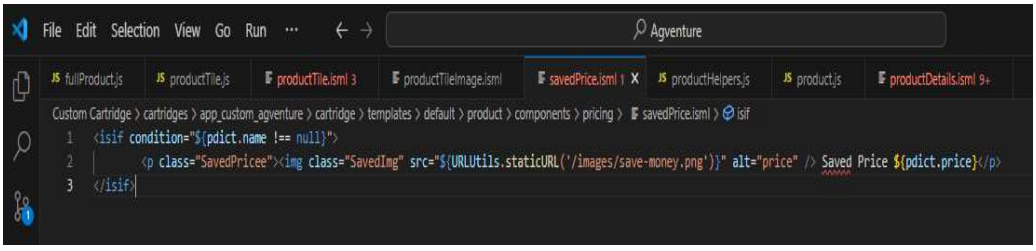
- Displaying Promotion name and Saved Price on Product Tile



```
<div class="image-container">
    <a class="wishlistTile" href="${URLUtils.url('Wishlist-AddProduct')}" title="${Resource.msg('wishlist.ad
        <span class="fa-stack fa-lg">
        </span>
    </a>

    <isif condition="${pdict.name !== null}">
        <p class="promo-info">${pdict.name}</p>
    </isif>

    <isif condition="${pdict.display.showQuickView !== false}">
        <a class="quickview hidden-sm-down" href="${pdict.urls.quickView}" title="${Resource.msg('button.qui
            <span class="fa-stack fa-lg">
                <i class="fa fa-circle fa-inverse fa-stack-2x"></i>
                <i class="fa fa-expand fa-stack-1x"></i>
            </span>
        </a>
    </isif>
</div>
```



```
<div class="product-tile">
    <!-- dwMarker="product" dwContentID="${product.uuid}" -->
    <isinclude template="product/components/productTileImage" />
    <div class="tile-body">
        <isif condition="${pdict.display.swatches !== false}">
            <isinclude template="product/components/productTileSwatch" />
        </isif>

        <isinclude template="product/components/productTileName" />

        <isset name="price" value="${product.price}" scope="page" />
        <isif condition="${product.productType === 'set'}">
            <isinclude template="product/components/pricing/setPrice" />
        <iselse>
            <isinclude template="product/components/pricing/main" />
            <isinclude template="product/components/pricing/savedPrice" />
        </isif>

        <isif condition="${pdict.display.ratings !== false}">
            <isinclude template="product/productTileFooter" />
        </isif>
    </div>
    <!-- END_dwmarker -->
</div>
```



```
<isif condition="${pdict.name !== null}">
    <p class="SavedPricee"><img class="SavedImg" src="${URLUtils.staticURL('/images/save-money.png')}" alt="price" /> Saved Price ${pdict.price}</p>
</isif>
```

- Displaying Promotion information on Product Detail page using productDetails.isml file
- Updating Saved Price value in fullProduct Object