HCL Digital Experience

HDX-DEV-200
Script Application

Edition July 2025

Herbert Hilhorst – herbert.hilhorst@hcl-software.com

Tim Golledge – tim.golledge@hcl-software.com

Table of Contents

Authors

This document was created by the following Subject Matter Experts:

Herbert Hilhorst
Company: HCLSoftware

Bio

Herbert Hilhorst is an HCL Digital Experience (DX) Technical Advisor at HCLSoftware.

Contact: herbert.hilhorst@hcl-software.com

Tim Golledge
Company: HCLSoftware

Bio

Tim Golledge is an HCL Digital Experience (DX) Technical Advisor at HCLSoftware.

Contact: tim.golledge@hcl-software.com

Introduction

This hands-on lab guides you through creating more advanced applications on HCL Digital Experience (DX), using its advanced Script Application capabilities. You will experience how quick and easy it is to use JavaScript frameworks, like React and Angular to create and update applications.

In this DX developer lab, you play the role of Gene, a developer for the fictitious Woodburn Studio company.

Gene Hayes, Developer, based in Chicago (USA)

As a Web Developer you will discover new ways to develop reusable Script Applications, including using JavaScript frameworks, like Angular and React. You will also learn how to have these applications use HCL DX and HCL Leap data and have the applications communicate to each other.

Prerequisites

1. Completion of HDX-INTRO, HDX-BU-100 and HDX-DEV-100 courses, including the labs. This gets you DXClient deployed and configured, and have some basic skills on how to develop Script Applications online and locally.
2. Access to download the Lab Resources. In the same place where you have found this lab, you will find corresponding resources which you may download and unzip in your Desktop. This helps you to run the lab more easily, and you may later replace them by your own ones.
3. A local code editor. This lab provides instructions with Visual Studio Code (VSC): https://code.visualstudio.com

You will be using the following user IDs and passwords:

Purpose	**User**	**Password**
SoFy Login	Your official email id	Your password
Developer Gene Hayes	ghayes (or wpsadmin)	HCL-Dem0 (or wpsadmin)

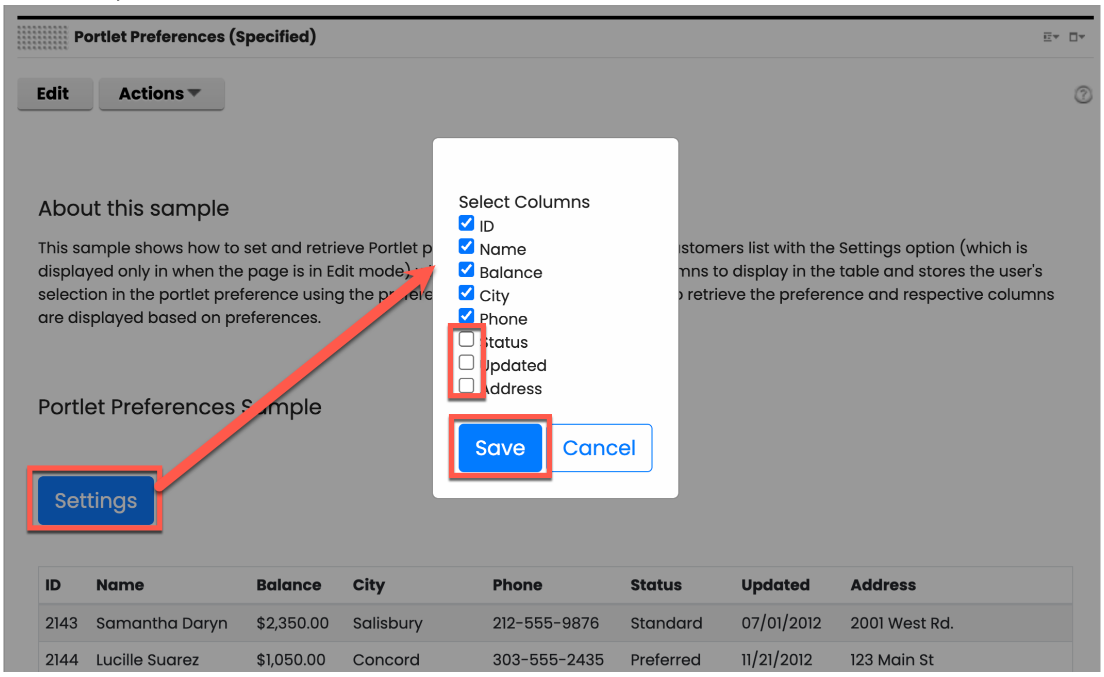
Lab Overview

In this lab you will learn more advanced features of the Script Application, like managing the preferences of each instance, using Angular and React, consuming data from HCL DX and HCL Leap and have different Script Applications communicate with each other.
You will use some existing Script Applications to easily get this working and you will learn how to change an existing JavaScript application into a DX Script Application. This is illustrated here by creating a new React Script Application from scratch, using one of the many ways documented in https://react.dev/learn/build-a-react-app-from-scratch, in this case you will use Vite https://vite.dev. You may also apply these steps to any JavaScript application you may have found or developed using an AI tool.

Here are the three parts.

Part 1: Use multiple instances of the same Script Application with different preferences

You will first deploy a Script Application twice on the same page and configure each instance with different preferences.



About this sample

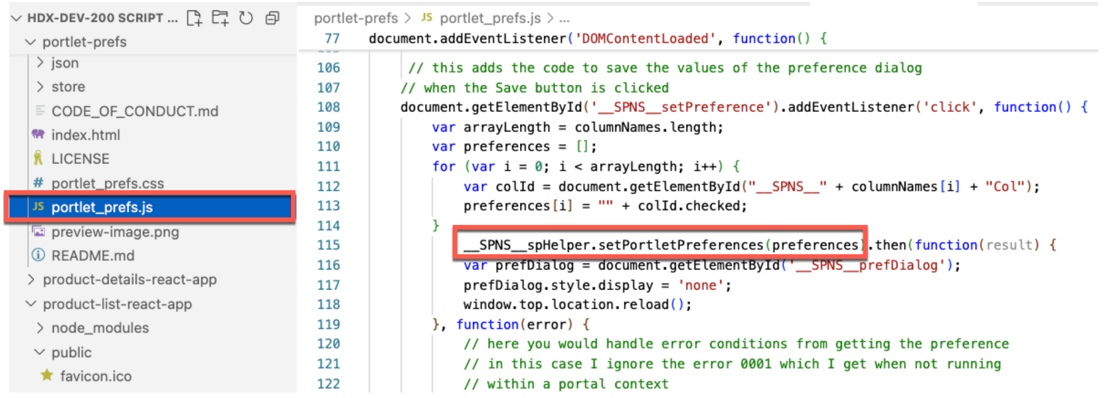
This sample shows how to set and retrieve Portlet p
displayed only in when the page is in Edit mode)
selection in the portlet preference using the p
are displayed based on preferences.

stomers list with the Settings option (which is
mns to display in the table and stores the user's
retrieve the preference and respective columns

Portlet Preferences Sample

ID	Name	Balance	City	Phone	Status	Updated	Address
2143	Samantha Daryn	$2,350.00	Salisbury	212-555-9876	Standard	07/01/2012	2001 West Rd.
2144	Lucille Suarez	$1,050.00	Concord	303-555-2435	Preferred	11/21/2012	123 Main St

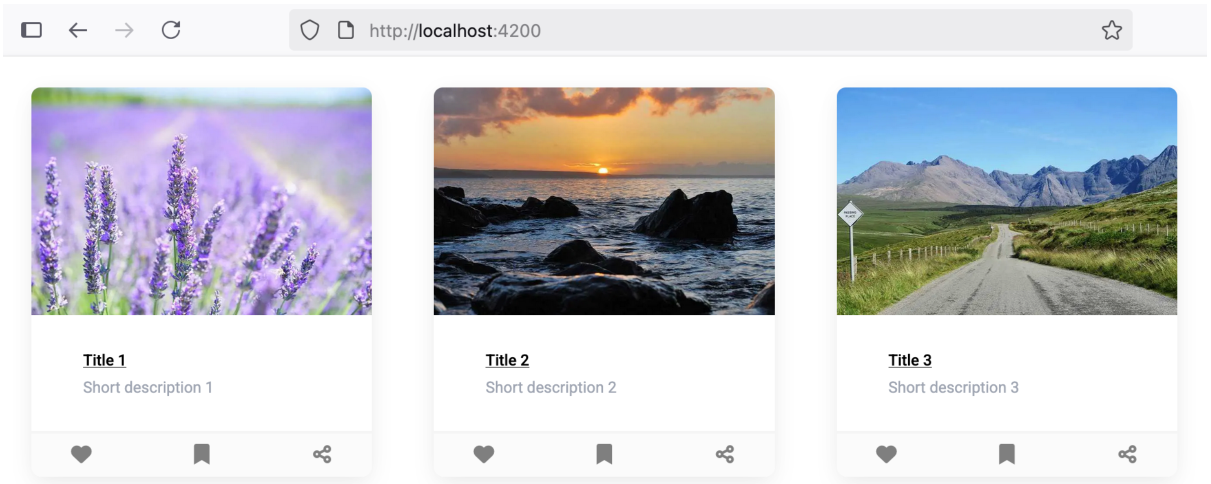
Then you will discover how this was developed.



```javascript
document.addEventListener('DOMContentLoaded', function() {
...
    // this adds the code to save the values of the preference dialog
    // when the Save button is clicked
    document.getElementById('__SPNS__setPreference').addEventListener('click', function() {
        var arrayLength = columnNames.length;
        var preferences = [];
        for (var i = 0; i < arrayLength; i++) {
            var colId = document.getElementById("__SPNS__" + columnNames[i] + "Col");
            preferences[i] = "" + colId.checked;
        }
        __SPNS__spHelper.setPortletPreferences(preferences).then(function(result) {
            var prefDialog = document.getElementById('__SPNS__prefDialog');
            prefDialog.style.display = 'none';
            window.top.location.reload();
        }, function(error) {
            // here you would handle error conditions from getting the preference
            // in this case I ignore the error 0001 which I get when not running
            // within a portal context
```

Part 2: Create a new Angular Script Application using DX articles content

You will update a new Angular Script Application that uses a local and public resources

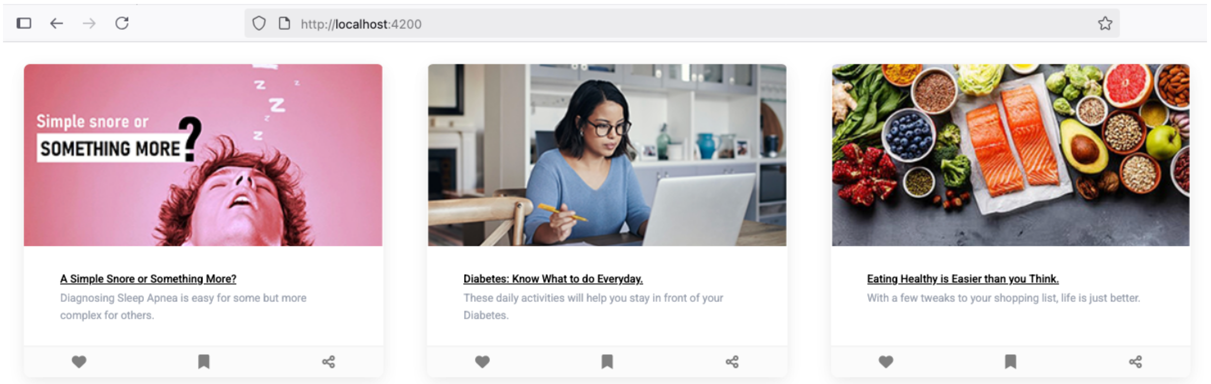

http://localhost:4200

Title 1
Short description 1

Title 2
Short description 2

Title 3
Short description 3

to use Woodburn Healthcare articles stored in DX and show them in a carousel. You will test it locally first and then deploy it to your DX server and add it to a page.


http://localhost:4200

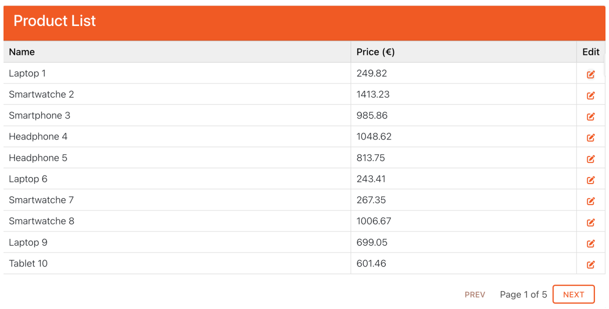
A Simple Snore or Something More?
Diagnosing Sleep Apnea is easy for some but more complex for others.

Diabetes: Know What to do Everyday.
These daily activities will help you stay in front of your Diabetes.

Eating Healthy is Easier than you Think.
With a few tweaks to your shopping list, life is just better.

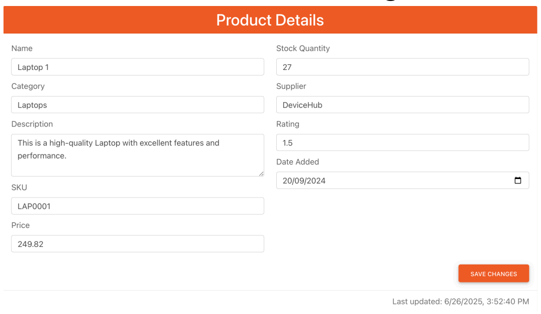
Part 3: Create a new React Script Application that manage product data stored in HCL Leap

You will create new React Product List and Product Details Script Applications that work with HCL Leap application data. You will first test them locally, then build minified versions, deploy them to the server, add them to a DX page and finally have them communicate with each together.


Product List

Name	Price (€)	Edit
Laptop 1	249.82	✎
Smartwatche 2	1413.23	✎
Smartphone 3	985.86	✎
Headphone 4	1048.62	✎
Headphone 5	813.75	✎
Laptop 6	243.41	✎
Smartwatche 7	267.35	✎
Smartwatche 8	1006.67	✎
Laptop 9	699.05	✎
Tablet 10	601.46	✎

PREV Page 1 of 5 NEXT


Product Details

Name	Laptop 1
Category	Laptops
Description	This is a high-quality Laptop with excellent features and performance.
SKU	LAP0001
Price	249.82
Stock Quantity	27
Supplier	DeviceHub
Rating	1.5
Date Added	20/09/2024

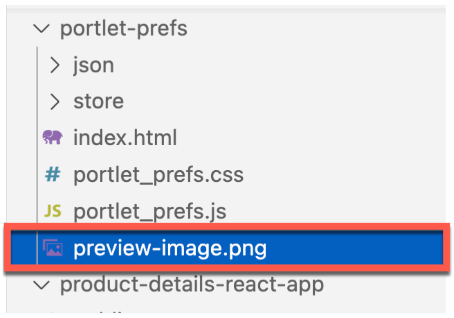
SAVE CHANGES

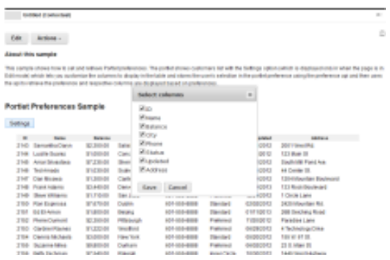
Last updated: 6/26/2025, 3:52:40 PM

Part 1: Use multiple instances of the same Script Application with different preferences

You will first deploy a Script Application twice on the same page and configure each instance with different preferences.

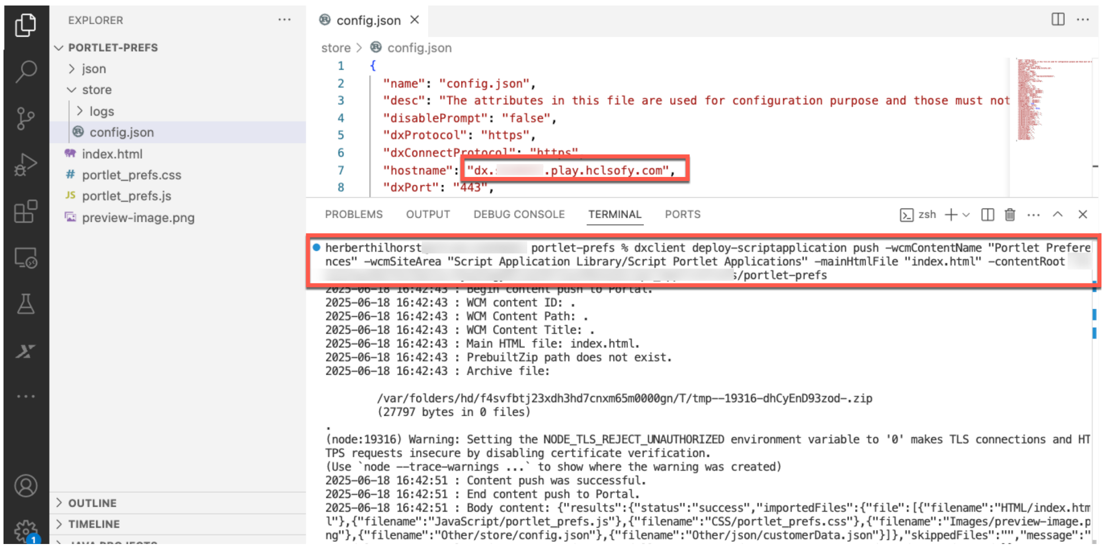
1. Download the portlet-prefs Script Application and open it in your favorite editor (e.g. Visual Studio Code). Notice the preview-image.png file used to manage the icon of the page component in DX.





2. Update the config.json with your hostname, open a Terminal and run the DXClient deploy Script Application command.

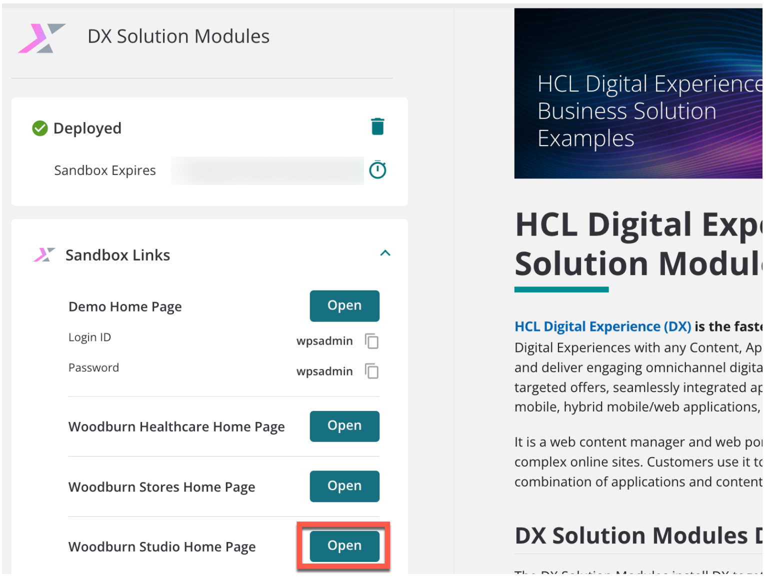
```
dxclient deploy-scriptapplication push -wcmContentName "Portlet Preferences" -wcmSiteArea "Script Application Library/Script Portlet Applications" -mainHtmlFile "index.html" -contentRoot <your path to where you downloaded the portlet-prefs Script Application>/portlet-prefs
```



```
{
  "name": "config.json",
  "desc": "The attributes in this file are used for configuration purpose and those must not
  "disablePrompt": "false",
  "dxProtocol": "https",
  "dxConnectProtocol": "https",
  "hostname": "dx.[redacted].play.hclsofy.com",
  "dxPort": "443",
```

```
herberthilhorst[redacted] portlet-prefs % dxclient deploy-scriptapplication push -wcmContentName "Portlet Prefere
nces" -wcmSiteArea "Script Application Library/Script Portlet Applications" -mainHtmlFile "index.html" -contentRoot [redacted]
s/portlet-prefs
2025-06-18 16:42:43 : Begin content push to Portal.
2025-06-18 16:42:43 : WCM content ID: .
2025-06-18 16:42:43 : WCM Content Path: .
2025-06-18 16:42:43 : WCM Content Title: .
2025-06-18 16:42:43 : Main HTML file: index.html.
2025-06-18 16:42:43 : PrebuiltZip path does not exist.
2025-06-18 16:42:43 : Archive file:

        /var/folders/hd/f4svfbtj23xdh3hd7cnxm65m0000gn/T/tmp--19316-dhCyEnD93zod-.zip
        (27797 bytes in 0 files)
.
(node:19316) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HT
TPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
2025-06-18 16:42:51 : Content push was successful.
2025-06-18 16:42:51 : End content push to Portal.
2025-06-18 16:42:51 : Body content: {"results":{"status":"success","importedFiles":{"file":[{"filename":"HTML/index.htm
l"},{"filename":"JavaScript/portlet_prefs.js"},{"filename":"CSS/portlet_prefs.css"},{"filename":"Images/preview-image.p
ng"},{"filename":"Other/store/config.json"},{"filename":"Other/json/customerData.json"}]},"skippedFiles":"","message":"
```

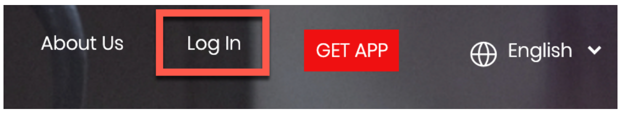
3. Then add the Script Application twice to the same DX page under the Woodburn Studio site. On the server where the DX Solution Modules has been installed, next to **Woodburn Studio Home** Page, click **Open**.



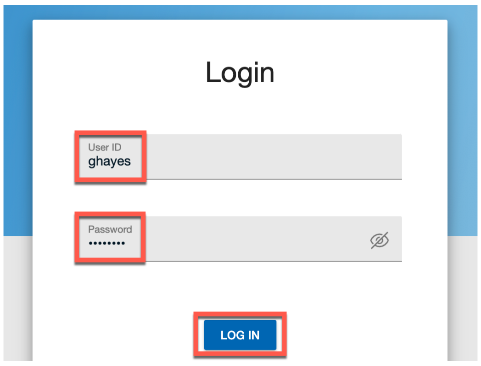
HCL Digital Experience (DX) is the fast
Digital Experiences with any Content, Ap
and deliver engaging omnichannel digita
targeted offers, seamlessly integrated ap
mobile, hybrid mobile/web applications,

It is a web content manager and web po
complex online sites. Customers use it t
combination of applications and content

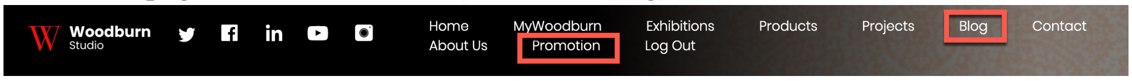
4. Log on to the DX server as Gene Hayes. Click **Log in**.



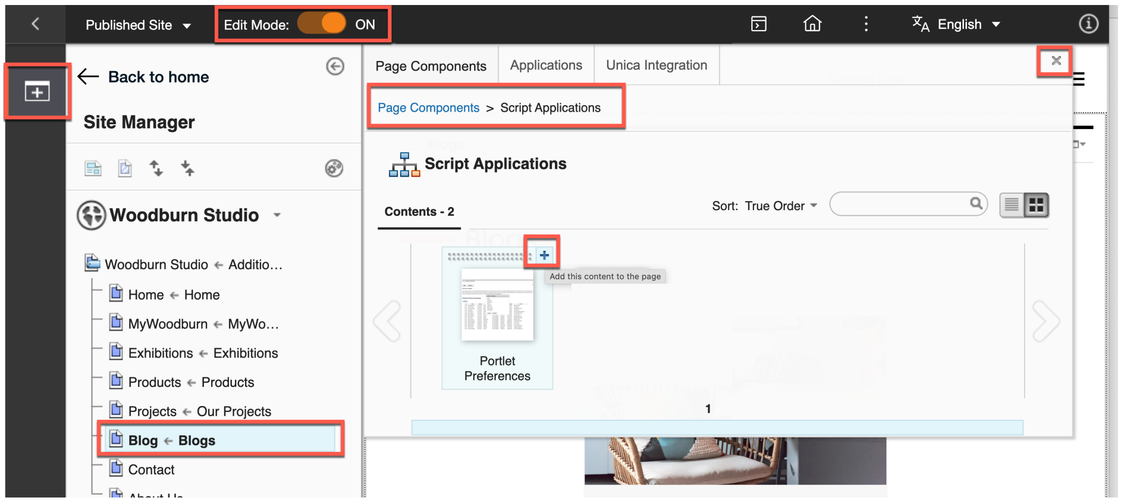
5. Then use the credentials of Gene (User ID: **ghayes**, Password: **HCL-Dem0** – if you do not have these, you may use the default administrator account with User ID: **wpsadmin,** Password: **wpsadmin)**



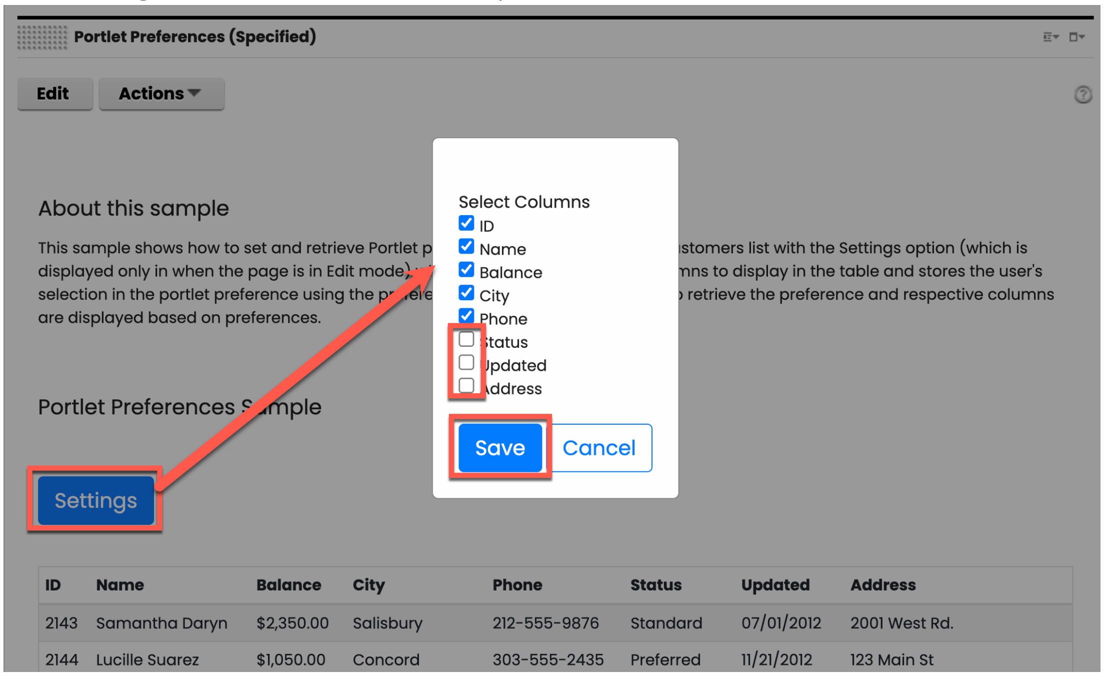
6. If you have created a Promotion page in the Theme Development lab, you may use that, otherwise use the existing Blog page. We'll show instructions with the Promotion page. Click **Promotion** (otherwise Blog).



7. Turn the Edit Mode on, click **Add page components and applications**, select Script Applications, add your new **Portlet Preferences** Script Application to your page twice (notice the icon) and close the toolbar.



8. Scroll down to the first instance, check the columns that show and update the preference. Click **Settings**, then uncheck the Status, Updated and Address columns and click **Save**.



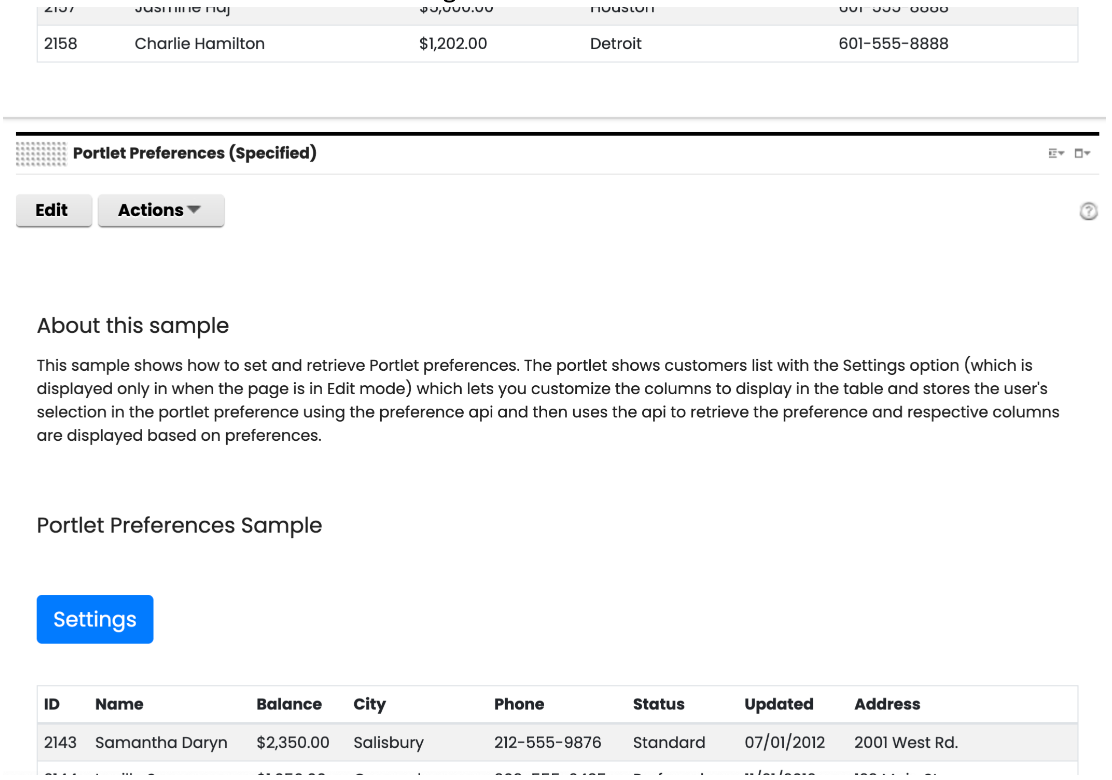
About this sample

This sample shows how to set and retrieve Portlet p
displayed only in when the page is in Edit mode)
selection in the portlet preference using the pr
are displayed based on preferences.
stomers list with the Settings option (which is
mns to display in the table and stores the user's
retrieve the preference and respective columns

Portlet Preferences Sample

ID	Name	Balance	City	Phone	Status	Updated	Address
2143	Samantha Daryn	$2,350.00	Salisbury	212-555-9876	Standard	07/01/2012	2001 West Rd.
2144	Lucille Suarez	$1,050.00	Concord	303-555-2435	Preferred	11/21/2012	123 Main St

9. You see that these columns do no longer appear. When you scroll further down, you will see that the other instance is still showing all the columns.



2158	Charlie Hamilton	$1,202.00	Detroit	601-555-8888

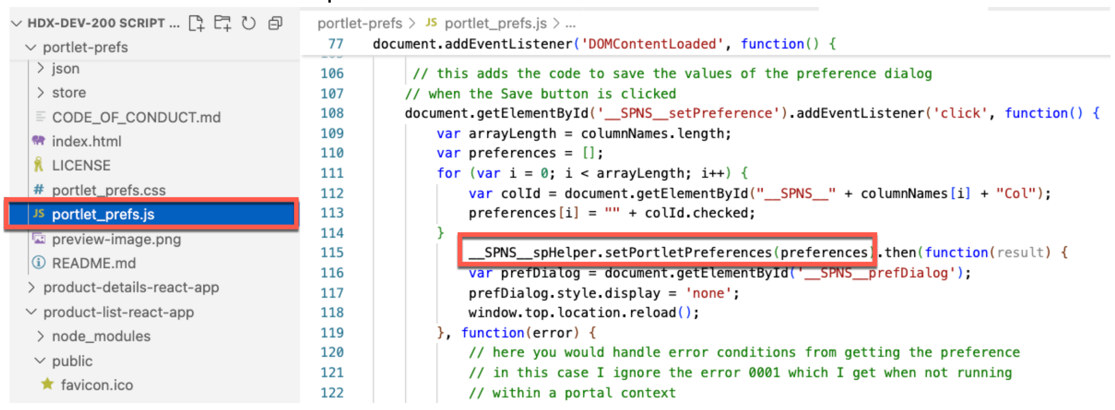
About this sample

This sample shows how to set and retrieve Portlet preferences. The portlet shows customers list with the Settings option (which is displayed only in when the page is in Edit mode) which lets you customize the columns to display in the table and stores the user's selection in the portlet preference using the preference api and then uses the api to retrieve the preference and respective columns are displayed based on preferences.

Portlet Preferences Sample

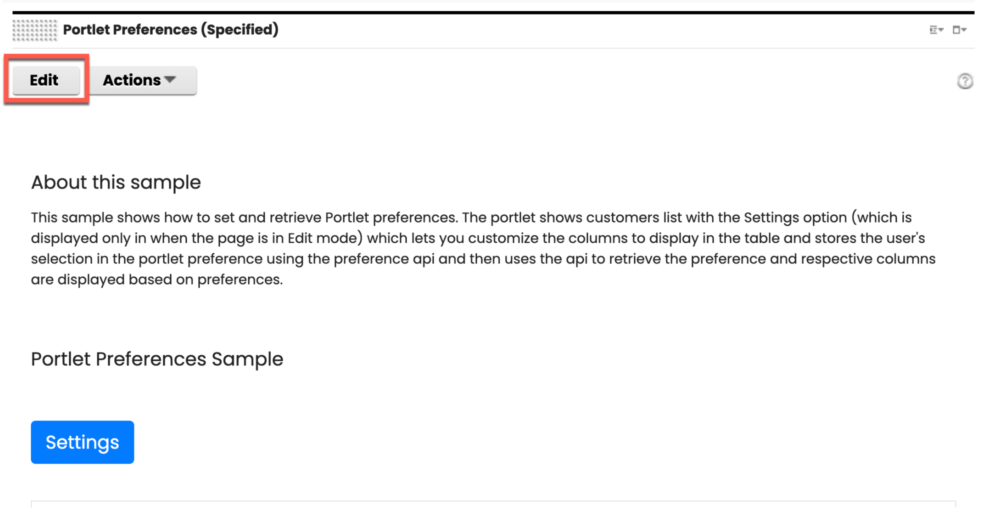
ID	Name	Balance	City	Phone	Status	Updated	Address
2143	Samantha Daryn	$2,350.00	Salisbury	212-555-9876	Standard	07/01/2012	2001 West Rd.

10. Feel free to try different settings on the other instance of your Portlet Preferences Script Application. Next, have a look at how this is implemented. Open the portlet_prefs.js file and have a look at the code. You will see it use the getPortletPreferences and setPortletPreferences methods of the spHelper. The prefix __SPNS__ is used to make the variables and methods unique.



```
document.addEventListener('DOMContentLoaded', function() {
    // this adds the code to save the values of the preference dialog
    // when the Save button is clicked
    document.getElementById('__SPNS__setPreference').addEventListener('click', function() {
        var arrayLength = columnNames.length;
        var preferences = [];
        for (var i = 0; i < arrayLength; i++) {
            var colId = document.getElementById("__SPNS__" + columnNames[i] + "Col");
            preferences[i] = "" + colId.checked;
        }
            __SPNS__spHelper.setPortletPreferences(preferences).then(function(result) {
            var prefDialog = document.getElementById('__SPNS__prefDialog');
            prefDialog.style.display = 'none';
            window.top.location.reload();
        }, function(error) {
            // here you would handle error conditions from getting the preference
            // in this case I ignore the error 0001 which I get when not running
            // within a portal context
```

11. Then edit the Script Application using the Script Application Editor. Click **Edit**.

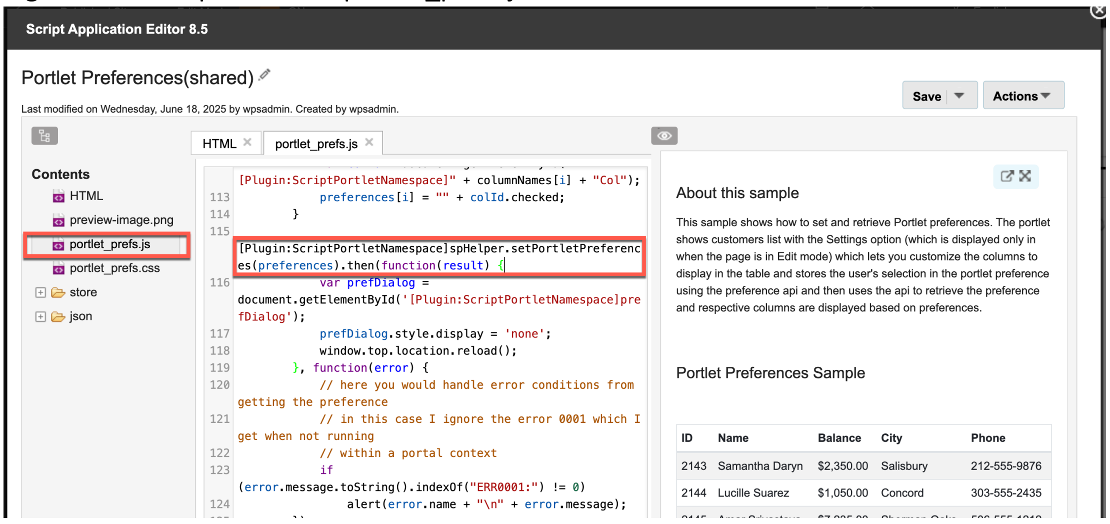


About this sample

This sample shows how to set and retrieve Portlet preferences. The portlet shows customers list with the Settings option (which is displayed only in when the page is in Edit mode) which lets you customize the columns to display in the table and stores the user's selection in the portlet preference using the preference api and then uses the api to retrieve the preference and respective columns are displayed based on preferences.

Portlet Preferences Sample

12. And you see that the __SPNS__ has been converted to the [Plugin:ScriptPortletNamespace]. E.g. in the same place in the portlet_prefs.js file.



Portlet Preferences(shared)

Last modified on Wednesday, June 18, 2025 by wpsadmin. Created by wpsadmin.

```
[Plugin:ScriptPortletNamespace]" + columnNames[i] + "Col");
                preferences[i] = "" + colId.checked;
        }

[Plugin:ScriptPortletNamespace]spHelper.setPortletPreferenc
es(preferences).then(function(result) {
                var prefDialog =
document.getElementById('[Plugin:ScriptPortletNamespace]pre
fDialog');
                prefDialog.style.display = 'none';
                window.top.location.reload();
        }, function(error) {
                // here you would handle error conditions from
getting the preference
                // in this case I ignore the error 0001 which I
get when not running
                // within a portal context
                if
(error.message.toString().indexOf("ERR0001:") != 0)
                    alert(error.name + "\n" + error.message);
```

About this sample

This sample shows how to set and retrieve Portlet preferences. The portlet shows customers list with the Settings option (which is displayed only in when the page is in Edit mode) which lets you customize the columns to display in the table and stores the user's selection in the portlet preference using the preference api and then uses the api to retrieve the preference and respective columns are displayed based on preferences.

Portlet Preferences Sample

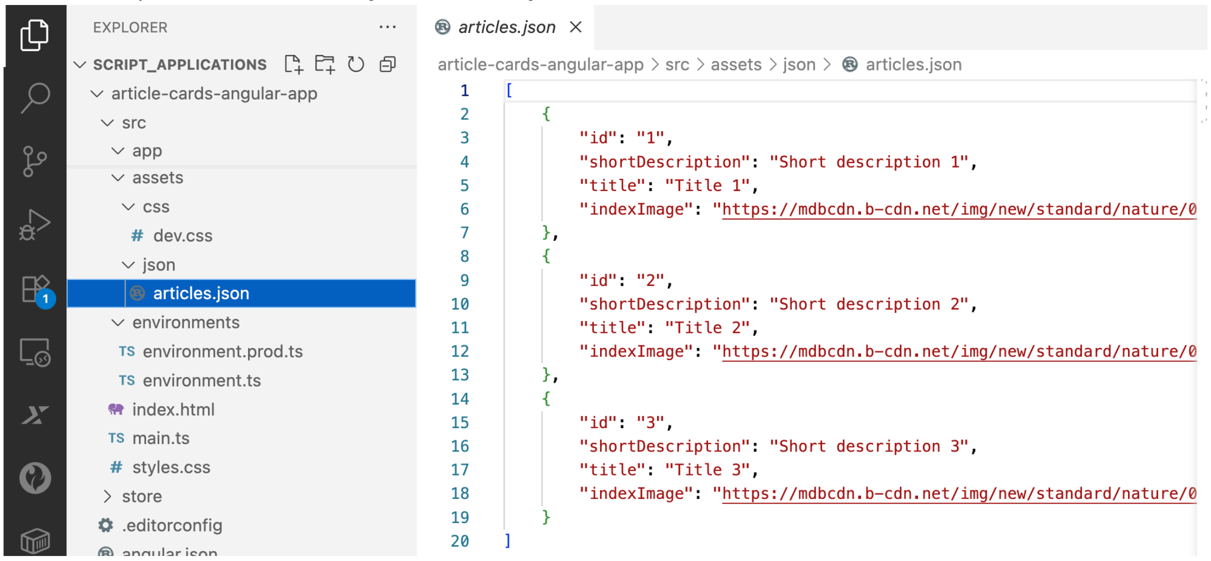
ID	Name	Balance	City	Phone
2143	Samantha Daryn	$2,350.00	Salisbury	212-555-9876
2144	Lucille Suarez	$1,050.00	Concord	303-555-2435

You have successfully deployed and discovered how to use reusable and configurable Script Applications using preferences with the spHelper.

Part 2: Create a new Angular Script Application using DX articles content

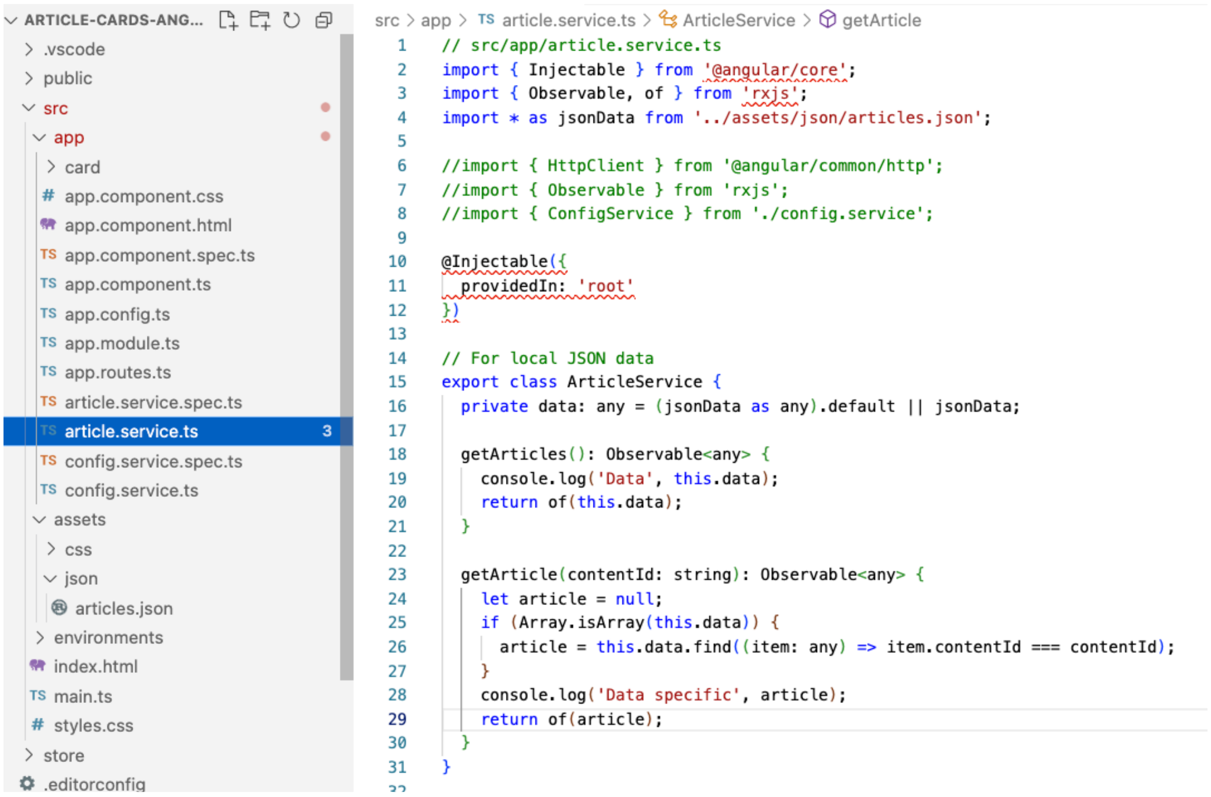
In this part, you will update a new Angular Script Application to use Woodburn Healthcare articles stored in DX and show them in a carousel. You will test it locally first and then deploy it to your DX server and add it to a page.

1. First download the article-cards-angular-app Script Application and open it in your favorite editor (e.g. Visual Studio Code). Have a look at the JSON that manages the content of the carousel. Open the **src/assets/json/articles.json** file.



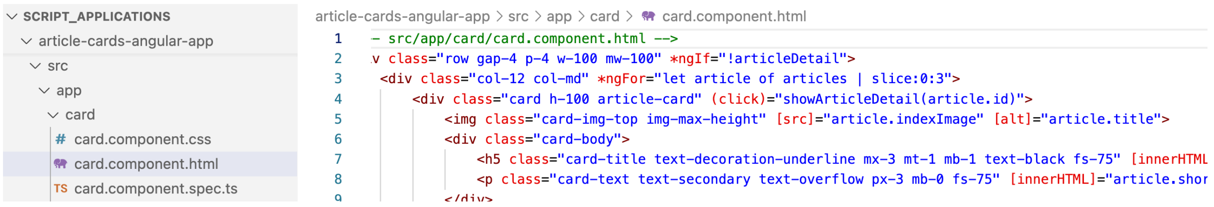
```
[
    {
        "id": "1",
        "shortDescription": "Short description 1",
        "title": "Title 1",
        "indexImage": "https://mdbcdn.b-cdn.net/img/new/standard/nature/0
    },
    {
        "id": "2",
        "shortDescription": "Short description 2",
        "title": "Title 2",
        "indexImage": "https://mdbcdn.b-cdn.net/img/new/standard/nature/0
    },
    {
        "id": "3",
        "shortDescription": "Short description 3",
        "title": "Title 3",
        "indexImage": "https://mdbcdn.b-cdn.net/img/new/standard/nature/0
    }
]
```

2. Have a look at the place where this is read. Open the src/app/article.service.ts file and notice how the getArticles and getArticle methods are constructed.



```
// src/app/article.service.ts
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import * as jsonData from '../assets/json/articles.json';

//import { HttpClient } from '@angular/common/http';
//import { Observable } from 'rxjs';
//import { ConfigService } from './config.service';

@Injectable({
  providedIn: 'root'
})

// For local JSON data
export class ArticleService {
  private data: any = (jsonData as any).default || jsonData;

  getArticles(): Observable<any> {
    console.log('Data', this.data);
    return of(this.data);
  }

  getArticle(contentId: string): Observable<any> {
    let article = null;
    if (Array.isArray(this.data)) {
      article = this.data.find((item: any) => item.contentId === contentId);
    }
    console.log('Data specific', article);
    return of(article);
  }
}
```

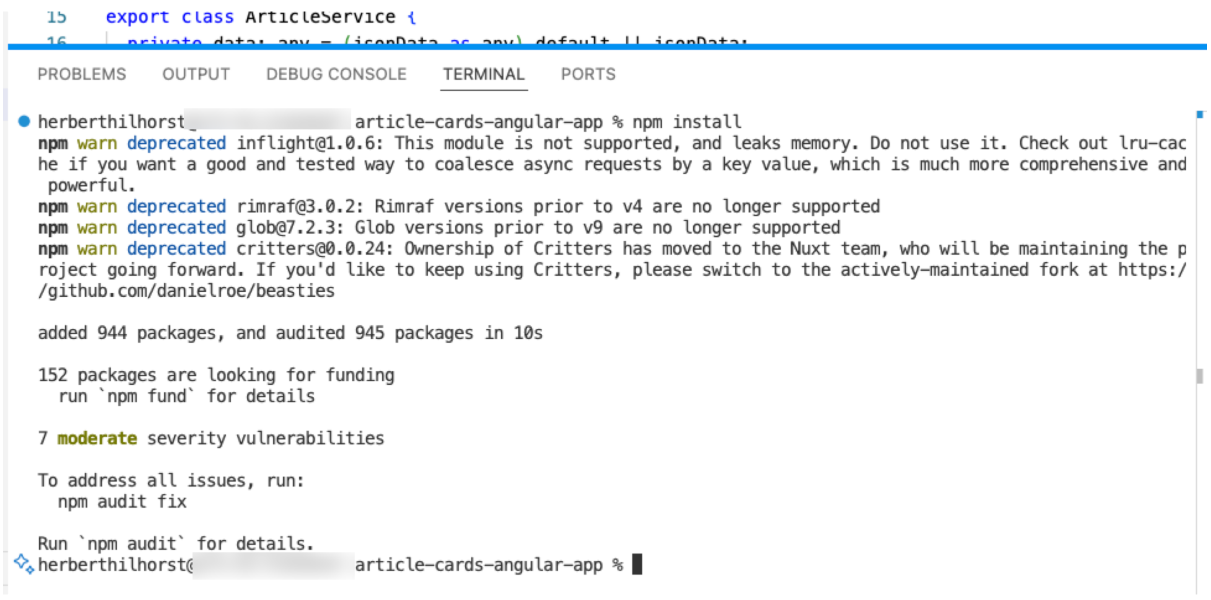
3. And how these are used to create the output in the src/app/card/card.component.html file.


SCRIPT_APPLICATIONS
article-cards-angular-app
src
app
card
card.component.css
card.component.html
card.component.spec.ts
article-cards-angular-app > src > app > card > card.component.html

```
- src/app/card/card.component.html -->
v class="row gap-4 p-4 w-100 mw-100" *ngIf="!articleDetail">
 <div class="col-12 col-md" *ngFor="let article of articles | slice:0:3">
     <div class="card h-100 article-card" (click)="showArticleDetail(article.id)">
         <img class="card-img-top img-max-height" [src]="article.indexImage" [alt]="article.title">
         <div class="card-body">
             <h5 class="card-title text-decoration-underline mx-3 mt-1 mb-1 text-black fs-75" [innerHTML
             <p class="card-text text-secondary text-overflow px-3 mb-0 fs-75" [innerHTML]="article.shor
         </div>
```

4. Then run the application locally first. Open a Terminal and run the following command.

```
npm install
```

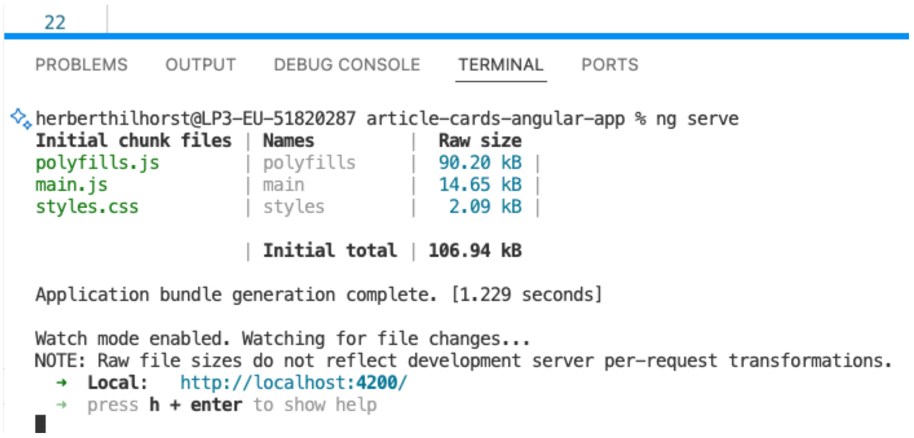


```
15    export class ArticleService {
16        private data: any = (jsonData as any).default || jsonData;
```

```
herberthilhorst[redacted] article-cards-angular-app % npm install
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cac
he if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and
 powerful.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated critters@0.0.24: Ownership of Critters has moved to the Nuxt team, who will be maintaining the p
roject going forward. If you'd like to keep using Critters, please switch to the actively-maintained fork at https:/
/github.com/danielroe/beasties

added 944 packages, and audited 945 packages in 10s

152 packages are looking for funding
  run `npm fund` for details

7 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
herberthilhorst@[redacted] article-cards-angular-app % ▌
```

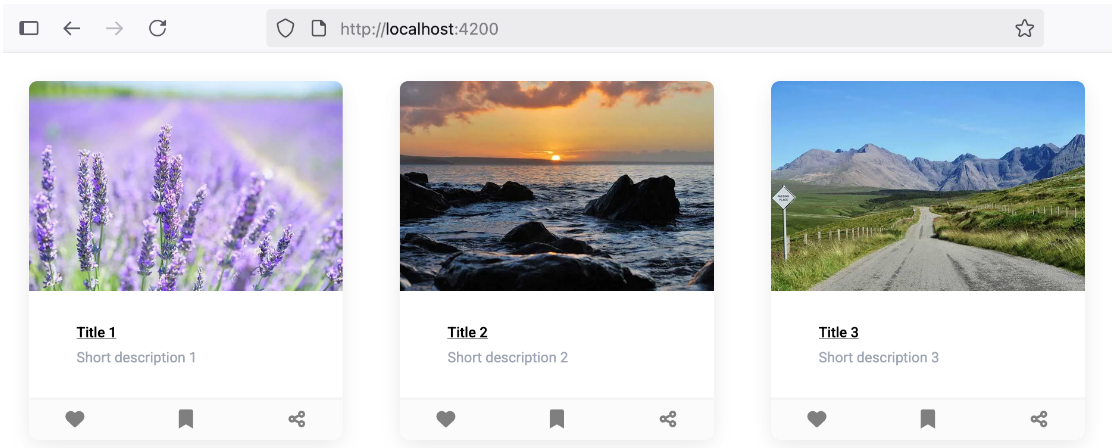
5. Run the following command to start a local server.

```
ng serve
```

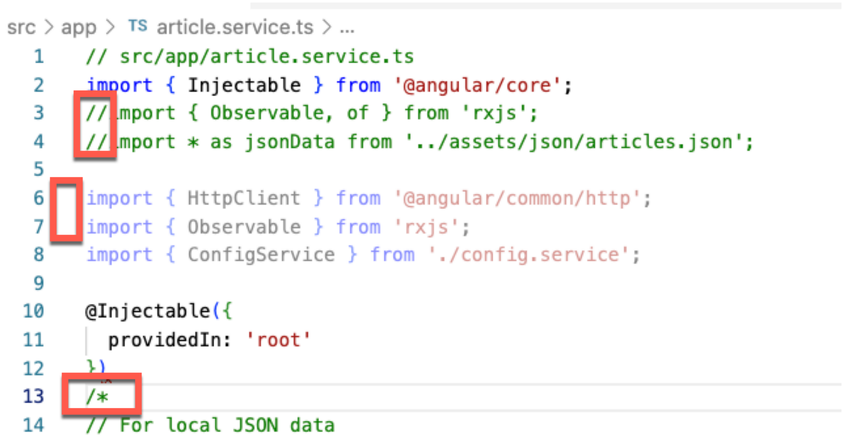


```
herberthilhorst@LP3-EU-51820287 article-cards-angular-app % ng serve
Initial chunk files | Names         |  Raw size
polyfills.js        | polyfills     |  90.20 kB |
main.js             | main          |  14.65 kB |
styles.css          | styles        |   2.09 kB |

                    | Initial total | 106.94 kB

Application bundle generation complete. [1.229 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
▌
```

6. To test the application, navigate to your local server http://localhost:4200/ and you should see the application output. The application will automatically reload if you change any of the source files.



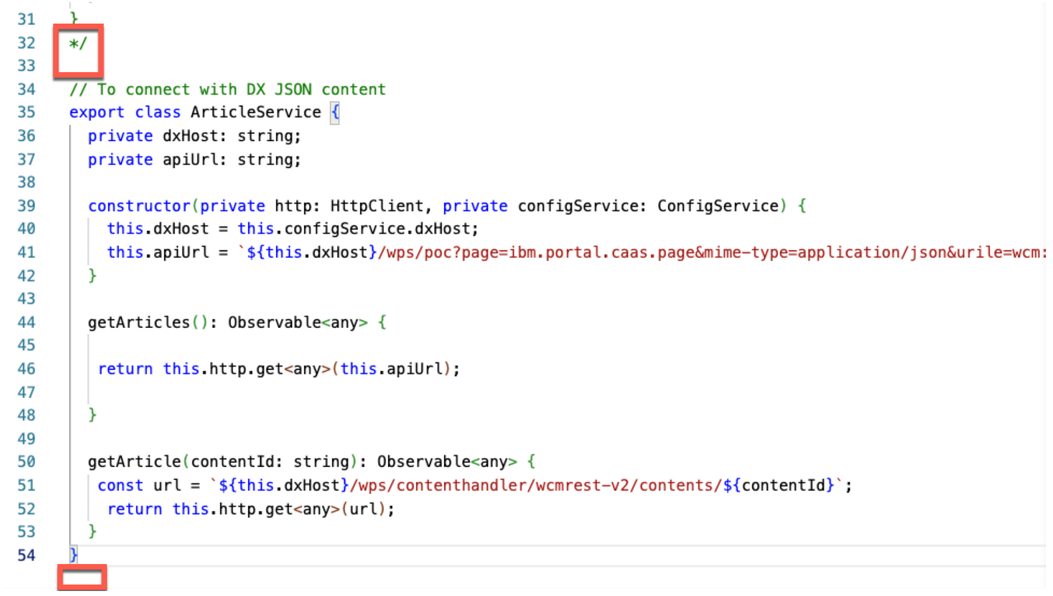
7. Now update it to work with HCL DX content. The code is already available and commented out. Hence comment out the code for the local JSON use (add // before lines 3 and 4), uncomment lines 6 and 7 to use of DX JSON content and comment out the script for local JSON data (add /* in line 13).



src > app > TS article.service.ts > ...

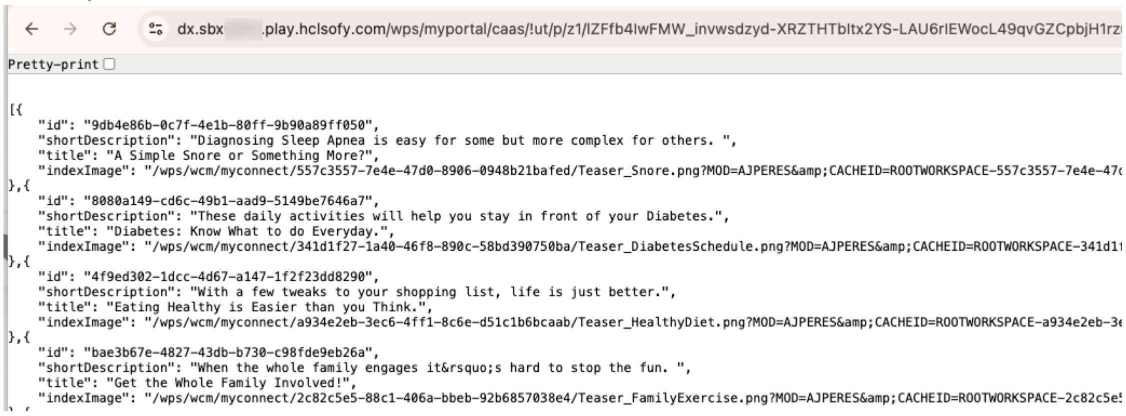
```
1   // src/app/article.service.ts
2   import { Injectable } from '@angular/core';
3   //import { Observable, of } from 'rxjs';
4   //import * as jsonData from '../assets/json/articles.json';
5
6   import { HttpClient } from '@angular/common/http';
7   import { Observable } from 'rxjs';
8   import { ConfigService } from './config.service';
9
10  @Injectable({
11    providedIn: 'root'
12  })
13  /*
14  // For local JSON data
```

8. At the end of the local JSON code end the commenting out (add */ on line 32) and comment out the use of DX JSON content (remove the /* from line 33 and */ from line 55). Notice how the same getArticles method now uses a DX Content as a Service (CaaS) URL and getArticle uses the WCM V2 API.



```
31    }
32    */
33
34    // To connect with DX JSON content
35    export class ArticleService {
36      private dxHost: string;
37      private apiUrl: string;
38
39      constructor(private http: HttpClient, private configService: ConfigService) {
40        this.dxHost = this.configService.dxHost;
41        this.apiUrl = `${this.dxHost}/wps/poc?page=ibm.portal.caas.page&mime-type=application/json&urile=wcm:
42      }
43
44      getArticles(): Observable<any> {
45
46        return this.http.get<any>(this.apiUrl);
47
48      }
49
50      getArticle(contentId: string): Observable<any> {
51        const url = `${this.dxHost}/wps/contenthandler/wcmrest-v2/contents/${contentId}`;
52         return this.http.get<any>(url);
53      }
54    }
```

9. You may try the CaaS page to see what DX content it shows. It is constructed from your SoFy DX server host and the CaaS part /wps/poc?page=ibm.portal.caas.page&mime-type=application/json&urile=wcm:path:/woodburn%20healthcare%20design/json/articles.json. E.g. https://dx.sbx0000.play.hclsofy.com/wps/poc?page=ibm.portal.caas.page&mime-type=application/json&urile=wcm:path:/woodburn%20healthcare%20design/json/articles.json. If you do not have this JSON yet, you may use the HDX-DEV-200 Web Content Development lab to create it.

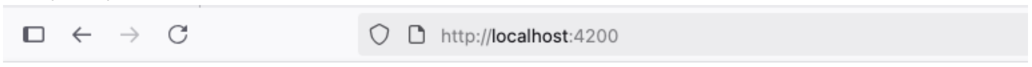


dx.sbx[redacted].play.hclsofy.com/wps/myportal/caas/!ut/p/z1/lZFfb4lwFMW_invwsdzyd-XRZTHTbltx2YS-LAU6rlEWocL49qvGZCpbjH1rz

Pretty-print ☐

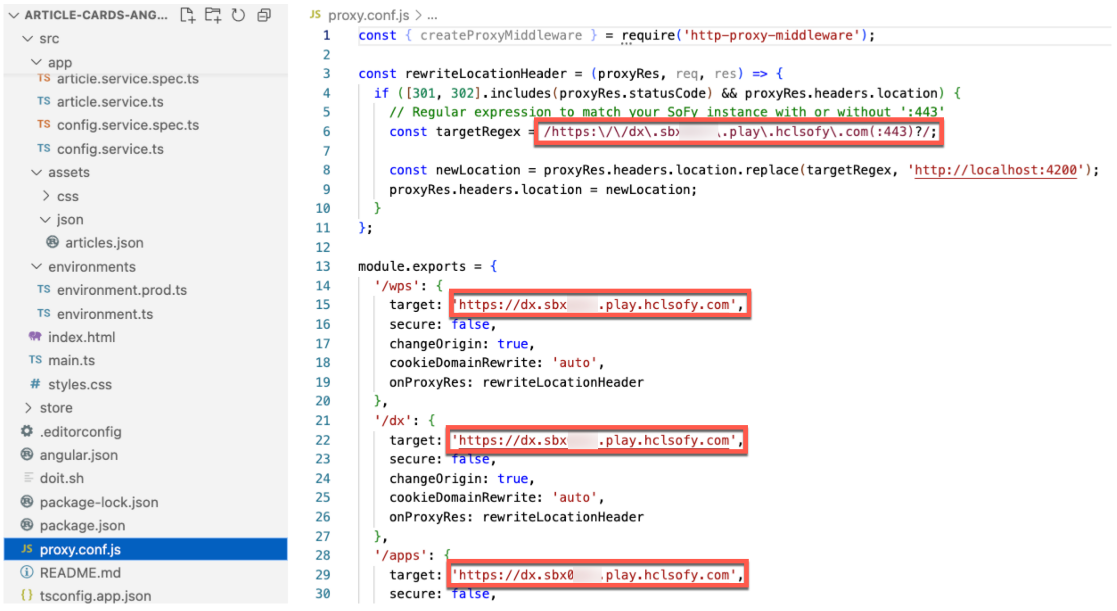
```
[{
    "id": "9db4e86b-0c7f-4e1b-80ff-9b90a89ff050",
    "shortDescription": "Diagnosing Sleep Apnea is easy for some but more complex for others. ",
    "title": "A Simple Snore or Something More?",
    "indexImage": "/wps/wcm/myconnect/557c3557-7e4e-47d0-8906-0948b21bafed/Teaser_Snore.png?MOD=AJPERES&amp;CACHEID=ROOTWORKSPACE-557c3557-7e4e-47
},{
    "id": "8080a149-cd6c-49b1-aad9-5149be7646a7",
    "shortDescription": "These daily activities will help you stay in front of your Diabetes.",
    "title": "Diabetes: Know What to do Everyday.",
    "indexImage": "/wps/wcm/myconnect/341d1f27-1a40-46f8-890c-58bd390750ba/Teaser_DiabetesSchedule.png?MOD=AJPERES&amp;CACHEID=ROOTWORKSPACE-341d1
},{
    "id": "4f9ed302-1dcc-4d67-a147-1f2f23dd8290",
    "shortDescription": "With a few tweaks to your shopping list, life is just better.",
    "title": "Eating Healthy is Easier than you Think.",
    "indexImage": "/wps/wcm/myconnect/a934e2eb-3ec6-4ff1-8c6e-d51c1b6bcaab/Teaser_HealthyDiet.png?MOD=AJPERES&amp;CACHEID=ROOTWORKSPACE-a934e2eb-3
},{
    "id": "bae3b67e-4827-43db-b730-c98fde9eb26a",
    "shortDescription": "When the whole family engages it&rsquo;s hard to stop the fun. ",
    "title": "Get the Whole Family Involved!",
    "indexImage": "/wps/wcm/myconnect/2c82c5e5-88c1-406a-bbeb-92b6857038e4/Teaser_FamilyExercise.png?MOD=AJPERES&amp;CACHEID=ROOTWORKSPACE-2c82c5e
```

10. Then save the file **src/app/article.service.ts** and you will notice that the server gets updated automatically and now shows nothing anymore. This is because the DX content is blocked by the proxy.



http://localhost:4200

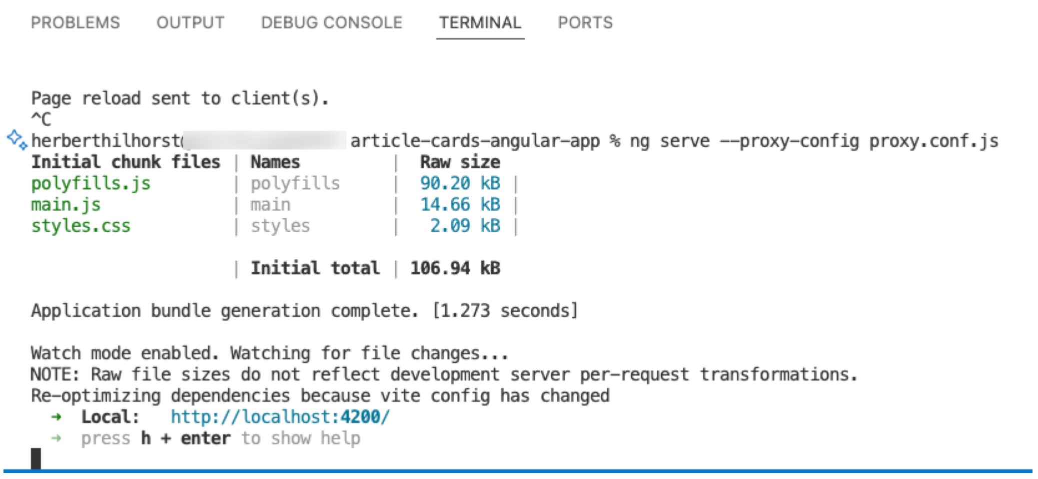
11. For local use of the DX Content, you need to configure and use the proxy. This has been set up for you in the proxy.conf.js file. You need to update this to match your DX server. If you are using HCL SoFy, just update the part of the host with sbx0000 to yours and save the file.



```
const { createProxyMiddleware } = require('http-proxy-middleware');

const rewriteLocationHeader = (proxyRes, req, res) => {
  if ([301, 302].includes(proxyRes.statusCode) && proxyRes.headers.location) {
    // Regular expression to match your SoFy instance with or without ':443'
    const targetRegex = /https:\/\/dx\.sb[redacted]\.play\.hclsofy\.com(:443)?/;

    const newLocation = proxyRes.headers.location.replace(targetRegex, 'http://localhost:4200');
    proxyRes.headers.location = newLocation;
  }
};

module.exports = {
  '/wps': {
    target: 'https://dx.sbx[redacted].play.hclsofy.com',
    secure: false,
    changeOrigin: true,
    cookieDomainRewrite: 'auto',
    onProxyRes: rewriteLocationHeader
  },
  '/dx': {
    target: 'https://dx.sbx[redacted].play.hclsofy.com',
    secure: false,
    changeOrigin: true,
    cookieDomainRewrite: 'auto',
    onProxyRes: rewriteLocationHeader
  },
  '/apps': {
    target: 'https://dx.sbx0[redacted].play.hclsofy.com',
    secure: false,
```

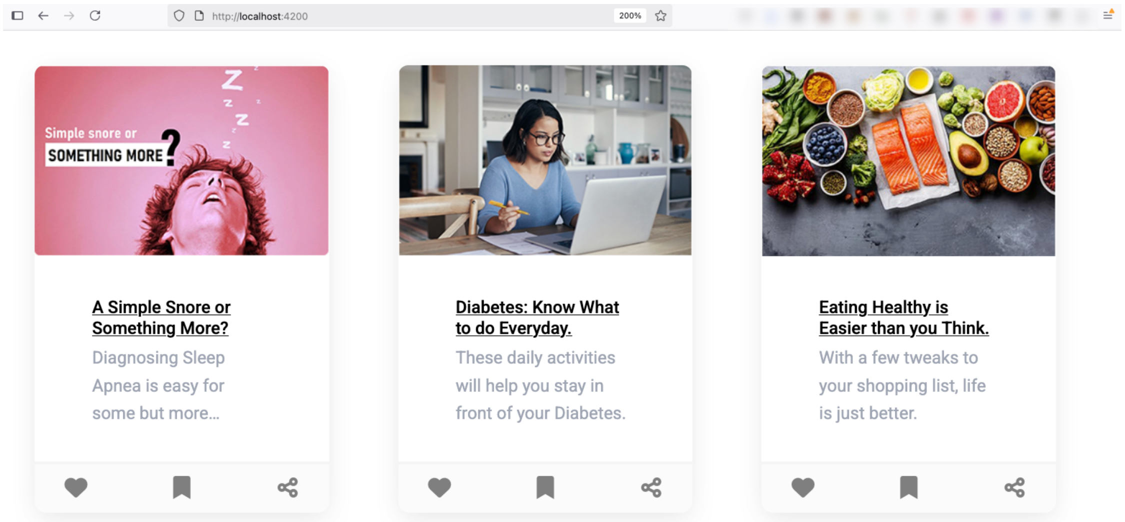
12. Then you need to run your local server with this proxy. Stop the existing server (CTRL C) and then start it again using the following command.

```
ng serve --proxy-config proxy.conf.js
```



```
Page reload sent to client(s).
^C
herberthilhorst[redacted]article-cards-angular-app % ng serve --proxy-config proxy.conf.js
Initial chunk files | Names         |  Raw size |
polyfills.js        | polyfills     |  90.20 kB |
main.js             | main          |  14.66 kB |
styles.css          | styles        |   2.09 kB |

                    | Initial total | 106.94 kB

Application bundle generation complete. [1.273 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
Re-optimizing dependencies because vite config has changed
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
```

13. You should now see the application load the articles from DX.

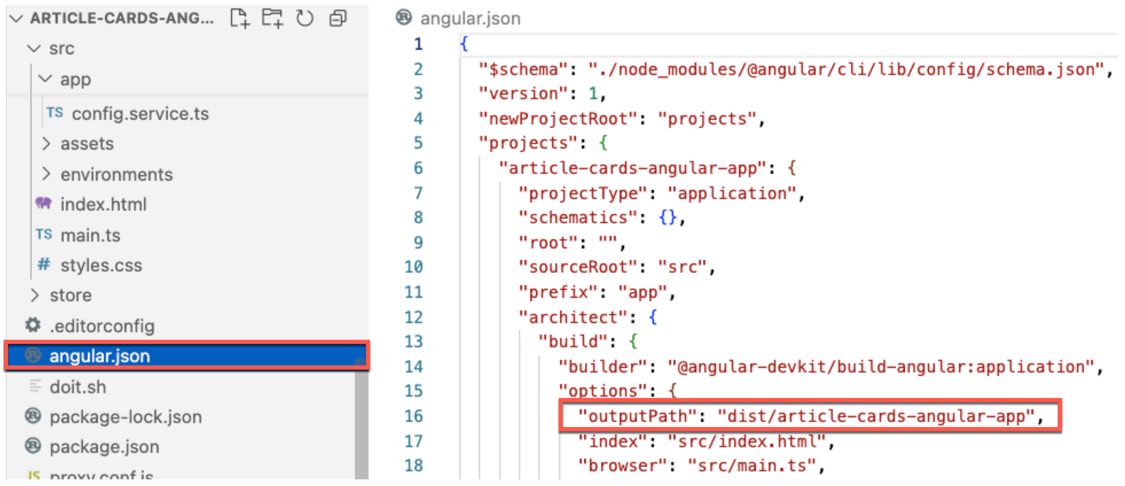


A Simple Snore or Something More?
Diagnosing Sleep Apnea is easy for some but more…

Diabetes: Know What to do Everyday.
These daily activities will help you stay in front of your Diabetes.

Eating Healthy is Easier than you Think.
With a few tweaks to your shopping list, life is just better.

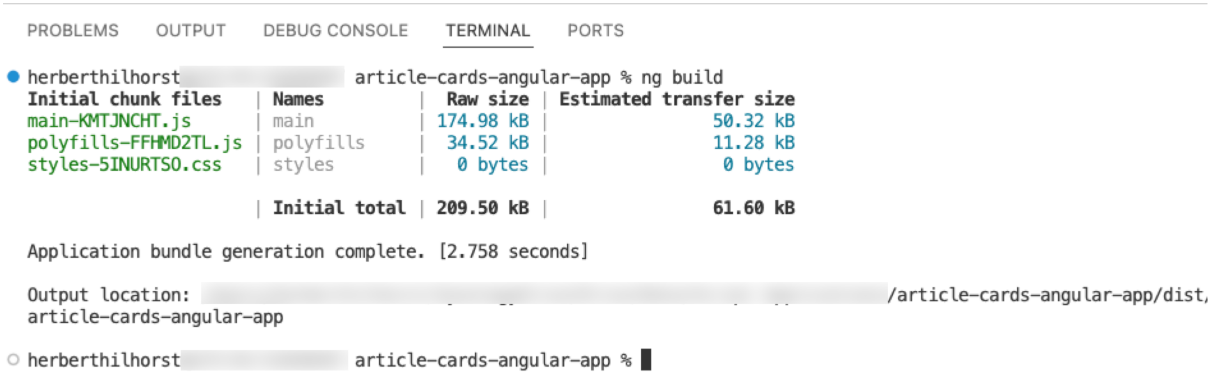
14. Now you will build the application. This uses the Angular configuration managed in the **angular.json** file. Open it to see that it is going to be built in the **dist/article-cards-angular-app** folder, managed by the build, options, and outputPath here.



```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "article-cards-angular-app": {
      "projectType": "application",
      "schematics": {},
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:application",
          "options": {
            "outputPath": "dist/article-cards-angular-app",
            "index": "src/index.html",
            "browser": "src/main.ts",
```

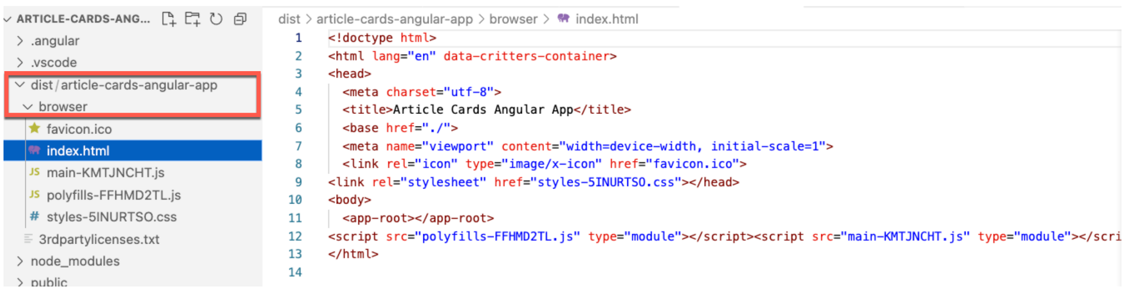
15. Run the build command.

```
ng build
```



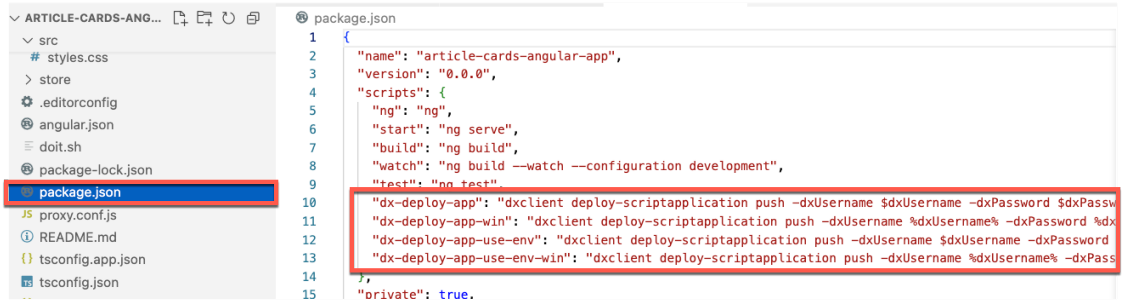
```
herberthilhorst[redacted] article-cards-angular-app % ng build
Initial chunk files     | Names         |  Raw size | Estimated transfer size
main-KMTJNCHT.js        | main          | 174.98 kB |                50.32 kB
polyfills-FFHMD2TL.js   | polyfills     |  34.52 kB |                11.28 kB
styles-5INURTSO.css     | styles        |   0 bytes |                 0 bytes

                        | Initial total | 209.50 kB |                61.60 kB

Application bundle generation complete. [2.758 seconds]

Output location: [redacted]/article-cards-angular-app/dist/
article-cards-angular-app

herberthilhorst[redacted] article-cards-angular-app % 
```

16. This creates the **dist/article-cards-angular-app** folder for your application with a browser sub-folder and all the minified resources.



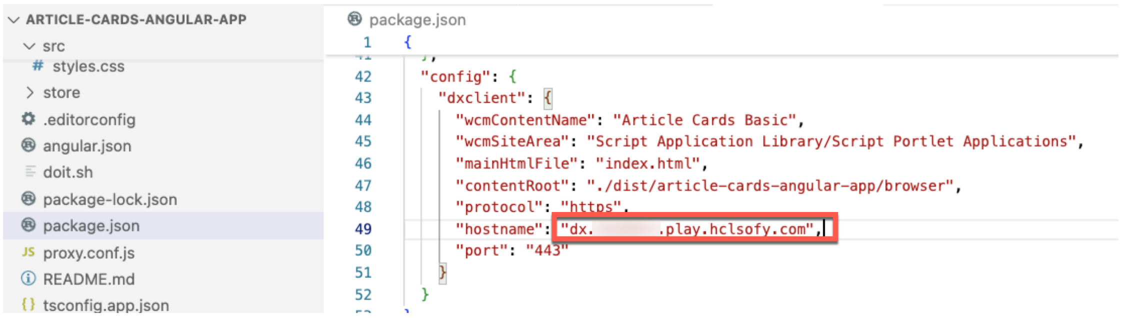
```
dist > article-cards-angular-app > browser > index.html
<!doctype html>
<html lang="en" data-critters-container>
<head>
  <meta charset="utf-8">
  <title>Article Cards Angular App</title>
  <base href="./">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
<link rel="stylesheet" href="styles-5INURTSO.css"></head>
<body>
  <app-root></app-root>
<script src="polyfills-FFHMD2TL.js" type="module"></script><script src="main-KMTJNCHT.js" type="module"></scri
</html>
```

17. Deploy this new Angular application to your DX server. You will use the Webpack for this, which is partly preconfigured for you. Open the **package.json** and check under the scripts section. There are several DX Deploy scripts that use the DXClient deploy-scriptapplication push command.



```
package.json
{
  "name": "article-cards-angular-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "dx-deploy-app": "dxclient deploy-scriptapplication push -dxUsername $dxUsername -dxPassword $dxPassw
    "dx-deploy-app-win": "dxclient deploy-scriptapplication push -dxUsername %dxUsername% -dxPassword %dx
    "dx-deploy-app-use-env": "dxclient deploy-scriptapplication push -dxUsername $dxUsername -dxPassword
    "dx-deploy-app-use-env-win": "dxclient deploy-scriptapplication push -dxUsername %dxUsername% -dxPass
  },
  "private": true,
```

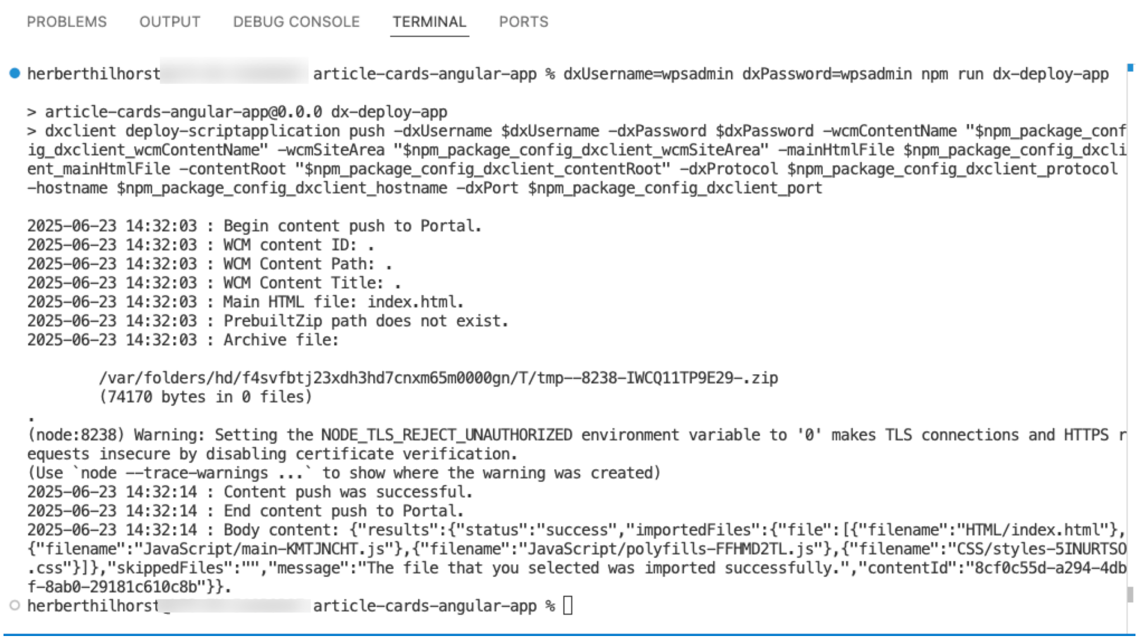
18. If you scroll down further, you will notice the parameter configuration for the DXClient command for this Script Application. It stores the name of the Script Application in wcmContentName, the web content library and site area to store it under wcmSiteArea and the contentRoot of the Angular content. You need to update the hostname to match your DX server one and then save the file.



```
package.json
{
  "config": {
    "dxclient": {
      "wcmContentName": "Article Cards Basic",
      "wcmSiteArea": "Script Application Library/Script Portlet Applications",
      "mainHtmlFile": "index.html",
      "contentRoot": "./dist/article-cards-angular-app/browser",
      "protocol": "https",
      "hostname": "dx.[redacted].play.hclsofy.com",
      "port": "443"
    }
  }
```

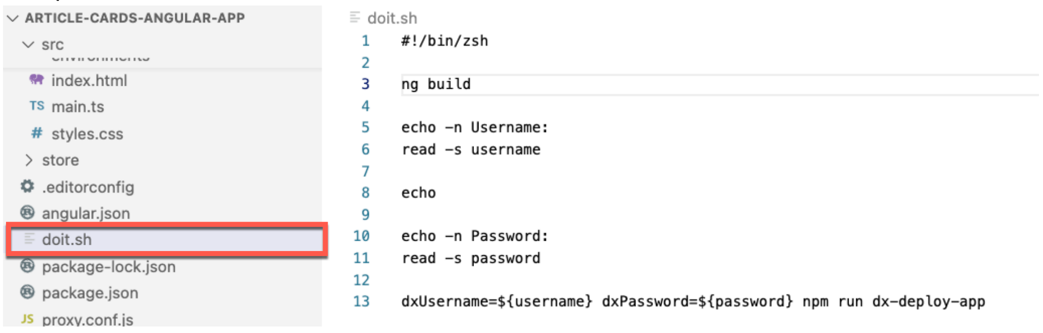
19. You may deploy the application using the dx-deploy-app script for Mac and Linux or dx-deploy-win for Windows. Notice that the user's name and password are not set. To deploy it set them first and then run the appropriate deploy target. E.g. on a Mac, using a user that has the rights to create this content in the selected library and site area, such as the wpsadmin user with the corresponding password wpsadmin:

```
dxUsername=wpsadmin dxPassword=wpsadmin npm run dx-deploy-app
```



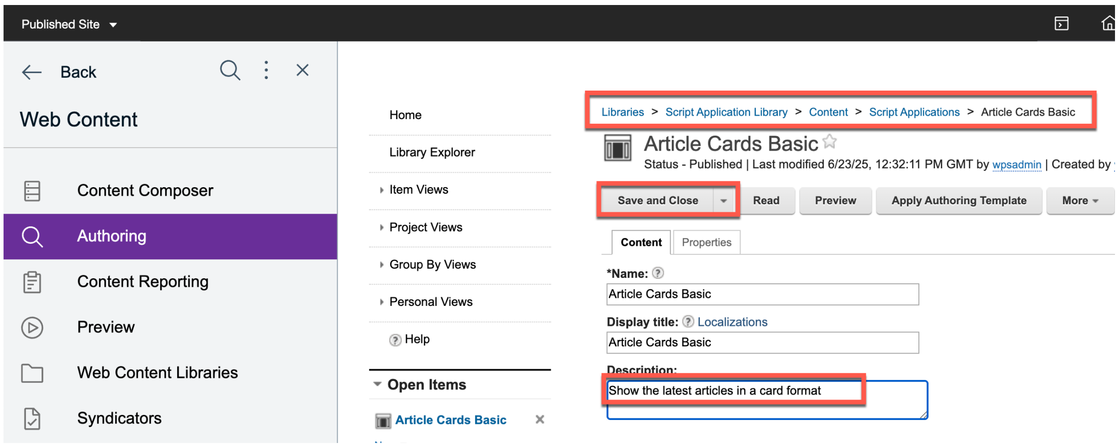
```
● herberthilhorst[redacted] article-cards-angular-app % dxUsername=wpsadmin dxPassword=wpsadmin npm run dx-deploy-app

> article-cards-angular-app@0.0.0 dx-deploy-app
> dxclient deploy-scriptapplication push -dxUsername $dxUsername -dxPassword $dxPassword -wcmContentName "$npm_package_conf
ig_dxclient_wcmContentName" -wcmSiteArea "$npm_package_config_dxclient_wcmSiteArea" -mainHtmlFile $npm_package_config_dxcli
ent_mainHtmlFile -contentRoot "$npm_package_config_dxclient_contentRoot" -dxProtocol $npm_package_config_dxclient_protocol
-hostname $npm_package_config_dxclient_hostname -dxPort $npm_package_config_dxclient_port

2025-06-23 14:32:03 : Begin content push to Portal.
2025-06-23 14:32:03 : WCM content ID: .
2025-06-23 14:32:03 : WCM Content Path: .
2025-06-23 14:32:03 : WCM Content Title: .
2025-06-23 14:32:03 : Main HTML file: index.html.
2025-06-23 14:32:03 : PrebuiltZip path does not exist.
2025-06-23 14:32:03 : Archive file:

        /var/folders/hd/f4svfbtj23xdh3hd7cnxm65m0000gn/T/tmp--8238-IWCQ11TP9E29-.zip
        (74170 bytes in 0 files)
.
(node:8238) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS r
equests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
2025-06-23 14:32:14 : Content push was successful.
2025-06-23 14:32:14 : End content push to Portal.
2025-06-23 14:32:14 : Body content: {"results":{"status":"success","importedFiles":{"file":[{"filename":"HTML/index.html"},
{"filename":"JavaScript/main-KMTJNCHT.js"},{"filename":"JavaScript/polyfills-FFHMD2TL.js"},{"filename":"CSS/styles-5INURTSO
.css"}]},"skippedFiles":"","message":"The file that you selected was imported successfully.","contentId":"8cf0c55d-a294-4db
f-8ab0-29181c610c8b"}}.
○ herberthilhorst[redacted] article-cards-angular-app % 
```

20. Notice that the local file doit.sh allows you to do these two steps and prompts the username and password.



```
#!/bin/zsh

ng build

echo -n Username:
read -s username

echo

echo -n Password:
read -s password

dxUsername=${username} dxPassword=${password} npm run dx-deploy-app
```

21. Have a look at how this is stored in the Script Application Library. Open the **Script Application Library, Content, Script Applications, Angular Cards Basic** content and have a look at the new elements that have been added. You can edit the description that appears when looking at this page component. Enter a clear description for your content authors, like **Show the latest articles in a card format** and save it. Click **Save and Close**.



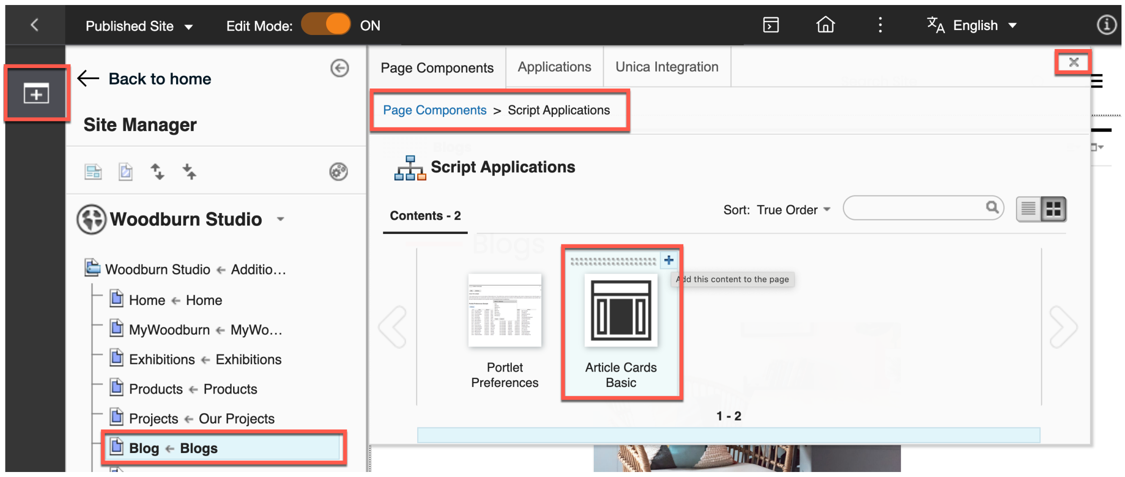
Status - Published | Last modified 6/23/25, 12:32:11 PM GMT by wpsadmin | Created by

*Name: Article Cards Basic

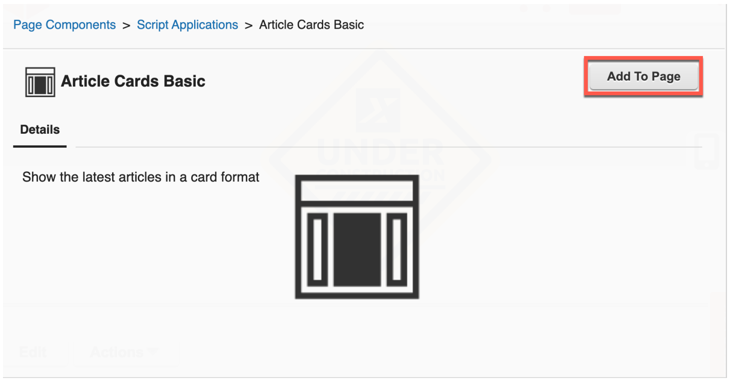
Display title: Localizations Article Cards Basic

Description: Show the latest articles in a card format

22. Finally, add it to a page to test it. You may add it also to the Blogs or Promotion page, as you did in the earlier exercise. Select it first to see the details.

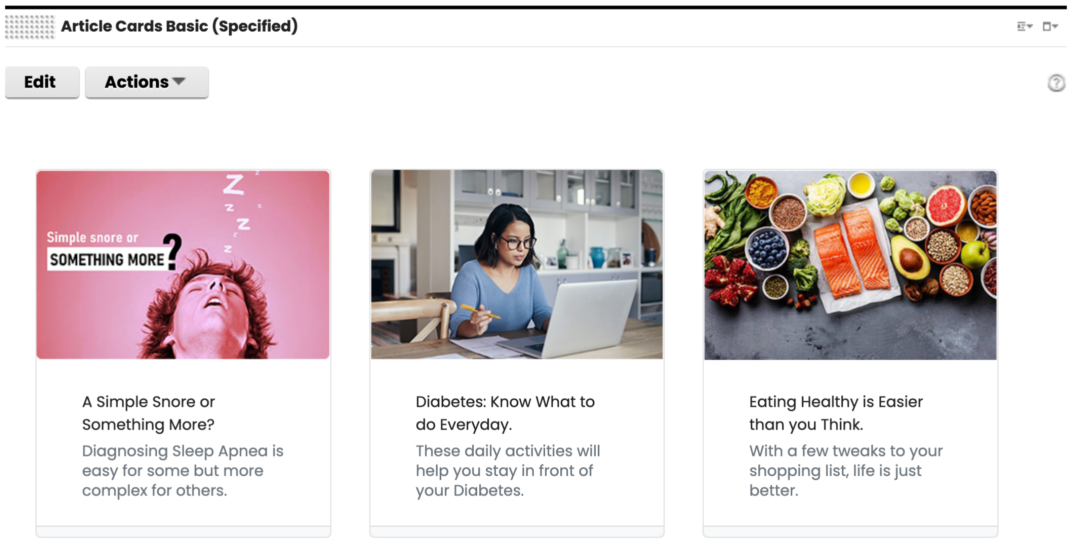


23. You can see the description you added under the Details section here. Now add it to the page. Click **Add To Page**.



Show the latest articles in a card format

24. Scroll down to see your new Script Application working. Click **Edit** to see its code.

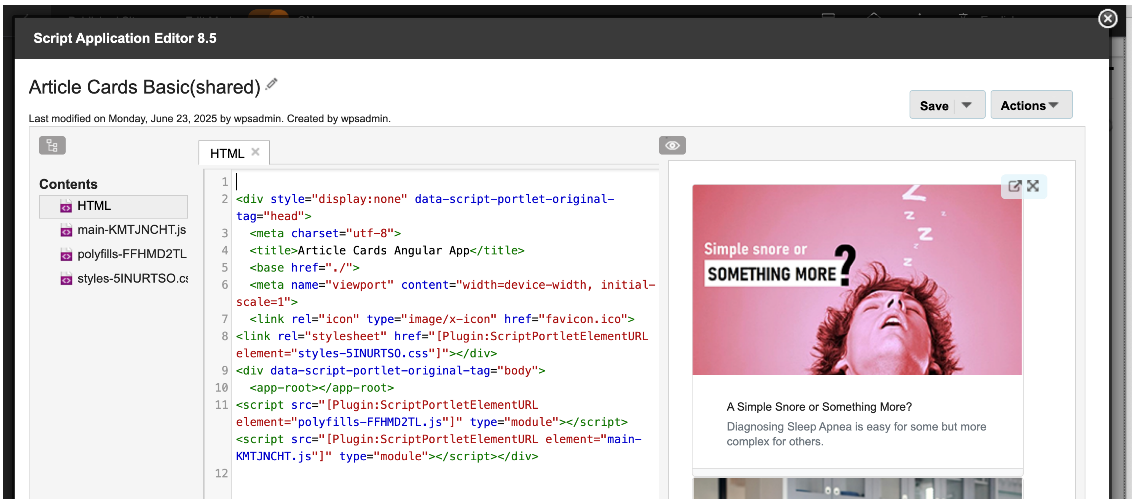


A Simple Snore or Something More?
Diagnosing Sleep Apnea is easy for some but more complex for others.

Diabetes: Know What to do Everyday.
These daily activities will help you stay in front of your Diabetes.

Eating Healthy is Easier than you Think.
With a few tweaks to your shopping list, life is just better.

25. And notice the different files from the build's dist directory.



Last modified on Monday, June 23, 2025 by wpsadmin. Created by wpsadmin.

```html

<div style="display:none" data-script-portlet-original-tag="head">
  <meta charset="utf-8">
  <title>Article Cards Angular App</title>
  <base href="./">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
<link rel="stylesheet" href="[Plugin:ScriptPortletElementURL element="styles-5INURTSO.css"]"></div>
<div data-script-portlet-original-tag="body">
  <app-root></app-root>
<script src="[Plugin:ScriptPortletElementURL element="polyfills-FFHMD2TL.js"]" type="module"></script><script src="[Plugin:ScriptPortletElementURL element="main-KMTJNCHT.js"]" type="module"></script></div>

```

A Simple Snore or Something More?
Diagnosing Sleep Apnea is easy for some but more complex for others.

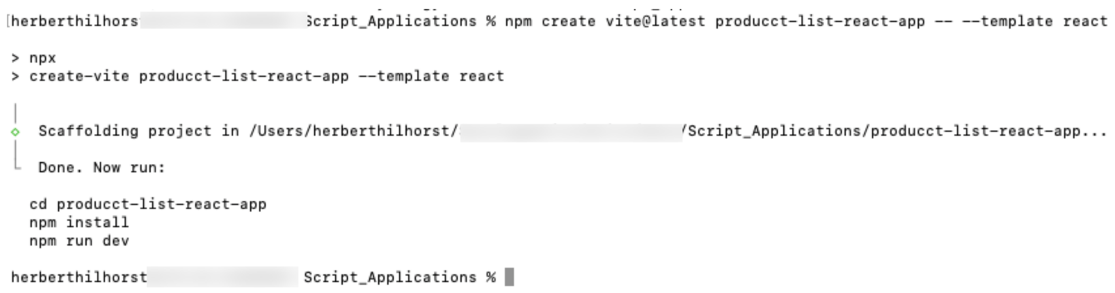
Congratulations! You have now learned how to create a new Angular application that uses DX content and Webpack to simply test, build and deploy to your DX server.

Part 3: Create a new React Script Application that manage product data stored in HCL Leap

You will create new React Product List and Product Details Script Applications that work with HCL Leap application data. You will first test them locally, then build minified versions, deploy them to the server, add them to a DX page and finally have them communicate with each together.

1. You will use one of the many ways to create React applications, using https://vite.dev. One of the ways to create the React application is using npm create create vite@latest <name of your application> -- --template react.
   First create the Product List React application with the name **product-list-react-app**. Go to your directory where you develop your Script Applications and create this new application using the following command in a terminal.

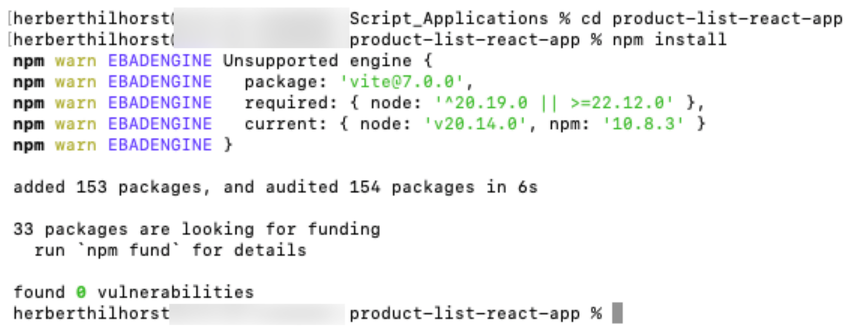
```
npm create vite@latest product-list-react-app -- --template react
```



```
herberthilhors[redacted]Script_Applications % npm create vite@latest producct-list-react-app -- --template react

> npx
> create-vite producct-list-react-app --template react

|
◇  Scaffolding project in /Users/herberthilhorst/[redacted]/Script_Applications/producct-list-react-app...
|
└  Done. Now run:

  cd producct-list-react-app
  npm install
  npm run dev

herberthilhorst[redacted] Script_Applications % ▌
```

2. As suggested by the generator, go to this new directory and install it, using the commands:

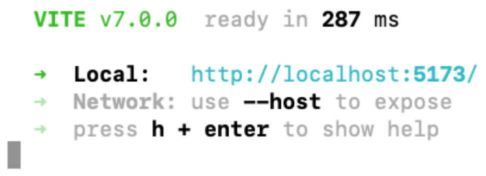
```
cd product-list-react-app
npm install
```



```
herberthilhorst([redacted] Script_Applications % cd product-list-react-app
herberthilhorst([redacted] product-list-react-app % npm install
npm warn EBADENGINE Unsupported engine {
npm warn EBADENGINE   package: 'vite@7.0.0',
npm warn EBADENGINE   required: { node: '^20.19.0 || >=22.12.0' },
npm warn EBADENGINE   current: { node: 'v20.14.0', npm: '10.8.3' }
npm warn EBADENGINE }

added 153 packages, and audited 154 packages in 6s

33 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
herberthilhorst[redacted] product-list-react-app % ▌
```

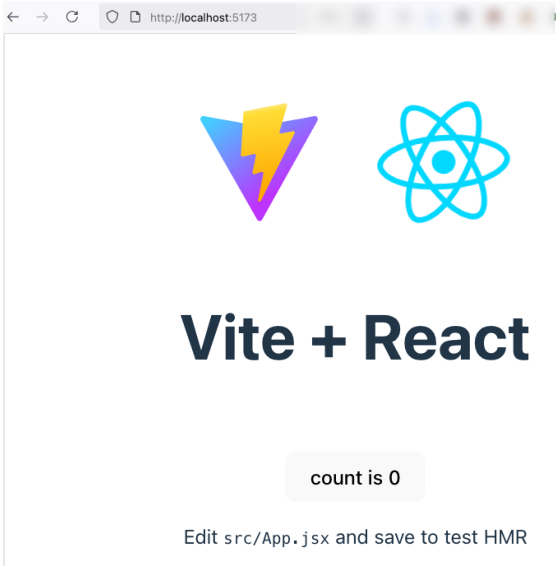
3. Run it on your local development server, using the command:

```
npm run dev
```



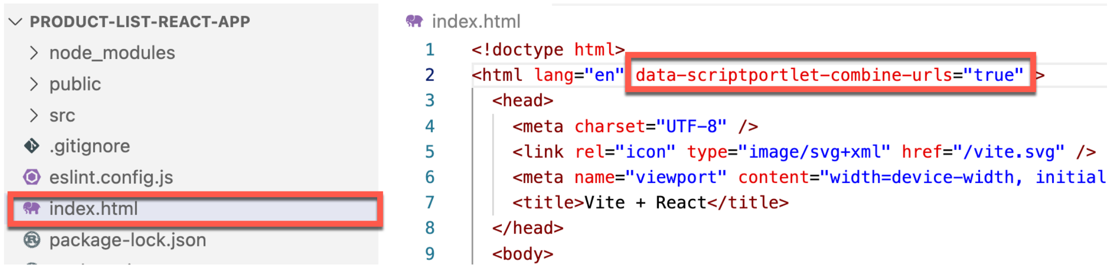
```
  VITE v7.0.0  ready in 287 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
▌
```

4. Open a browser with this URL https://localhost:5173/. As you can see, the code is located in the src/App.jsx. You may click on the **count is 0** button to have it count.



Edit `src/App.jsx` and save to test HMR

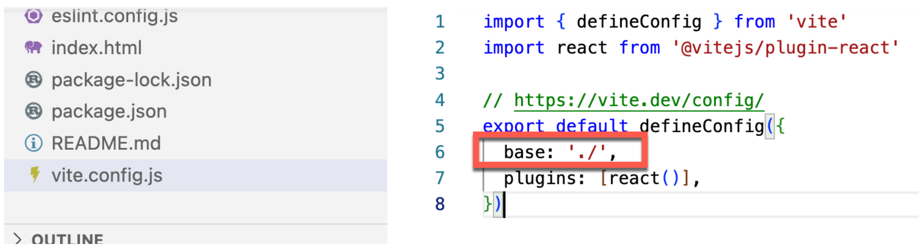
5. Open your new application in your favorite code editor, in this lab, we have used Microsoft Visual Studio Code. To turn this into a reusable DX Script Application, there are a few things you need to do. First, you need to ensure you can combine and access all the resources in this application, once it is deployed. It is important that in the build directory, all URLs that point to files in the application are relative (e.g. ./src/assets/apps.css) and not absolute (like /src/assets/apps.css). To change all these relative URLs to the right URLs in a Script Application, add **data-scriptportlet-combine-urls="true"** to the html tag, typically in the index.html. In this application, edit the **index.html** and edit it, as shown.



```
<!doctype html>
<html lang="en" data-scriptportlet-combine-urls="true" >
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial
    <title>Vite + React</title>
  </head>
  <body>
```

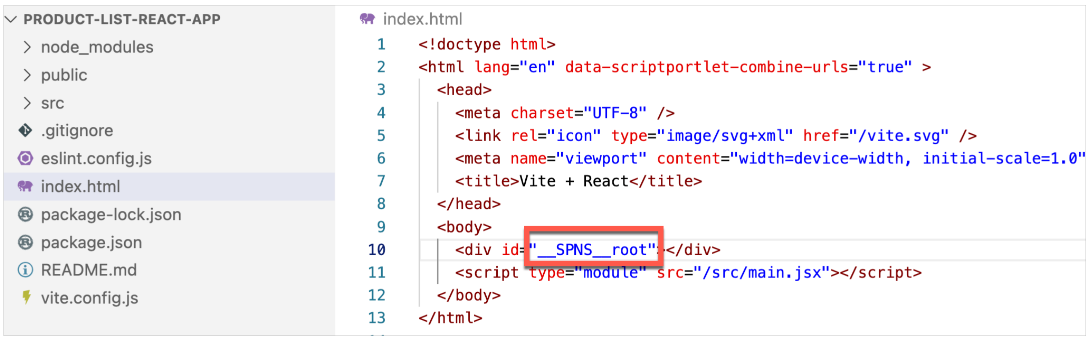
6. To ensure this Vite React application produces relative URLs, you also need to update the vite.config.js, add as shown and save it. Note that this may be managed differently for other React applications.

```
base: './',
```



```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  base: './',
  plugins: [react()],
})
```

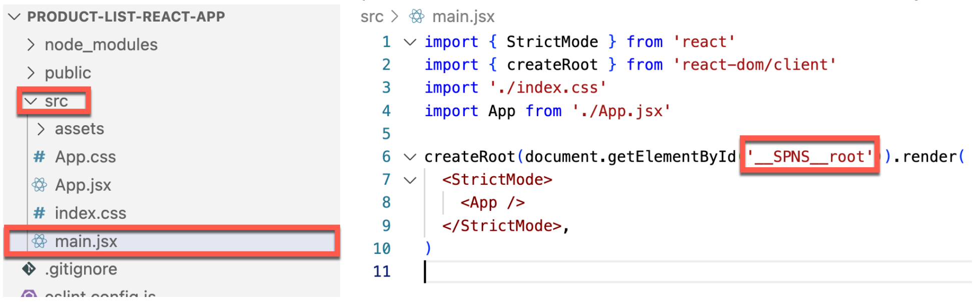
7. Each React application uses a root id in the body section. To allow you to put multiple DX Script Applications on the same page, this id must be unique. You may easily do this using the Script Application namespace. Update the **index.html** again and this time add the Script Application namespace before the **root id**. This is done locally managed using **__SPNS__**, as shown:



```
PRODUCT-LIST-REACT-APP
  > node_modules
  > public
  > src
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
```

```html
<!doctype html>
<html lang="en" data-scriptportlet-combine-urls="true" >
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <title>Vite + React</title>
  </head>
  <body>
    <div id="__SPNS__root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

8. This root id also needs to be updated in the **createRoot** method in **src/main.jsx**, as shown.

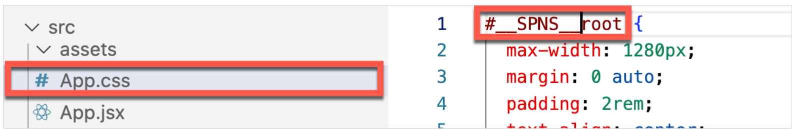


```
PRODUCT-LIST-REACT-APP
  > node_modules
  > public
  v src
    > assets
    App.css
    App.jsx
    index.css
    main.jsx
  .gitignore
```

src > main.jsx

```jsx
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('__SPNS__root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

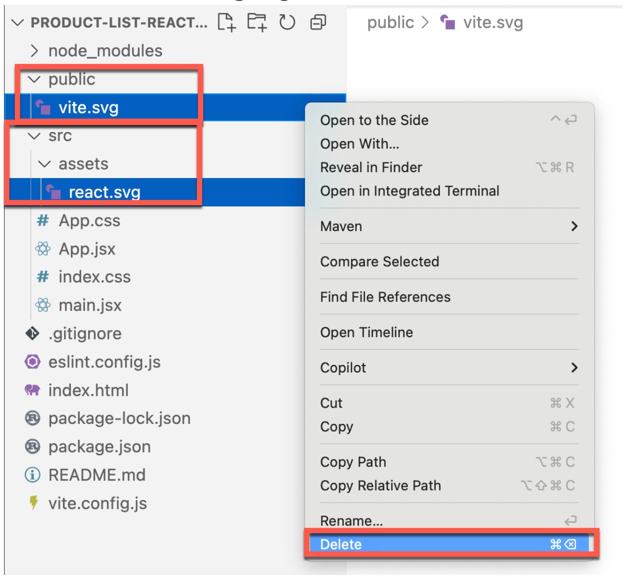
9. Update the style for root as well in **src/App.css**.



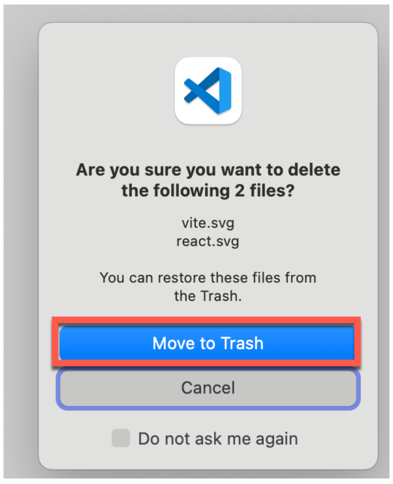
```
v src
  v assets
  App.css
  App.jsx
```

```css
#__SPNS__root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
```

10. Check that the application still looks and works the same way locally (remember, the server refreshes the page with every change). This React application has some SVG files. Since CF208, SVG files are disabled in DX for security reasons. See https://help.hcl-software.com/digital-experience/9.5/latest/whatsnew/cf20/newcf208/#web-content-manager-set-svg-to-disabled-by-default. You may either allow them, or simply remove or replace them. In this case, you may just delete them. Select the files **public/vite.svg** and **src/assets/react.svg**, right click **Delete** and confirm with **Move to Trash**.



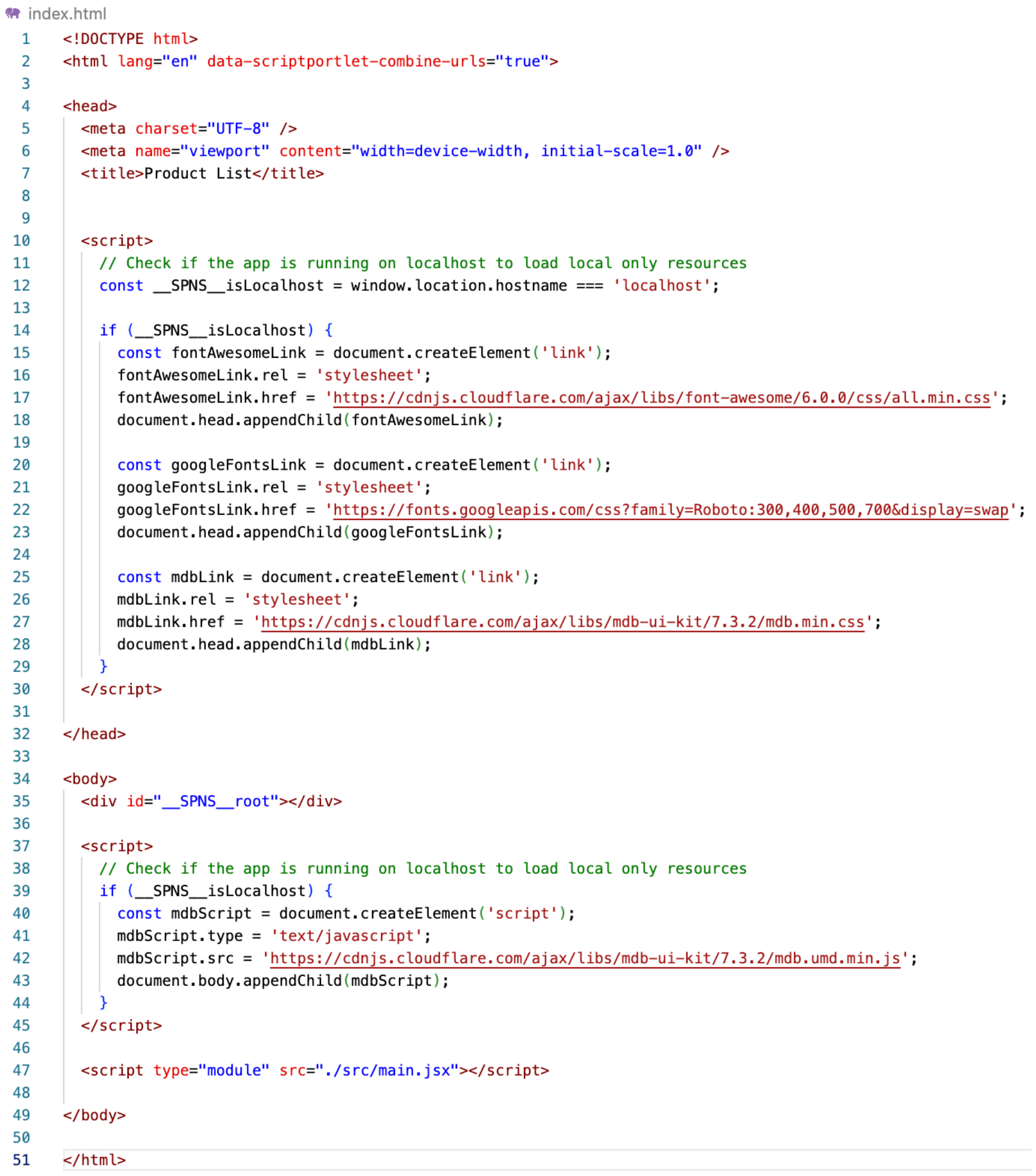
11. And confirm. Click **Move to Trash**.


Are you sure you want to delete the following 2 files?

vite.svg
react.svg

You can restore these files from the Trash.

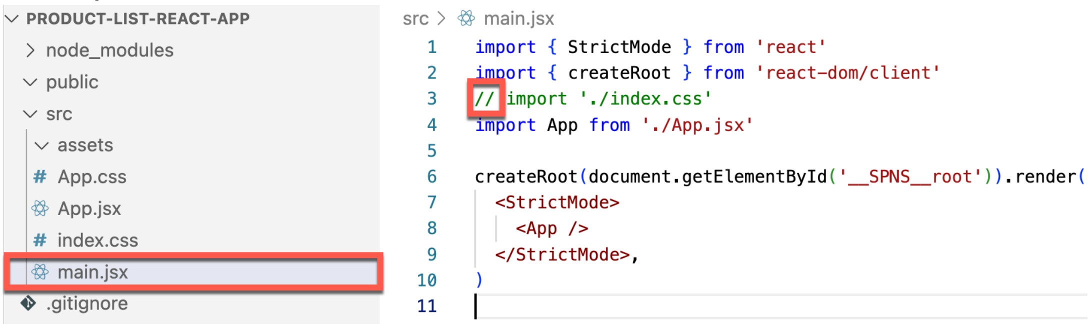
12. Update it your application with code to show a list of products, managed in an HCL Leap application. First, you create the styling. When your Script Application is deployed, you want this to be managed by your theme. However, when developing and testing locally, you may want to use the same or other styling and load it locally only. This is managed in the index.html file. Replace this with your downloaded file **product-list-react-app/index.html** version where special stylesheets for fontAwesome, Google Fonts and Material Design Bootstrap (MDB - https://mdbootstrap.com/) are loaded. Notice how the Script Application namespace (__SPNS__) is used to create unique identifiers, similar to how it is used for variables and methods.



index.html

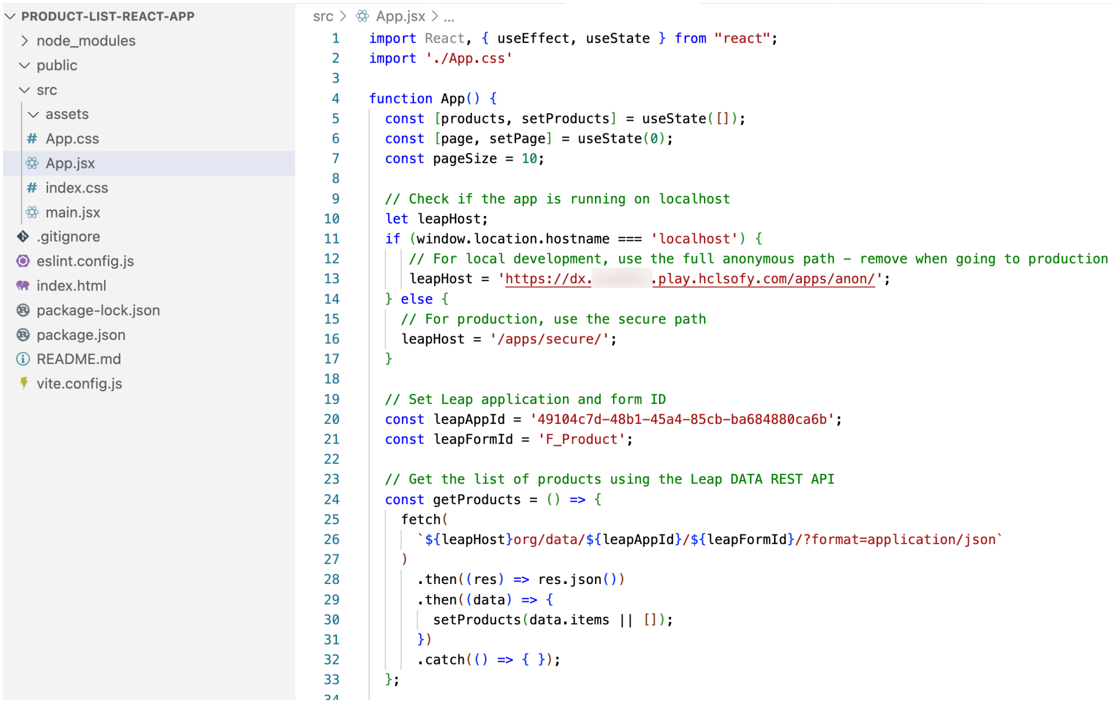
```html
<!DOCTYPE html>
<html lang="en" data-scriptportlet-combine-urls="true">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Product List</title>


  <script>
    // Check if the app is running on localhost to load local only resources
    const __SPNS__isLocalhost = window.location.hostname === 'localhost';

    if (__SPNS__isLocalhost) {
      const fontAwesomeLink = document.createElement('link');
      fontAwesomeLink.rel = 'stylesheet';
      fontAwesomeLink.href = 'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css';
      document.head.appendChild(fontAwesomeLink);

      const googleFontsLink = document.createElement('link');
      googleFontsLink.rel = 'stylesheet';
      googleFontsLink.href = 'https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap';
      document.head.appendChild(googleFontsLink);

      const mdbLink = document.createElement('link');
      mdbLink.rel = 'stylesheet';
      mdbLink.href = 'https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/7.3.2/mdb.min.css';
      document.head.appendChild(mdbLink);
    }
  </script>

</head>

<body>
  <div id="__SPNS__root"></div>

  <script>
    // Check if the app is running on localhost to load local only resources
    if (__SPNS__isLocalhost) {
      const mdbScript = document.createElement('script');
      mdbScript.type = 'text/javascript';
      mdbScript.src = 'https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/7.3.2/mdb.umd.min.js';
      document.body.appendChild(mdbScript);
    }
  </script>

  <script type="module" src="./src/main.jsx"></script>

</body>

</html>
```

13. As your styling is now managed externally, you should update or remove the locally managed CSS in the **src/index.css** file. Likely, in your environment, your design with styles would be managed in the DX theme and locally you may refer to them instead. In this case, you can simply remove it, by commenting out line 3 from the **src/main.jsx** that imports it with **import './index.css'**.



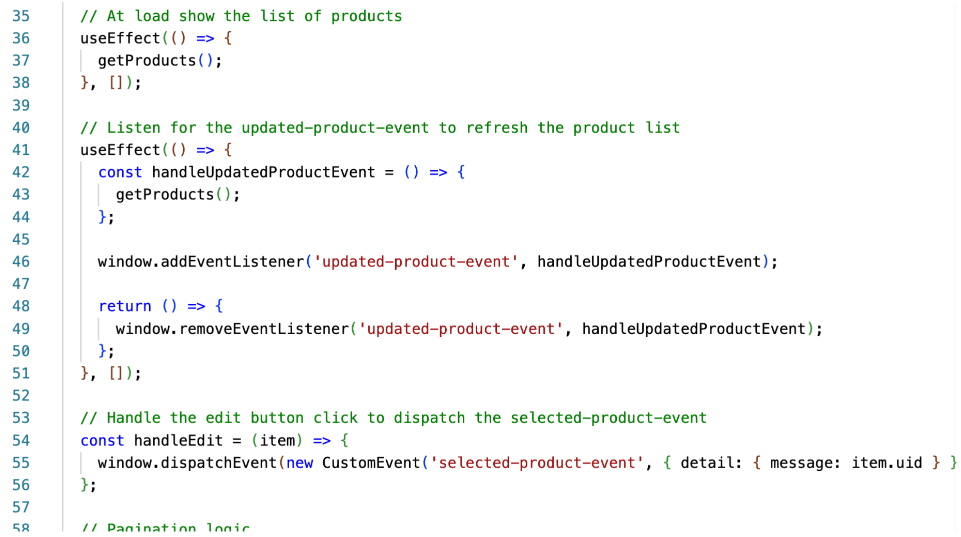
```
1   import { StrictMode } from 'react'
2   import { createRoot } from 'react-dom/client'
3   // import './index.css'
4   import App from './App.jsx'
5
6   createRoot(document.getElementById('__SPNS__root')).render(
7     <StrictMode>
8       <App />
9     </StrictMode>,
10  )
11
```

14. Now update it to get the products from an HCL Leap application. Replace the **src/app.jsx** with the one you downloaded under product-list-react-app and see how this reads product data from an HCL Leap application, using the Leap REST APIs that include the application ID and form ID. When testing it locally, it uses the anonymous Leap REST API, while using the authenticated REST API, once deployed on the DX server. This is done since CORS is blocking secured Leap data. You may overcome this by using Foundry as an API middleware or configure CORS for your HCL Leap development server.



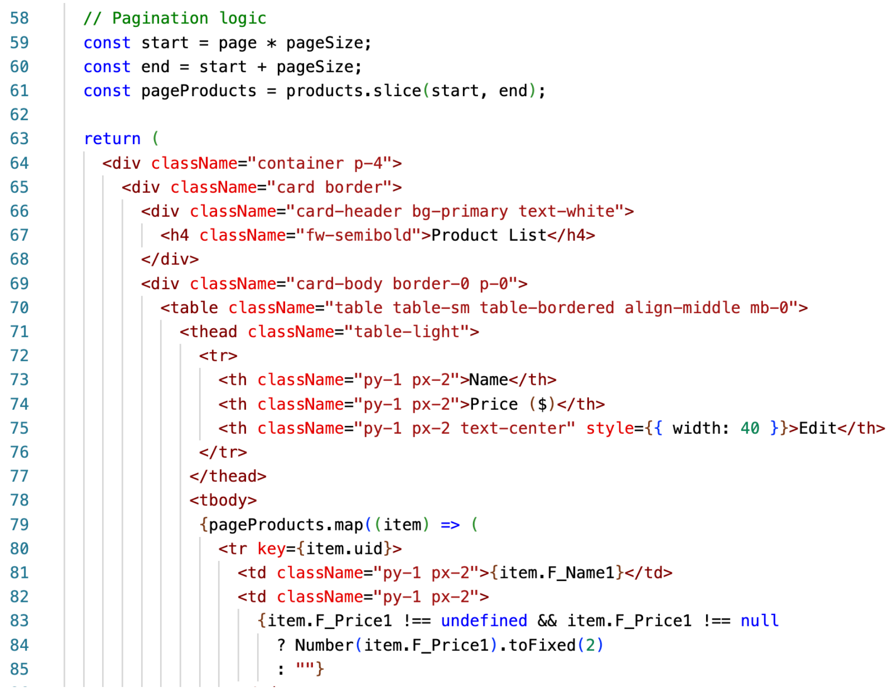
```
import React, { useEffect, useState } from "react";
import './App.css'

function App() {
  const [products, setProducts] = useState([]);
  const [page, setPage] = useState(0);
  const pageSize = 10;

  // Check if the app is running on localhost
  let leapHost;
  if (window.location.hostname === 'localhost') {
    // For local development, use the full anonymous path - remove when going to production
    leapHost = 'https://dx.[redacted].play.hclsofy.com/apps/anon/';
  } else {
    // For production, use the secure path
    leapHost = '/apps/secure/';
  }

  // Set Leap application and form ID
  const leapAppId = '49104c7d-48b1-45a4-85cb-ba684880ca6b';
  const leapFormId = 'F_Product';

  // Get the list of products using the Leap DATA REST API
  const getProducts = () => {
    fetch(
      `${leapHost}org/data/${leapAppId}/${leapFormId}/?format=application/json`
    )
      .then((res) => res.json())
      .then((data) => {
        setProducts(data.items || []);
      })
      .catch(() => { });
  };
```

15. A bit lower, you see the listener that is configured to listen for updates coming from the Product Details React application and creates a custom event with the selected product.



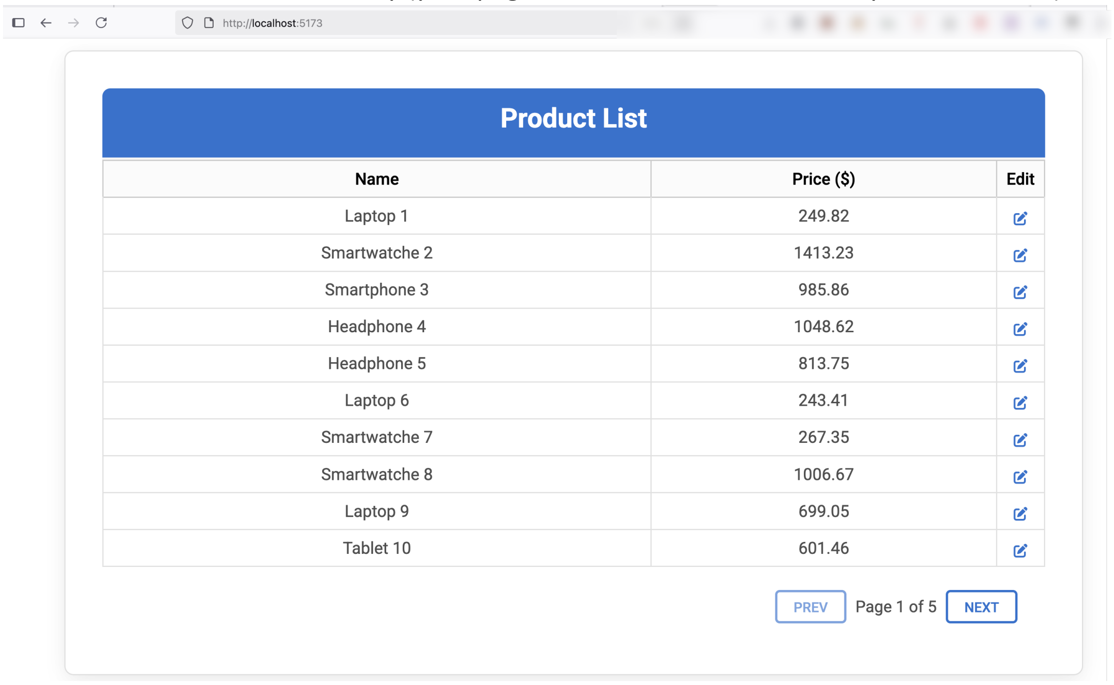
```
// At load show the list of products
useEffect(() => {
  getProducts();
}, []);

// Listen for the updated-product-event to refresh the product list
useEffect(() => {
  const handleUpdatedProductEvent = () => {
    getProducts();
  };

  window.addEventListener('updated-product-event', handleUpdatedProductEvent);

  return () => {
    window.removeEventListener('updated-product-event', handleUpdatedProductEvent);
  };
}, []);

// Handle the edit button click to dispatch the selected-product-event
const handleEdit = (item) => {
  window.dispatchEvent(new CustomEvent('selected-product-event', { detail: { message: item.uid } }
};

// Pagination logic
```

16. And further below, you see how it creates a Bootstrap styled card with the list of product entries.



```
// Pagination logic
const start = page * pageSize;
const end = start + pageSize;
const pageProducts = products.slice(start, end);

return (
  <div className="container p-4">
    <div className="card border">
      <div className="card-header bg-primary text-white">
        <h4 className="fw-semibold">Product List</h4>
      </div>
      <div className="card-body border-0 p-0">
        <table className="table table-sm table-bordered align-middle mb-0">
          <thead className="table-light">
            <tr>
              <th className="py-1 px-2">Name</th>
              <th className="py-1 px-2">Price ($)</th>
              <th className="py-1 px-2 text-center" style={{ width: 40 }}>Edit</th>
            </tr>
          </thead>
          <tbody>
           {pageProducts.map((item) => (
              <tr key={item.uid}>
                <td className="py-1 px-2">{item.F_Name1}</td>
                <td className="py-1 px-2">
                  {item.F_Price1 !== undefined && item.F_Price1 !== null
                    ? Number(item.F_Price1).toFixed(2)
                    : ""}
```

17. Now save this and check it locally (your page should refresh automatically and show this).

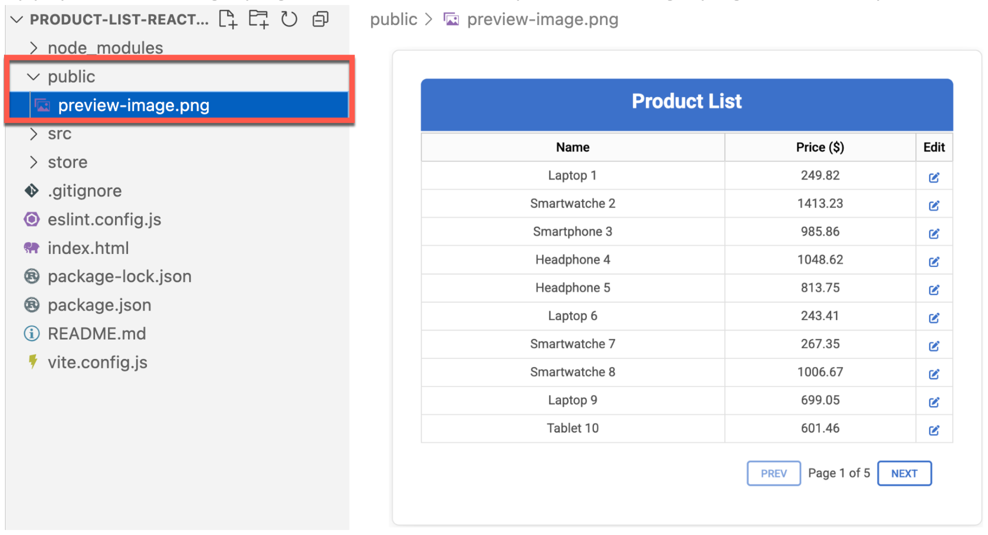


http://localhost:5173

Product List

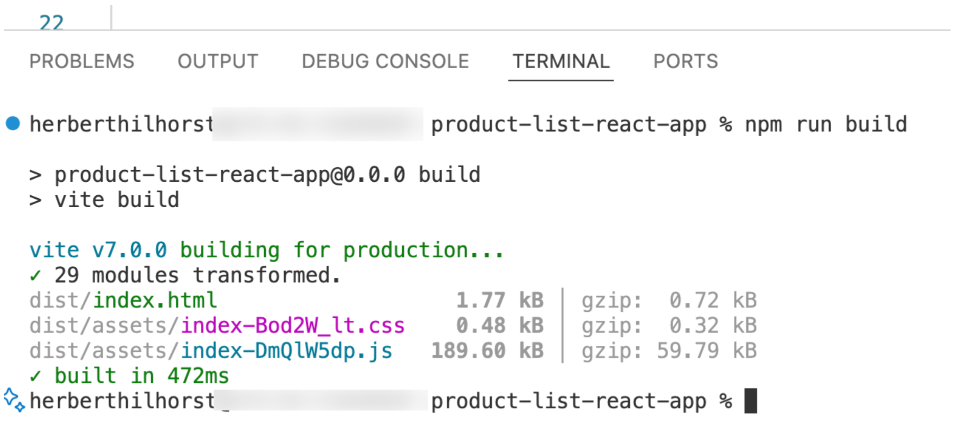
Name	Price ($)	Edit
Laptop 1	249.82	✎
Smartwatche 2	1413.23	✎
Smartphone 3	985.86	✎
Headphone 4	1048.62	✎
Headphone 5	813.75	✎
Laptop 6	243.41	✎
Smartwatche 7	267.35	✎
Smartwatche 8	1006.67	✎
Laptop 9	699.05	✎
Tablet 10	601.46	✎

18. This may be a good way to introduce this application to your business users in DX. You may take a screenshot or use a good image (or use the downloaded product-list-react-app/preview-image.png file) and save it as preview-image.png under the public folder.



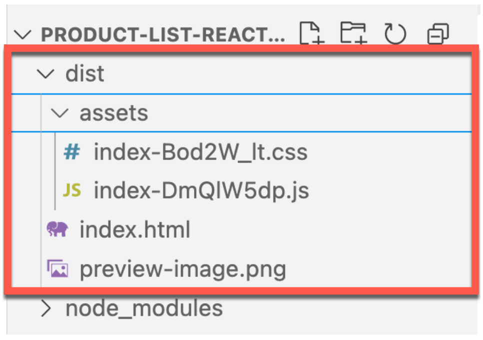
19. Now you need to build your application. Open a terminal and run the command

```
npm run build
```



```
● herberthilhorst                    product-list-react-app % npm run build

  > product-list-react-app@0.0.0 build
  > vite build

  vite v7.0.0 building for production...
  ✓ 29 modules transformed.
  dist/index.html                    1.77 kB │ gzip:  0.72 kB
  dist/assets/index-Bod2W_lt.css     0.48 kB │ gzip:  0.32 kB
  dist/assets/index-DmQlW5dp.js    189.60 kB │ gzip: 59.79 kB
  ✓ built in 472ms
  herberthilhorst                    product-list-react-app % 
```

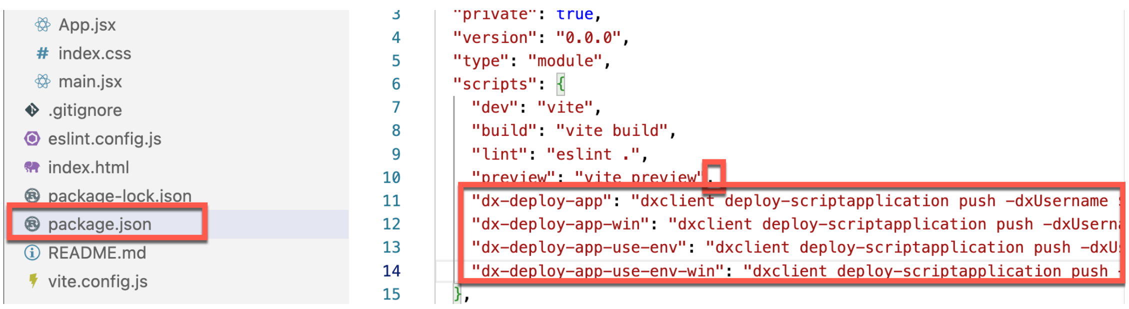
20. Notice where the build is created. In this case, in the dist folder which contains the index.html, the assets folder and the preview image.



```
PRODUCT-LIST-REACT...
  dist
    assets
      index-Bod2W_lt.css
      index-DmQlW5dp.js
    index.html
    preview-image.png
  node_modules
```

21. Now deploy this as a Script Application to your DX server. As this application is using Webpack, you may simply edit the package.json to add the DXClient commands and its corresponding configuration. Edit the **package.json** and under scripts, add these extra dxclient scripts:

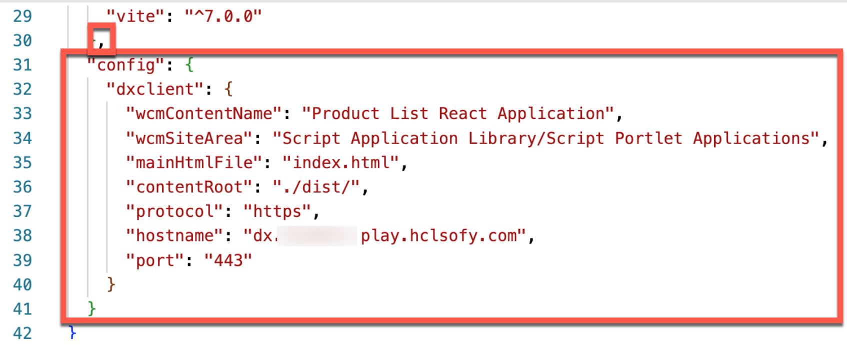
```
, "dx-deploy-app": "dxclient deploy-scriptapplication push -dxUsername $dxUsername -dxPassword $dxPassword -wcmContentName \"$npm_package_config_dxclient_wcmContentName\" -wcmSiteArea \"$npm_package_config_dxclient_wcmSiteArea\" -mainHtmlFile $npm_package_config_dxclient_mainHtmlFile -contentRoot \"$npm_package_config_dxclient_contentRoot\" -dxProtocol $npm_package_config_dxclient_protocol -hostname $npm_package_config_dxclient_hostname -dxPort $npm_package_config_dxclient_port",

"dx-deploy-app-win": "dxclient deploy-scriptapplication push -dxUsername %dxUsername% -dxPassword %dxPassword% -wcmContentName \"%npm_package_config_dxclient_wcmContentName%\" -wcmSiteArea \"%npm_package_config_dxclient_wcmSiteArea%\" -mainHtmlFile %npm_package_config_dxclient_mainHtmlFile% -contentRoot \"%npm_package_config_dxclient_contentRoot%\" -dxProtocol %npm_package_config_dxclient_protocol% -hostname %npm_package_config_dxclient_hostname% -dxPort %npm_package_config_dxclient_port%",

"dx-deploy-app-use-env": "dxclient deploy-scriptapplication push -dxUsername $dxUsername -dxPassword $dxPassword -wcmContentName \"$npm_package_config_dxclient_wcmContentName\" -wcmSiteArea \"$npm_package_config_dxclient_wcmSiteArea\" -mainHtmlFile $npm_package_config_dxclient_mainHtmlFile -contentRoot \"$npm_package_config_dxclient_contentRoot\" -dxProtocol $dxProtocol -hostname $dxHostname -dxPort $dxPort",

"dx-deploy-app-use-env-win": "dxclient deploy-scriptapplication push -dxUsername %dxUsername% -dxPassword %dxPassword% -wcmContentName \"%npm_package_config_dxclient_wcmContentName%\" -wcmSiteArea \"%npm_package_config_dxclient_wcmSiteArea%\" -mainHtmlFile %npm_package_config_dxclient_mainHtmlFile% -contentRoot \"%npm_package_config_dxclient_contentRoot%\" -dxProtocol %dxProtocol% -hostname %dxHostname% -dxPort %dxPort%"
```



```
"private": true,
"version": "0.0.0",
"type": "module",
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "lint": "eslint .",
  "preview": "vite preview",
  "dx-deploy-app": "dxclient deploy-scriptapplication push -dxUsername
  "dx-deploy-app-win": "dxclient deploy-scriptapplication push -dxUsern
  "dx-deploy-app-use-env": "dxclient deploy-scriptapplication push -dxU
  "dx-deploy-app-use-env-win": "dxclient deploy-scriptapplication push
},
```

22. In the same file, add a config section using the application's **name** (managed with **wcmContentName**, here set to **Product List React Application**). Then specify the deployment location (site area with **wcmSiteArea**). In this example, the application will be deployed under the **Script Portlet Applications** site area within the **Script Application Library**. This site area is already configured to make any Script Application content show in the toolbar. You need to specify the main HTML file with **mainHtmlFile** (in this application **index.html**), DX server **protocol**, **hostname** (**update this to your server**) and **port**. Also set the **contentRoot** to the folder where your application is built, in this case **./dist/**. Your configuration should look similar to this:

```
,
"config": {
  "dxclient": {
    "wcmContentName": "Product List React Application",
    "wcmSiteArea": "Script Application Library/Script Portlet Applications",
    "mainHtmlFile": "index.html",
    "contentRoot": "./dist/",
    "protocol": "https",
    "hostname": "dx.sbx0000.play.hclsofy.com",
    "port": "443"
  }
}
```



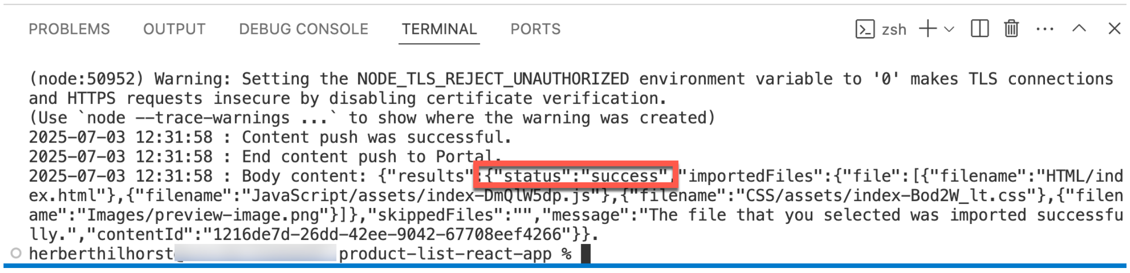
```
    "vite": "^7.0.0"
  },
  "config": {
    "dxclient": {
      "wcmContentName": "Product List React Application",
      "wcmSiteArea": "Script Application Library/Script Portlet Applications",
      "mainHtmlFile": "index.html",
      "contentRoot": "./dist/",
      "protocol": "https",
      "hostname": "dx.[redacted]play.hclsofy.com",
      "port": "443"
    }
  }
}
```

23. Deploy the application to your DX server using your new dx-deploy-app (or dx-deploy-app-win for Windows) script. As the user name and password are not set prepend, use the command with *dxUsername=<username> dxPassword=<password> run dx-deploy-app* to set them and run the dx-deploy-app (-win for Windows) script. Use your Configuration Wizard credentials for this. In SoFy, you may find them under the Sandbox Links; here wpsadmin for both.

Configuration Wizard	Open
Login ID	wpsadmin
Password	wpsadmin

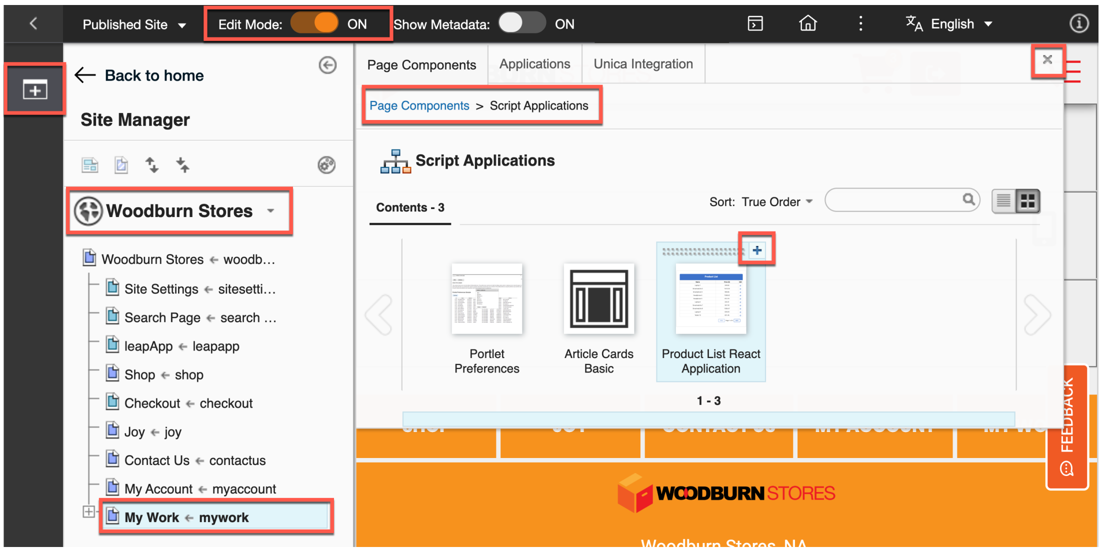
24. In this case, run the following command and check that the content push was successful:

```
dxUsername=wpsadmin dxPassword=wpsadmin npm run dx-deploy-app
```

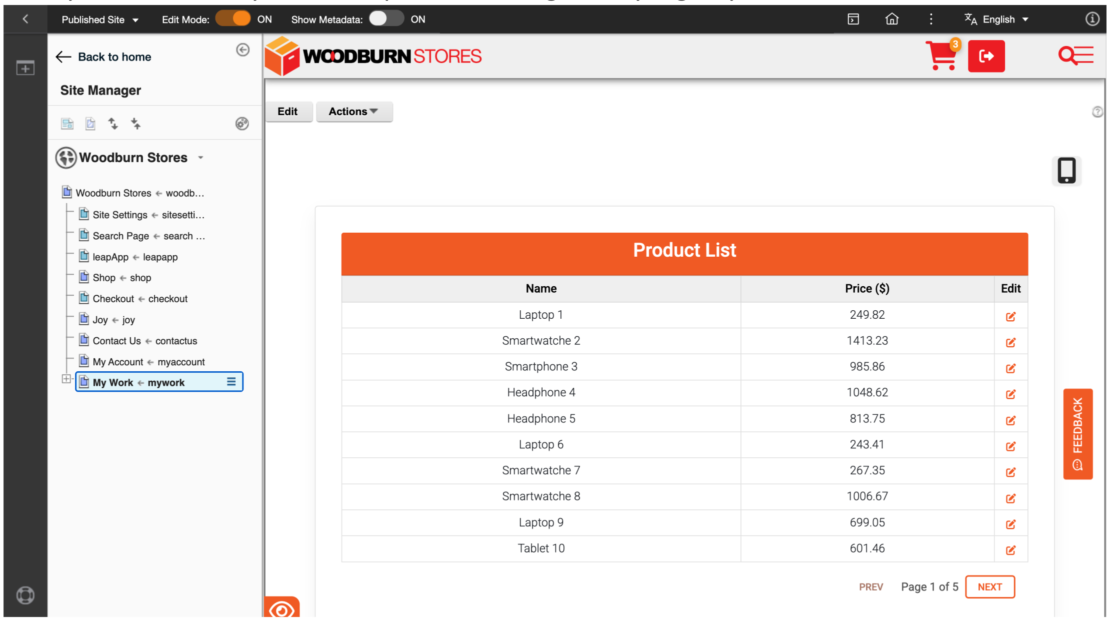


```
(node:50952) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections
and HTTPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
2025-07-03 12:31:58 : Content push was successful.
2025-07-03 12:31:58 : End content push to Portal.
2025-07-03 12:31:58 : Body content: {"results":{"status":"success","importedFiles":{"file":[{"filename":"HTML/ind
ex.html"},{"filename":"JavaScript/assets/index-DmQlW5dp.js"},{"filename":"CSS/assets/index-Bod2W_lt.css"},{"filen
ame":"Images/preview-image.png"}]},"skippedFiles":"","message":"The file that you selected was imported successfu
lly.","contentId":"1216de7d-26dd-42ee-9042-67708eef4266"}}.
herberthilhorst[redacted]product-list-react-app %
```

25. Add your Script Application to a DX page. Use a page that uses the styling referred to in this Script Application and is set up with a profile using a React module, e.g. under Woodburn Stores. Ensure you have the Edit Mode enabled. Go to the **Woodburn Stores** site, select the **My Work** page, open the **Add page components and applications**, select **Script Applications** and use the **+** to add your new **Product List React Application** to this page. Then close the toolbar.



26. Now you should see your list of products using the styling of your DX theme.

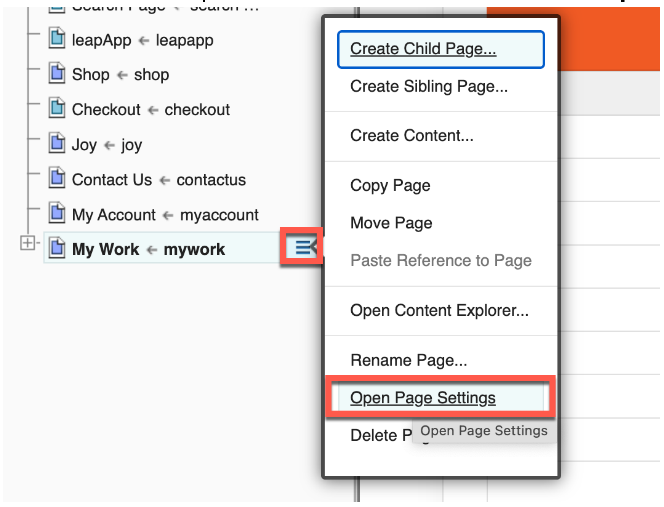


Product List

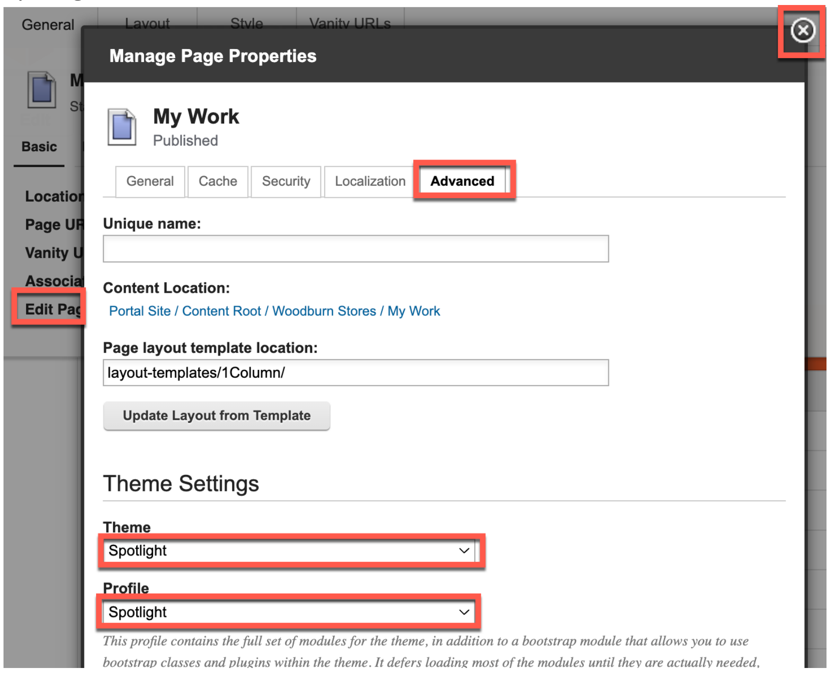
Name	Price ($)	Edit
Laptop 1	249.82	✎
Smartwatche 2	1413.23	✎
Smartphone 3	985.86	✎
Headphone 4	1048.62	✎
Headphone 5	813.75	✎
Laptop 6	243.41	✎
Smartwatche 7	267.35	✎
Smartwatche 8	1006.67	✎
Laptop 9	699.05	✎
Tablet 10	601.46	✎

PREV Page 1 of 5 NEXT

27. If you do not see the application, check that this page uses a theme profile configured with a React module. Open the context menu and select **Open Page Settings**.



28. Click **Edit Page Settings**, **Advanced** and notice your page uses a Spotlight profile in the Spotlight theme. Close it.



Manage Page Properties

My Work
Published

General | Cache | Security | Localization | Advanced

Unique name:

Content Location:
Portal Site / Content Root / Woodburn Stores / My Work

Page layout template location:
layout-templates/1Column/

Update Layout from Template

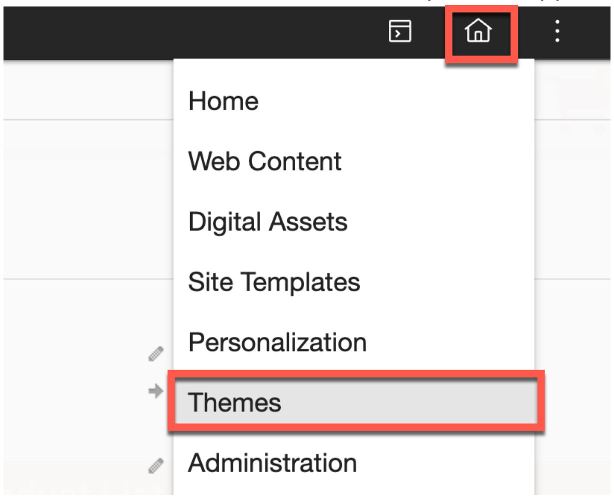
Theme Settings

Theme
Spotlight

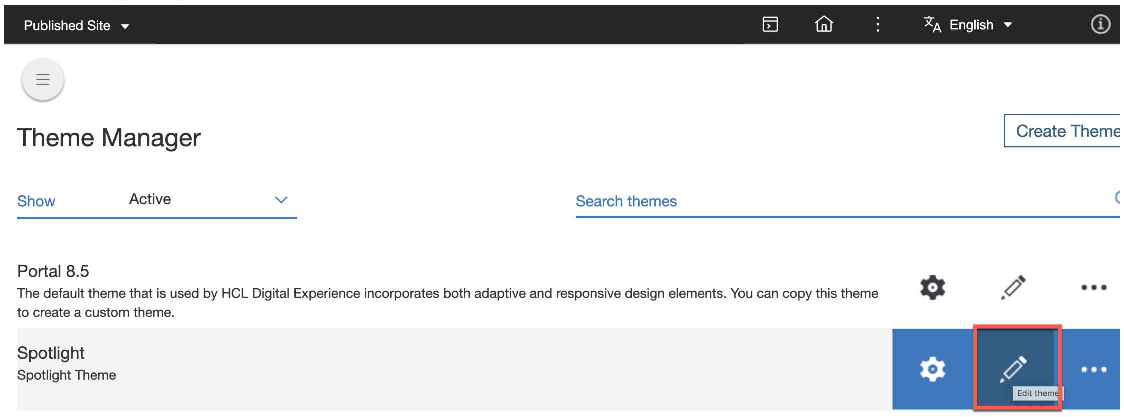
Profile
Spotlight

This profile contains the full set of modules for the theme, in addition to a bootstrap module that allows you to use bootstrap classes and plugins within the theme. It defers loading most of the modules until they are actually needed.

29. Check that React is in this theme. Open the applications menu and click **Themes**.



30. Edit the **Spotlight** theme.



Published Site | English

Theme Manager

Create Theme

Show: Active

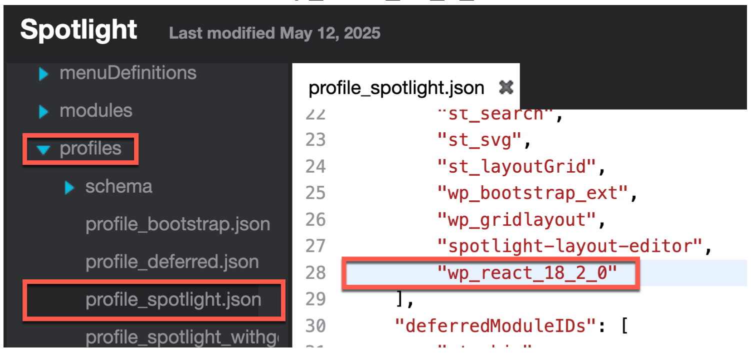
Search themes

Portal 8.5
The default theme that is used by HCL Digital Experience incorporates both adaptive and responsive design elements. You can copy this theme to create a custom theme.

Spotlight
Spotlight Theme

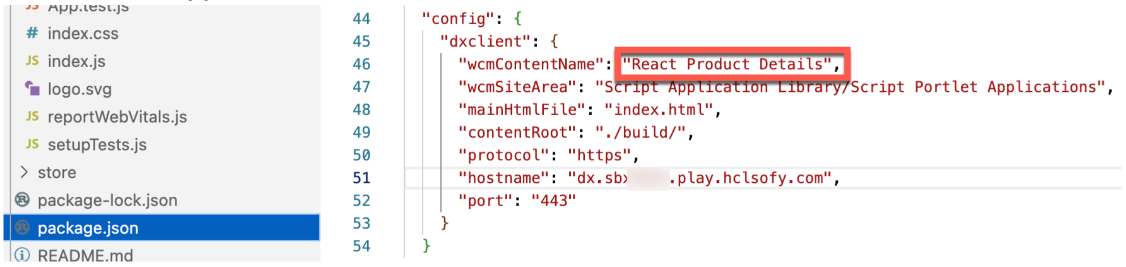
Edit theme

31. Open the **profiles**, select the spotlight profile **profile_spotlight.json** and notice the out of the box React module **wp_react_18_2_0** is selected.



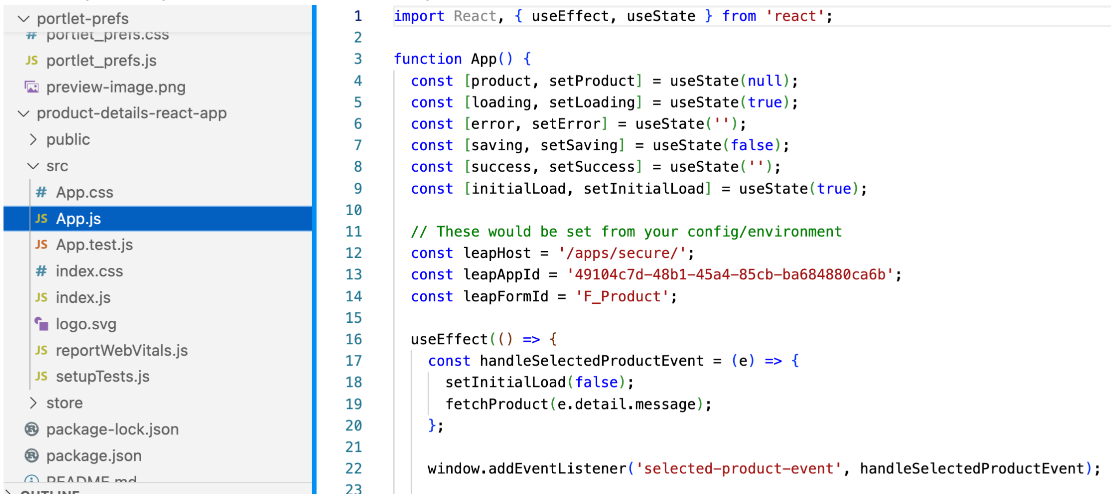
```
        "st_search",
        "st_svg",
        "st_layoutGrid",
        "wp_bootstrap_ext",
        "wp_gridlayout",
        "spotlight-layout-editor",
        "wp_react_18_2_0"
    ],
    "deferredModuleIDs": [
```

32. Now create a second Product Details React Application, called product-details-react-app. You may either use the one you downloaded or follow the same steps to create it from scratch as for your Product List React Application. In that case, use **npm create vite@latest product-details-react-app -- --template react**. Then replace **index.html** by the **product-details-react-app/index.html** and **src/apps.jsx** with the ones in the sample code under **product-details-react-app** you have downloaded. Update the **vite.config.js**, and **main.jsx** again. Delete the svg files. Add the DXClient deployment scripts and configuration to the **package.json** (you may copy it from the Product List and update as well or from the downloaded **product-details-react-app/package.json**), this time with the name **Product Details React Application**.



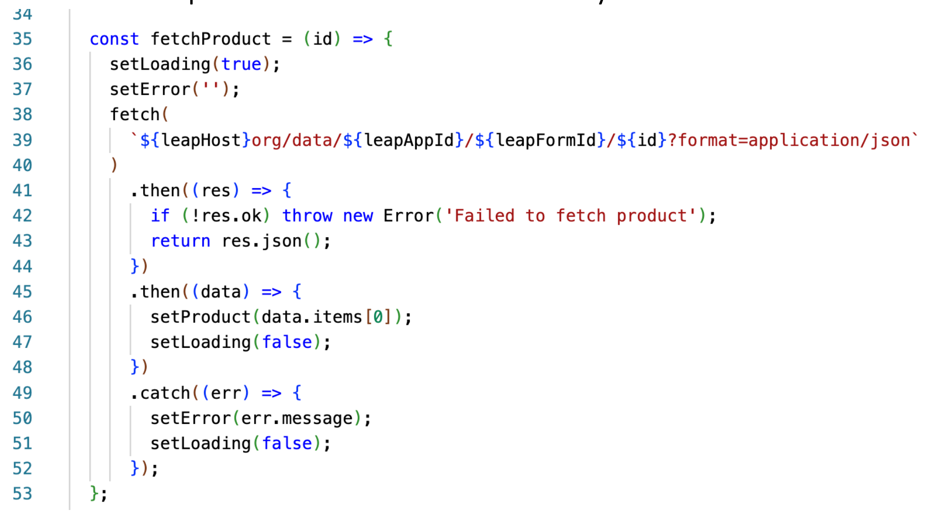
```
  "config": {
    "dxclient": {
      "wcmContentName": "React Product Details",
      "wcmSiteArea": "Script Application Library/Script Portlet Applications",
      "mainHtmlFile": "index.html",
      "contentRoot": "./build/",
      "protocol": "https",
      "hostname": "dx.sb[redacted].play.hclsofy.com",
      "port": "443"
    }
  }
```

33. The Leap application allows to read but not write as an anonymous user. Hence, the code uses the secure Leap REST API and will only run on the DX server, where it benefits from the Single Sign On capability that is set up on that environment. You also see below the listener that is put in place for the selected product event.



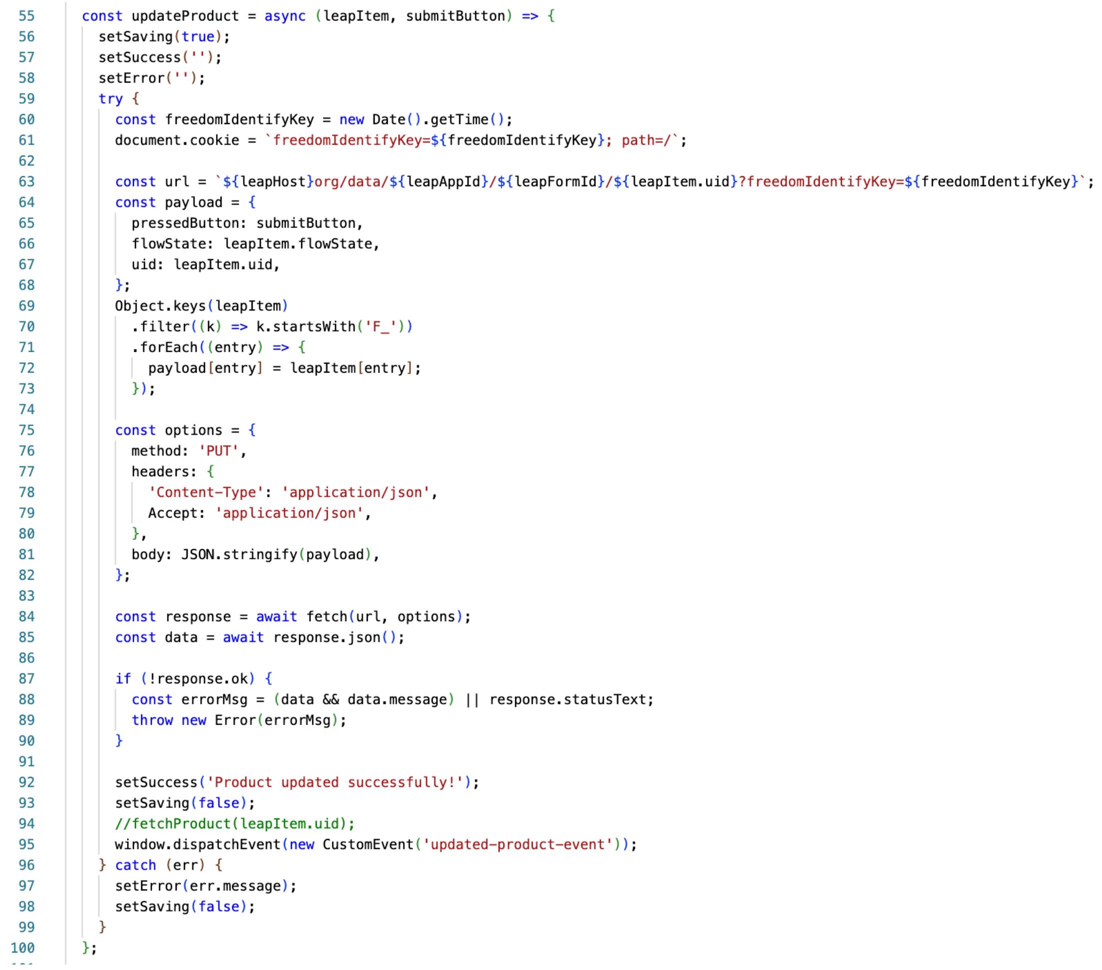
```
import React, { useEffect, useState } from 'react';

function App() {
  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [saving, setSaving] = useState(false);
  const [success, setSuccess] = useState('');
  const [initialLoad, setInitialLoad] = useState(true);

  // These would be set from your config/environment
  const leapHost = '/apps/secure/';
  const leapAppId = '49104c7d-48b1-45a4-85cb-ba684880ca6b';
  const leapFormId = 'F_Product';

  useEffect(() => {
    const handleSelectedProductEvent = (e) => {
      setInitialLoad(false);
      fetchProduct(e.detail.message);
    };

    window.addEventListener('selected-product-event', handleSelectedProductEvent);
```

34. Further below, you see below how the application reads a single Leap data form, in this case for the selected product and checks if there are any errors.



```
const fetchProduct = (id) => {
  setLoading(true);
  setError('');
  fetch(
    `${leapHost}org/data/${leapAppId}/${leapFormId}/${id}?format=application/json`
  )
    .then((res) => {
      if (!res.ok) throw new Error('Failed to fetch product');
      return res.json();
    })
    .then((data) => {
      setProduct(data.items[0]);
      setLoading(false);
    })
    .catch((err) => {
      setError(err.message);
      setLoading(false);
    });
};
```

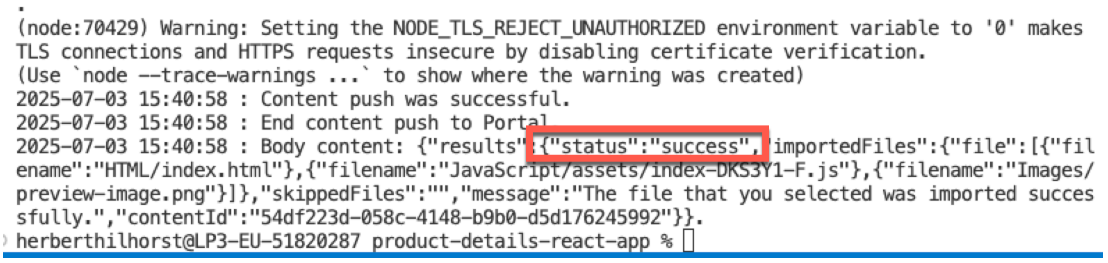
35. You can also see the code that updates the product data in the HCL Leap application and sends a custom event for the Product List React application to get it refreshed.



```
const updateProduct = async (leapItem, submitButton) => {
  setSaving(true);
  setSuccess('');
  setError('');
  try {
    const freedomIdentifyKey = new Date().getTime();
    document.cookie = `freedomIdentifyKey=${freedomIdentifyKey}; path=/`;

    const url = `${leapHost}org/data/${leapAppId}/${leapFormId}/${leapItem.uid}?freedomIdentifyKey=${freedomIdentifyKey}`;
    const payload = {
      pressedButton: submitButton,
      flowState: leapItem.flowState,
      uid: leapItem.uid,
    };
    Object.keys(leapItem)
      .filter((k) => k.startsWith('F_'))
      .forEach((entry) => {
        payload[entry] = leapItem[entry];
      });

    const options = {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        Accept: 'application/json',
      },
      body: JSON.stringify(payload),
    };

    const response = await fetch(url, options);
    const data = await response.json();

    if (!response.ok) {
      const errorMsg = (data && data.message) || response.statusText;
      throw new Error(errorMsg);
    }

    setSuccess('Product updated successfully!');
    setSaving(false);
    //fetchProduct(leapItem.uid);
    window.dispatchEvent(new CustomEvent('updated-product-event'));
  } catch (err) {
    setError(err.message);
    setSaving(false);
  }
};
```

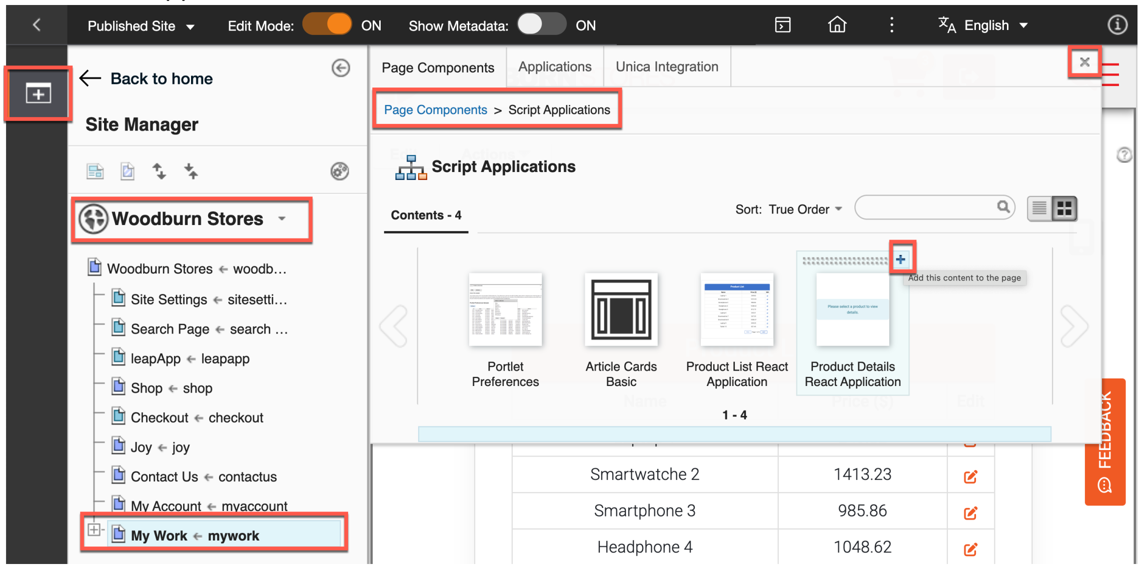
36. Create (or reuse the downloaded one) and add again a preview image under the public folder.

37. Deploy this application to your DX server. Open a Terminal, build it (npm run build), deploy it (using dxUsername=wpsadmin dxPassword=wpsadmin npm run dx-deploy-app) and check if was successfully deployed again.

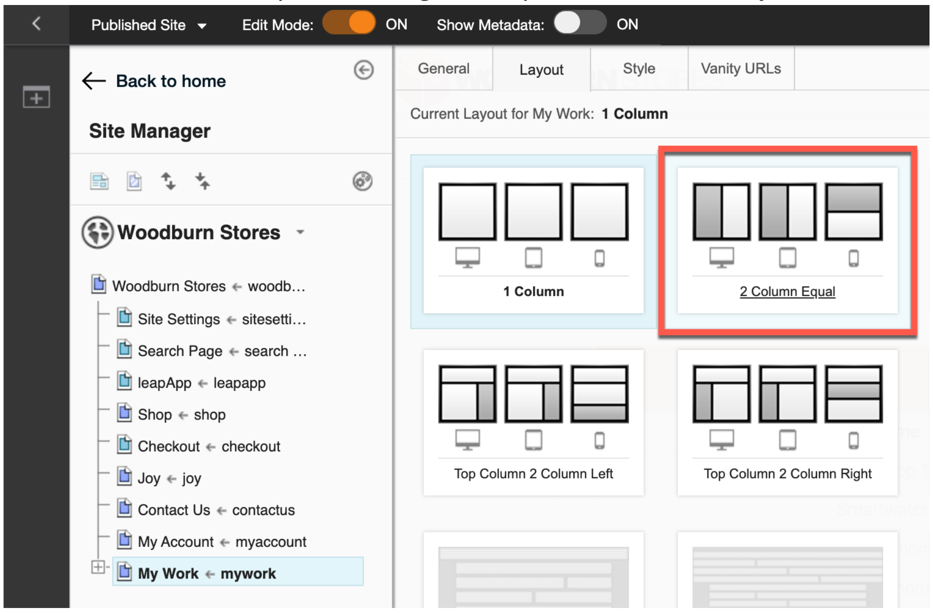


```
(node:70429) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes
TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
2025-07-03 15:40:58 : Content push was successful.
2025-07-03 15:40:58 : End content push to Portal
2025-07-03 15:40:58 : Body content: {"results":{"status":"success","importedFiles":{"file":[{"fil
ename":"HTML/index.html"},{"filename":"JavaScript/assets/index-DKS3Y1-F.js"},{"filename":"Images/
preview-image.png"}]},"skippedFiles":"","message":"The file that you selected was imported succes
sfully.","contentId":"54df223d-058c-4148-b9b0-d5d176245992"}}.
herberthilhorst@LP3-EU-51820287 product-details-react-app %
```

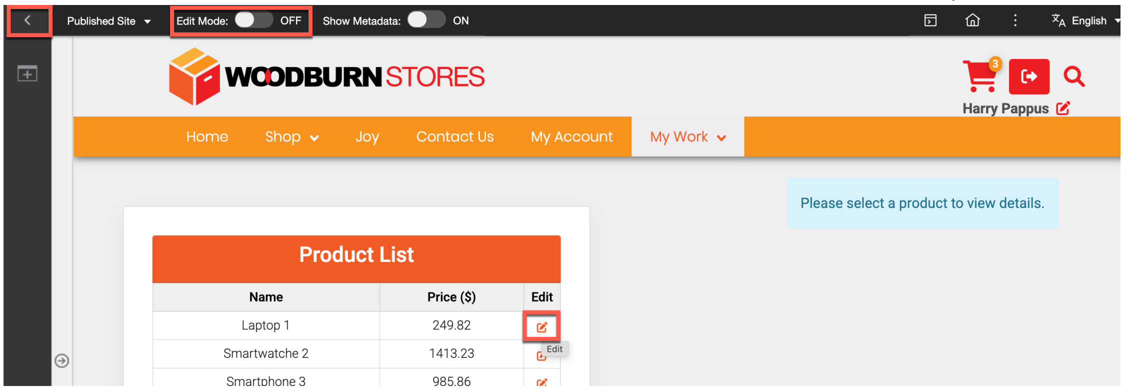
38. Add this new Product Details Script Application to the same page as where you put the Product List application.



39. Now show them side by side. Change the layout to **2 Column Equal**.

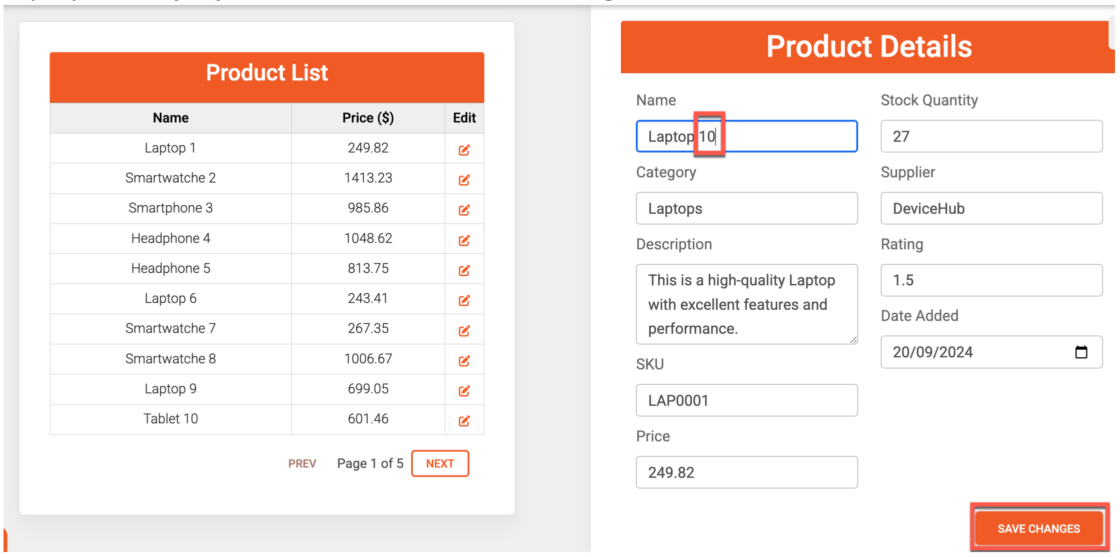


40. And move the Product Details React Application to the right. Then test if they work together. Disable the **Edit Mode**, close the toolbar and click the **Edit** button of one of the products.



Name	Price ($)	Edit
Laptop 1	249.82	
Smartwatche 2	1413.23	
Smartphone 3	985.86	

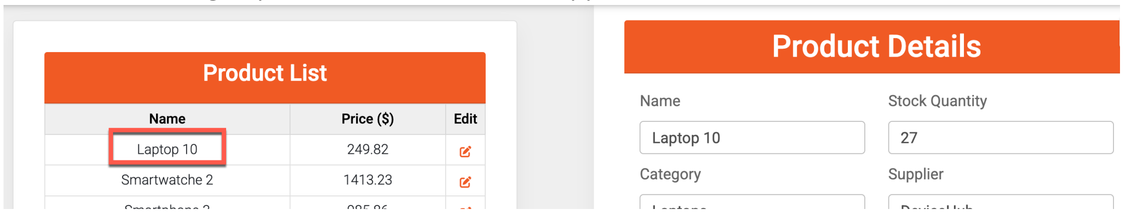
41. The Product Details application should now appear. Update it, e.g. change the **name** of Laptop 1 to **Laptop 10** and save it. Click **Save Changes**.



Name	Price ($)	Edit
Laptop 1	249.82	
Smartwatche 2	1413.23	
Smartphone 3	985.86	
Headphone 4	1048.62	
Headphone 5	813.75	
Laptop 6	243.41	
Smartwatche 7	267.35	
Smartwatche 8	1006.67	
Laptop 9	699.05	
Tablet 10	601.46	

Name: Laptop 10	Stock Quantity: 27
Category: Laptops	Supplier: DeviceHub
Description: This is a high-quality Laptop with excellent features and performance.	Rating: 1.5
SKU: LAP0001	Date Added: 20/09/2024
Price: 249.82	

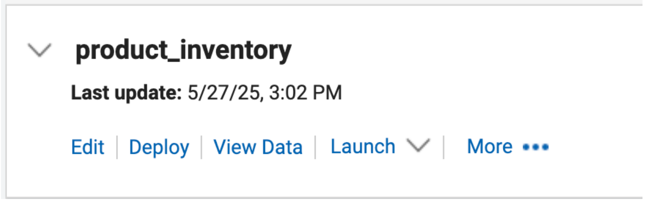
42. You see the change update in the Product List application.



Name	Price ($)	Edit
Laptop 10	249.82	
Smartwatche 2	1413.23	

Name: Laptop 10	Stock Quantity: 27

43. If you are interested, you may take a look at the HCL Leap application that was easily created from a Microsoft Excel sheet. It is called **product_inventory** and deployed on your instance.


product_inventory

Last update: 5/27/25, 3:02 PM

Congratulations! You have successfully created and deployed two new React Script Applications that use HCL Leap data and communicate with together on a DX page.

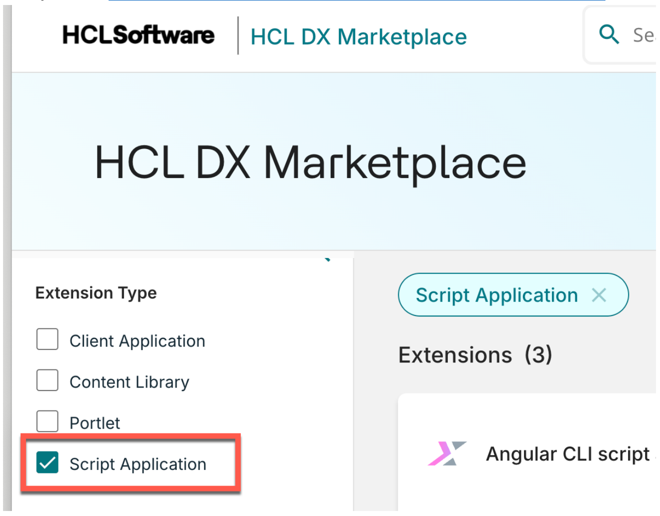
Conclusion

By following this lab, you have learned how to create reusable and configurable Script Applications that can use Vanilla JavaScript or any JavaScript framework , such as Angular and React.. You learned how to test JavaScript applications locally, use a DXClient configuration to simplify deploying Script Applications to a DX server and get them team up in DX sites.

You also learned how to turn any JavaScript application into a DX Script Application.

Feel free to use any existing applications or AI tools to help you build new and exciting JavaScript applications, hosted on HCL DX. Once deployed to DX, as it is a content item, you may use all the DX specific capabilities, such as managing its access, target it, syndicate it between DX servers, etc.

You may look at other more advanced how-to guides in the Help Center: https://help.hcl-software.com/digital-experience/9.5/latest/guide_me/tutorials/scriptapps/how_to/ and other samples in the DX Marketplace https://marketplace.hcl-software.com/dx where you may search for all Script Applications:



, such as https://marketplace.hcl-software.com/dx/catalog/dx-sample-react-script-app and those in the Digital Solutions Community https://developer.ds.hcl-software.com/tag/script-application.

Resources

Refer to the following resources to learn more:

HCL Digital Experience Home - https://hclsw.co/dx

HCL SoFy - https://hclsofy.com/

HCL Digital Experience on HCL SoFy - https://hclsofy.com/dx

HCL Software - https://hclsw.co/software

HCL Product Support - https://hclsw.co/product-support

HCL DX Product Documentation - https://hclsw.co/dx-product-documentation

HCL DX Support Q&A Forum - https://hclsw.co/dx-support-forum

HCL DX Video Playlist on YouTube - https://hclsw.co/dx-video-playlist

HCL DX Product Ideas - https://hclsw.co/dx-ideas

HCL DX Product Demos - https://hclsw.co/dx-product-demo

HCL DX Did You Know? Videos - https://hclsw.co/dx-dyk-videos

HCL DX GitHub - https://hclsw.co/dx-github

Legal statements

Disclaimers