Contents

[Workshop – Implementing Experience-Driven Commerce 1](#_Toc34366730)

[Exercise 1 – Setting up the Quicksilver website 1](#_Toc34366731)

[Configuring the Episerver NuGets package source 1](#_Toc34366732)

[https://nuget.episerver.com/feed/packages.svc/ 1](#_Toc34366733)

[Downloading the Quicksilver repository 1](#_Toc34366734)

[Configuring Quicksilver to use your Episerver Personalization account 2](#_Toc34366735)

[Starting Quicksilver for the first time and performing migration 3](#_Toc34366736)

[Updating the Episerver NuGet packages 4](#_Toc34366737)

[Updating the Episerver database 5](#_Toc34366738)

[(Optional) Exploring the Quicksilver site as a visitor 5](#_Toc34366739)

[(Optional) Exploring the Quicksilver site as a Commerce admin 8](#_Toc34366740)

[Exercise 2 – Product Feed and Diagnostics 10](#_Toc34366741)

[Exporting the Quicksilver product feed 10](#_Toc34366742)

[Adding diagnostics information to the Start page 11](#_Toc34366743)

[Exercise 3 – Getting product recommendations in web pages 14](#_Toc34366744)

[Implementing Server-side API Code with Tracking Attribute (Home Page Recommendations) 14](#_Toc34366745)

[Implementing Server-side API Code without Tracking Attribute (Product Recommendatioins) 19](#_Toc34366746)

[Implementing Client-side API Code (JavaScript) 22](#_Toc34366747)

[Exercise 4 – Trouble Shooting and Session Viewer 26](#_Toc34366748)

[Implementing a debugger tracking data interceptor 28](#_Toc34366749)

[Implementing a custom user data service 30](#_Toc34366750)

[Disable the debugger tracking data interceptor 31](#_Toc34366751)

[Exercise 5 – Customizing the product feed 32](#_Toc34366752)

[Configuring the catalog feed 32](#_Toc34366753)

[Implementing a catalog item filter 33](#_Toc34366754)

[Testing the catalog item filter 34](#_Toc34366755)

# Workshop – Implementing Experience-Driven Commerce

**Goal**

The overall goal of the exercises in this module is to learn how to implement an experience-driven commerce website. You will:

* Set up the Quicksilver reference website project.
* Explore Server-side code to get product recommendations on a web page.
* Explore Client-side code to get product recommendations.
* Customize the product feed.

## Exercise 1 – Setting up the Quicksilver website

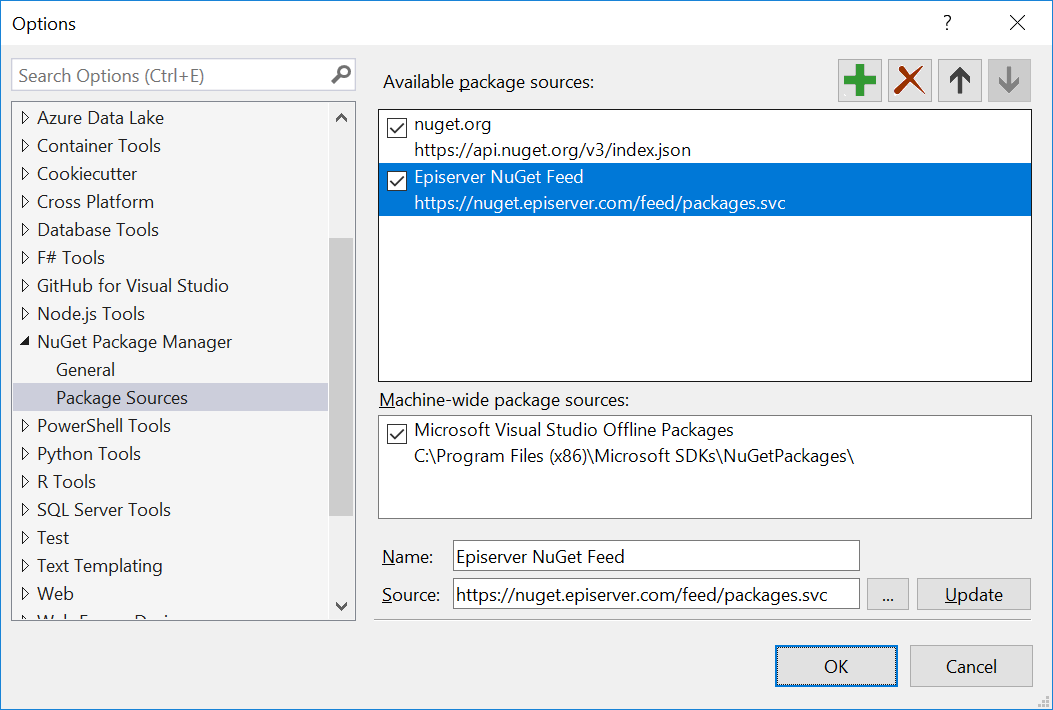
In this exercise, you will download the Quicksilver reference project repository from GitHub and then review how it has been implemented.

**Prerequisites:** none.

* This exercise you’ll use a demo account setup for Episerver Personalization (Product Recommendations).

### Configuring the Episerver NuGets package source

* If you have already configured the **Episerver NuGets** package source you can skip this section.

1. Start Microsoft Visual Studio.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Settings**.
3. In the **Options** dialog, in the list on the left, click **Package Sources**.
4. If the EpiserverNuGet feed does not exist as an available package source, as shown in the following screenshot, then click the **green plus** button to add it. The name can be anything, although we recommend using **Episerver NuGets**, and the path must be:

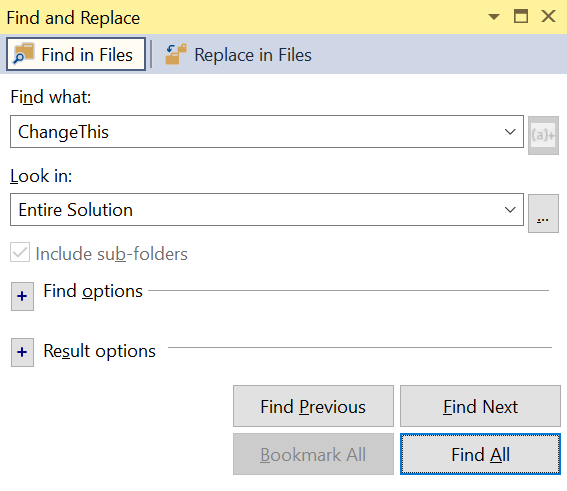
### <https://nuget.episerver.com/feed/packages.svc/>

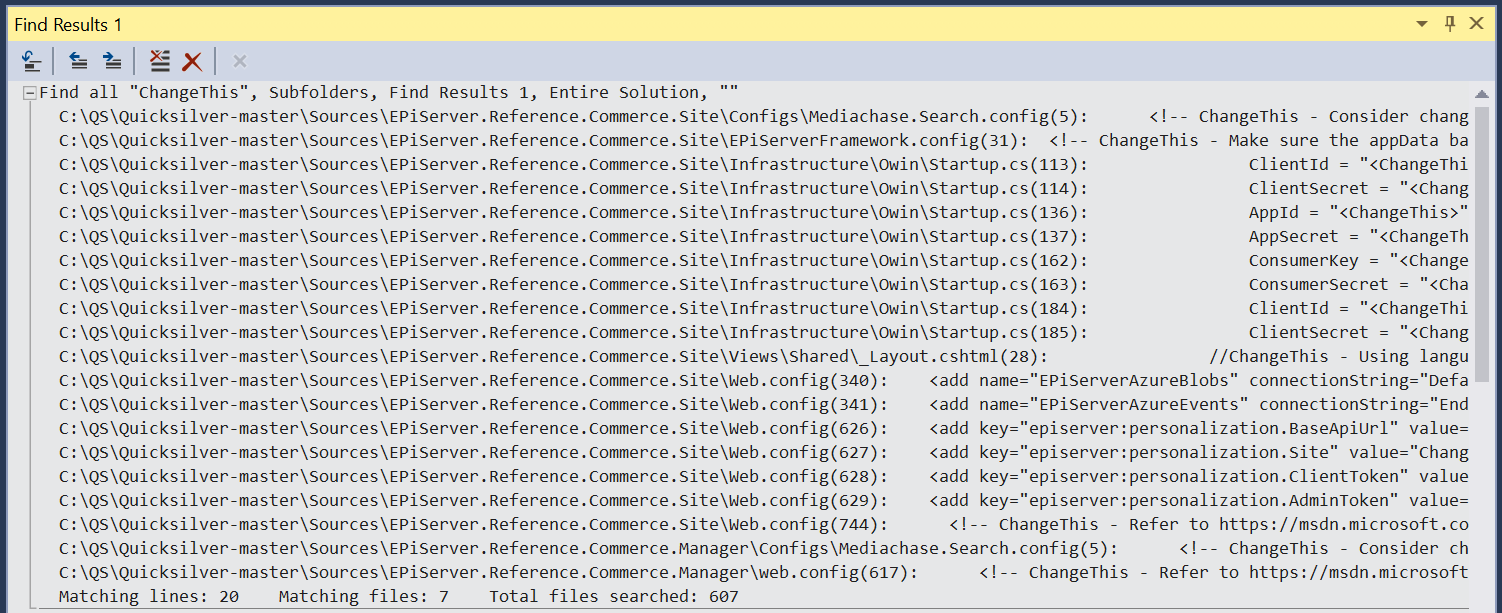
### Downloading the Quicksilver repository

1. Start your favorite web browser.
2. Navigate to <https://github.com/episerver/Quicksilver>
3. Follow the installation instructions, as summarized in the following steps:
   1. Download and unblock the ZIP.

* The Quicksilver-master.zip file is only about 55.5 MB because it does not include NuGet packages.
  1. Extract the ZIP to C:\QS\
  2. Open the solution at C:\QS\Quicksilver-master\Quicksilver.sln using Visual Studio.
  3. Navigate to **Build** | **Build Solution** or press *Ctrl + Shft + B*.
* After downloading NuGet packages while building the solution, the C:\QS\ folder will be about 680 MB.

### Configuring Quicksilver to use your Episerver Personalization account

1. Navigate to **Edit** | **Find and Replace** | **Find in Files** or press *Ctrl* + *Shift* + *F*.
2. For **Find what** enter **ChangeThis**, set **Look in** to **Entire Solution**, and click **Find All**, as shown in the screenshot:
3. Note the many changes that you could make, most of which can be left unset unless you want to configure authentication using social networking sites or deploy to Microsoft Azure, as shown in the following screenshot:



1. Double-click any of the four <add key="episerver:personalization entries in Web.config.
2. Enable tracking, add an entry to set the scope for track events to **epi-edu** (this will only apply to track events that do not have a scope explicitly set), and set the value attributes to what is shown in the following markup:

<add key="episerver:tracking.Enabled" value="true" />

<add key="episerver.profiles.Scope" value="epi-edu" />

<add key="episerver:personalization.BaseApiUrl" value="https://partnerdemo.peerius.com/ " />

<add key="episerver:personalization.Site" value="quicksilveredudemo" />

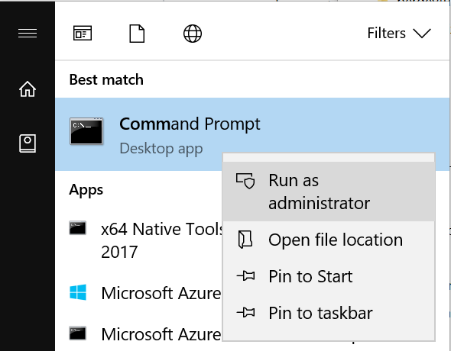
<add key="episerver:personalization.ClientToken" value="5269b51bc5bc086bb5a2a773f9b525ec0285719dfc72a01399f8a71ef18958b6" />

<add key="episerver:personalization.AdminToken" value="941a861158198154ba3a720d4012a05765a9165af7b636dc3c7baad1ca300670" />

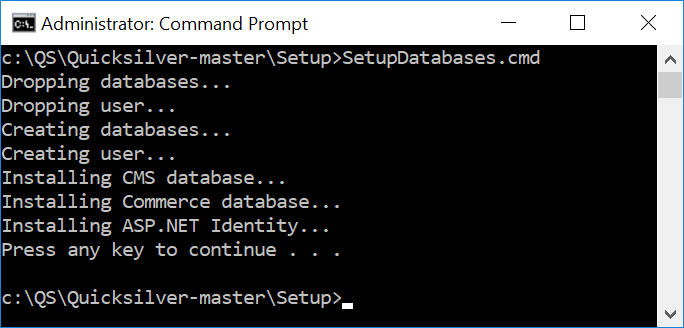
<add key="episerver:personalization.RequestTimeout" value="0:0:10" />

**Remarks:** These settings are for a demo account used for Episerver training. In a “real” environment you would need to enter your own configuration and keys assigned to you by Episerver.

1. Start **Command Prompt** as an administrator, as shown in the following screenshot:



1. Change to the C:\QS\Quicksilver-master\Setup\ directory.
2. Execute SetupDatabases.cmd, as shown in the following screenshot:



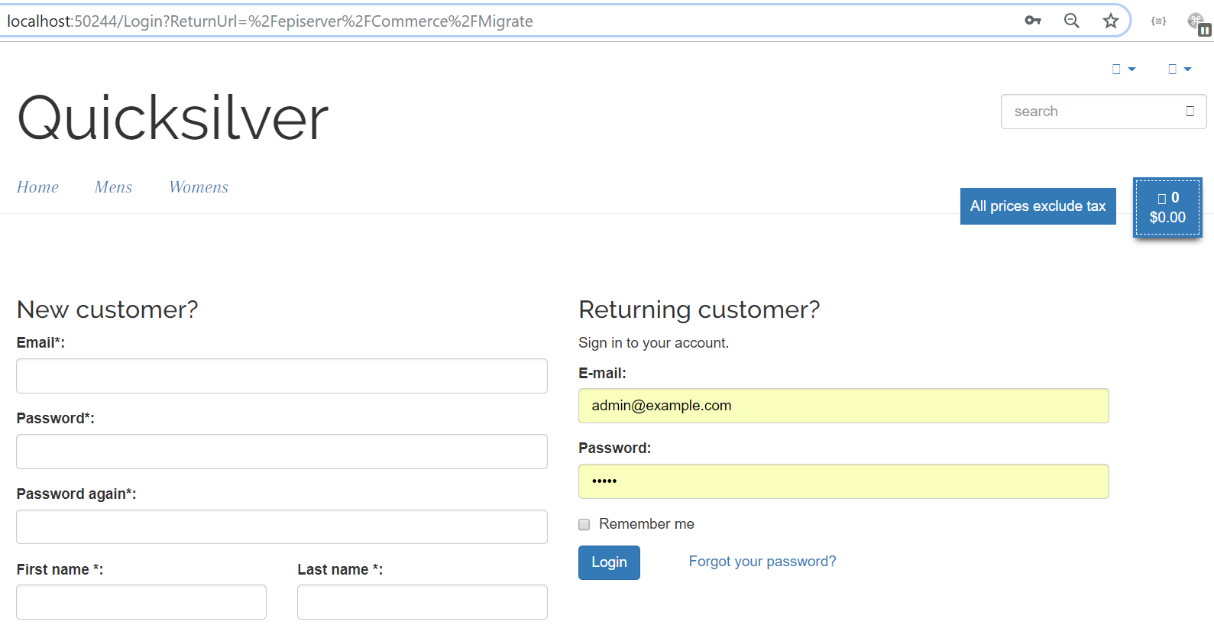
1. Type exit and press *Enter* to close **Command Prompt**.

### Starting Quicksilver for the first time and performing migration

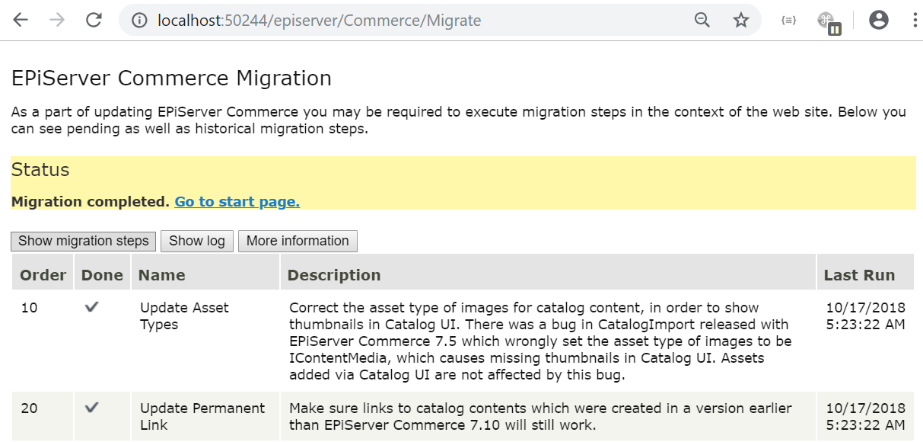
1. In Visual Studio, make sure you are set to use Chrome as your browser.
2. Start the Quicksilver site by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.

* Quicksilver is configured to use a fixed port number: http://localhost:50244/

1. On first run, you will be redirected to a migration page, which requires you to log in as an administrator, as shown in the following details and screenshot:
   1. E-mail: **admin@example.com**
   2. Password: **store**



1. Click **Login** and wait for migration to complete, as shown in the following screenshot:



1. Close the browser window and go back to Visual Studio to perform some package updates.

### Updating the Episerver NuGet packages

1. Right-click the **EPiServer.Reference.Commerce.Site** in the solution explorer panel and choose **Manage NuGet Packages…** from the menu.
2. Make sure the NuGet package source is set to **Episerver NuGets** and select the **Updates** tab.  
   A screenshot of a cell phone

   Description automatically generated
3. Check the **Select all packages** option and click the **Update** button to start the updates.  
   A close up of a logo

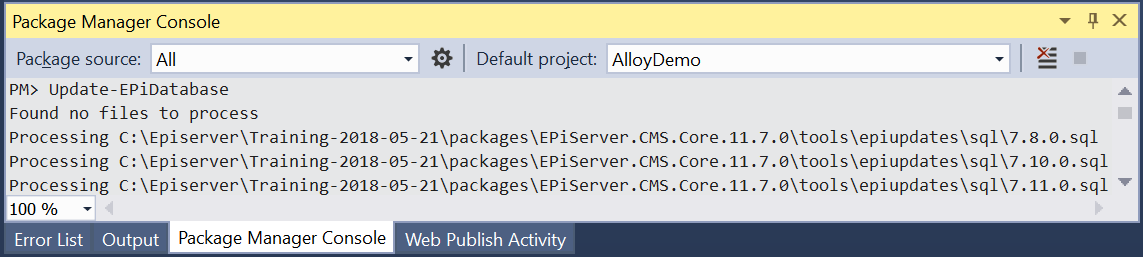
   Description automatically generated
4. Approve any dialogs from the updating packages and wait for the updates to complete.

### Updating the Episerver database

There are two ways to update the Episerver CMS database schema: manually with a console command, or automatically, with a Web.config setting, as explained in the error message. Recommended practice for deployments that you have control over is to perform migrations manually, especially if you have written custom SQL scripts to manipulate the CMS database schema, for example, to improve DDS performance. If you are deploying to DXC Service, we recommend using the Web.config setting.

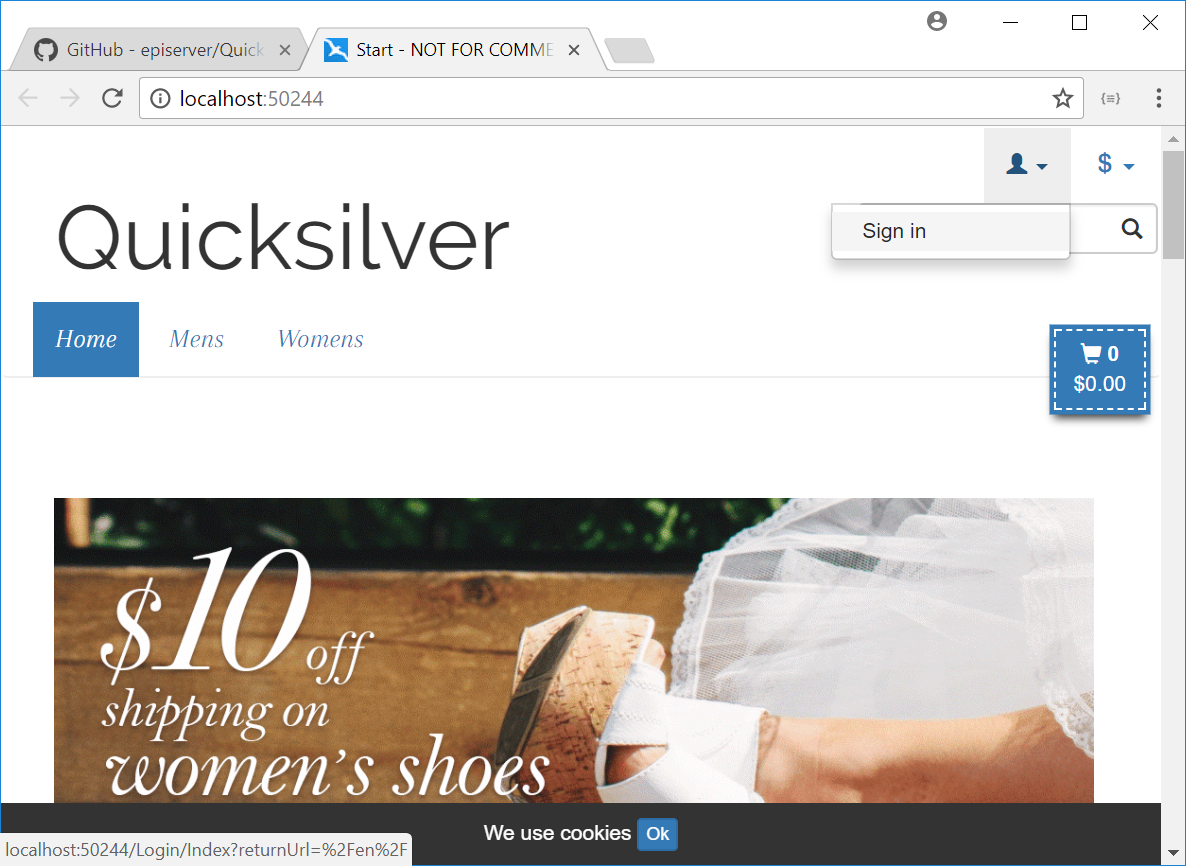
1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. Make sure the **Default project** is **AlloyDemo**, as shown in the following screenshot, and at the prompt enter:

Update-EPiDatabase



Ignore any messages that say **Found no files to process**, but if you get a warning about a missing **packages** folder, exit, and restart Visual Studio, and try again.

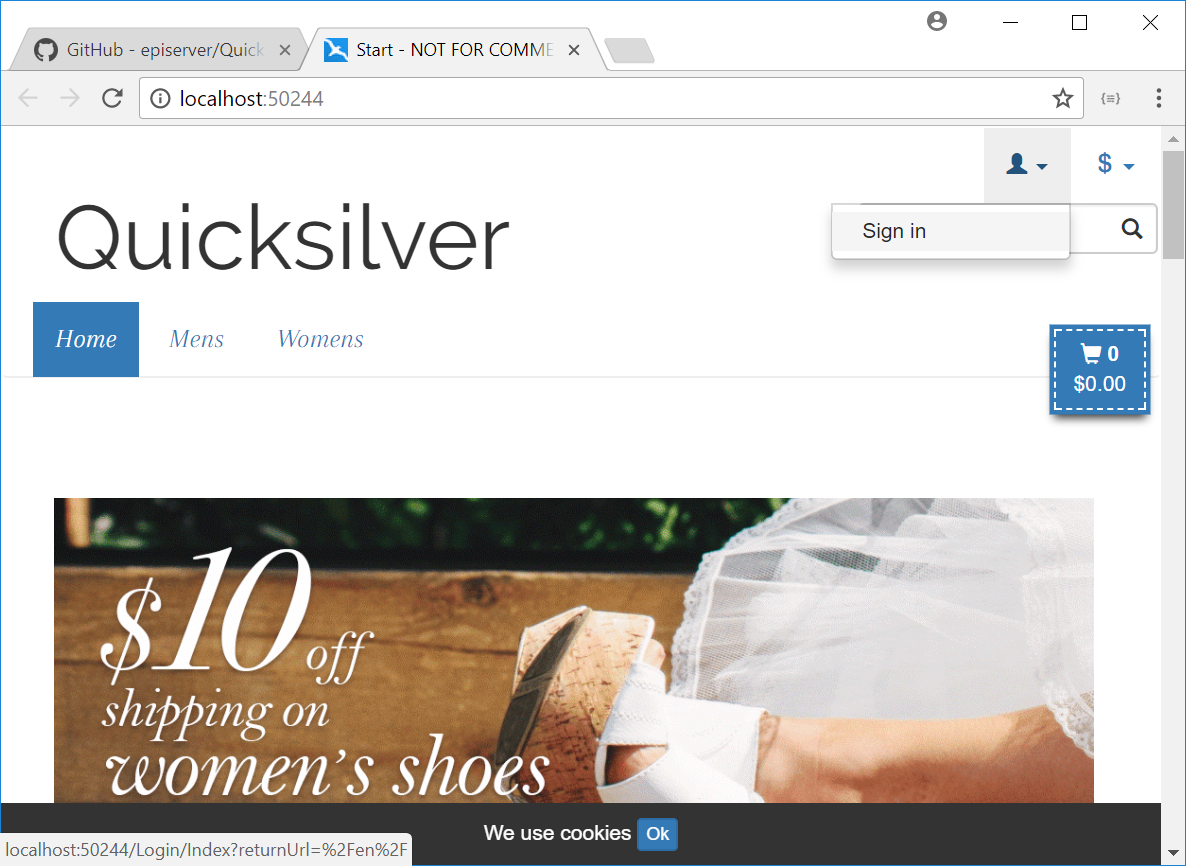
1. Start the Quicksilver site by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.
2. Log in again as [admin@example.com](mailto:admin@example.com) with the password of **store**.
3. Wait for the new round of migration tasks to complete.
4. Click **Go to start page**, and note the **Start** page, as shown in the following screenshot:



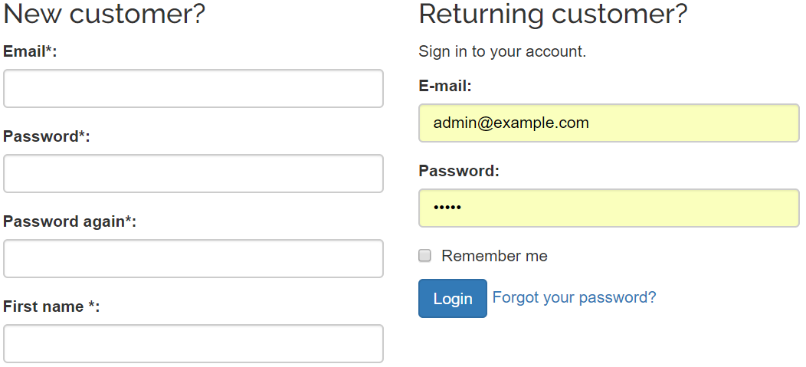
### (Optional) Exploring the Quicksilver site as a visitor

* If you are not familiar with the Quicksilver reference site, then use the following instructions to explore it from both a visitor and user point-of-view. If you already know it, then skip ahead to Exercise 2.

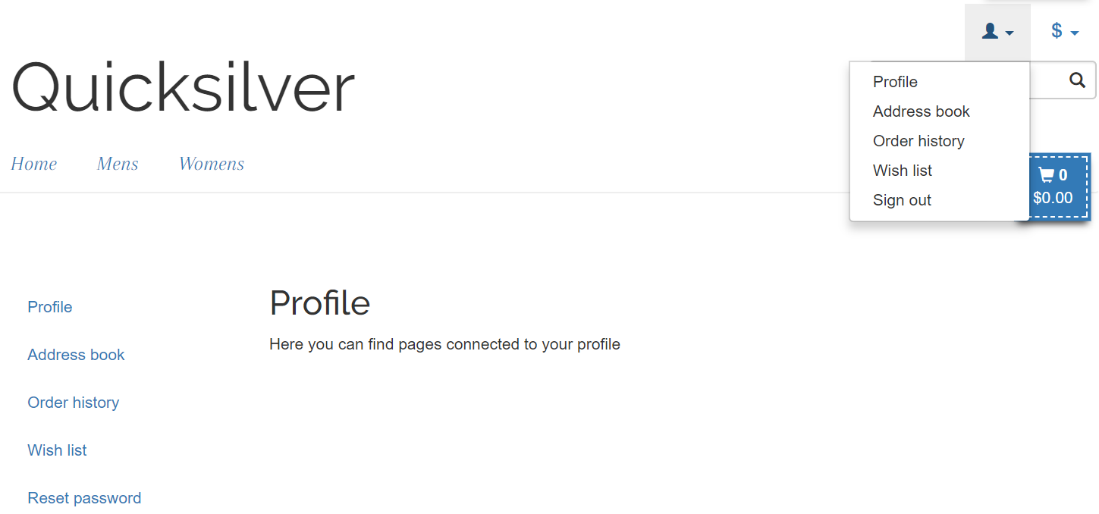
1. In Visual Studio, start the Quicksilver site by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.
2. From the Start page, click the account menu and choose **Sign In**, as shown in the following screenshot:

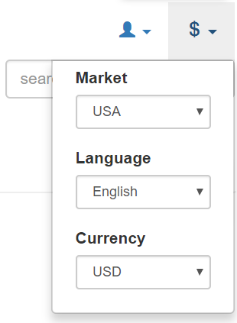


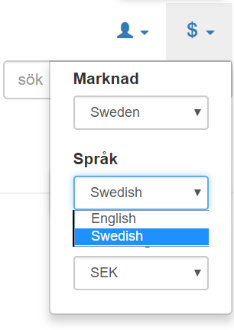
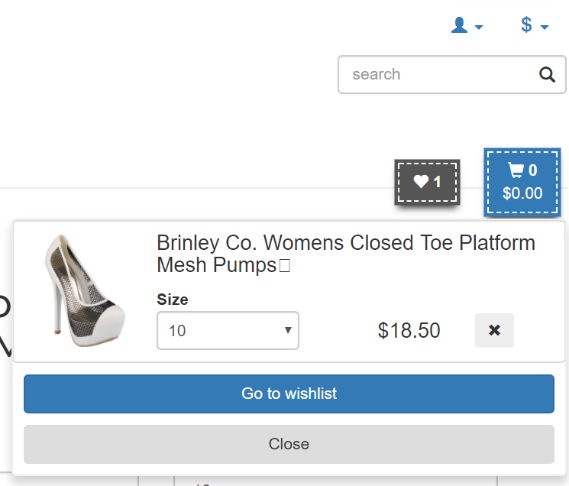
1. In the **Returning customer?** section, enter **admin@example.com** for the e-mail, and **store** for the password, and then click **Login**, as shown in the following screenshot:



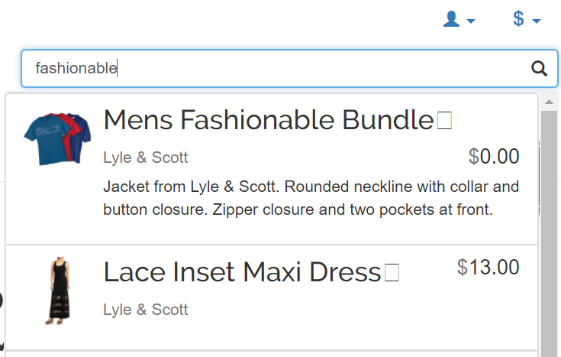
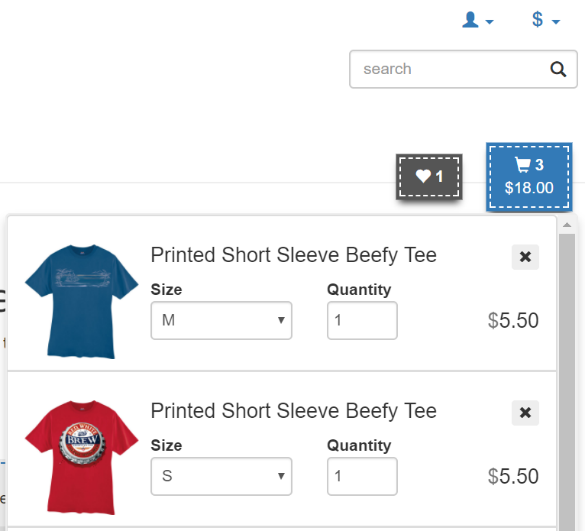
1. Note that after registering or signing in, a visitor can use the account menu to navigate to areas like their **Profile** and **Order history**, as shown in the following screenshot:



1. To the right of the account menu, the visitor can switch markets, as shown in the screenshot, and note the default market is **USA** which only has **English** for language and **USD** (US dollars) for currency.
2. Change the **Market** to **Sweden** and note the website user interface changes to the **Swedish** language. The **English** language is also available, and the currency can be either **SEK** (Swedish Krona) or USD (US dollars), as shown in the following screenshot:

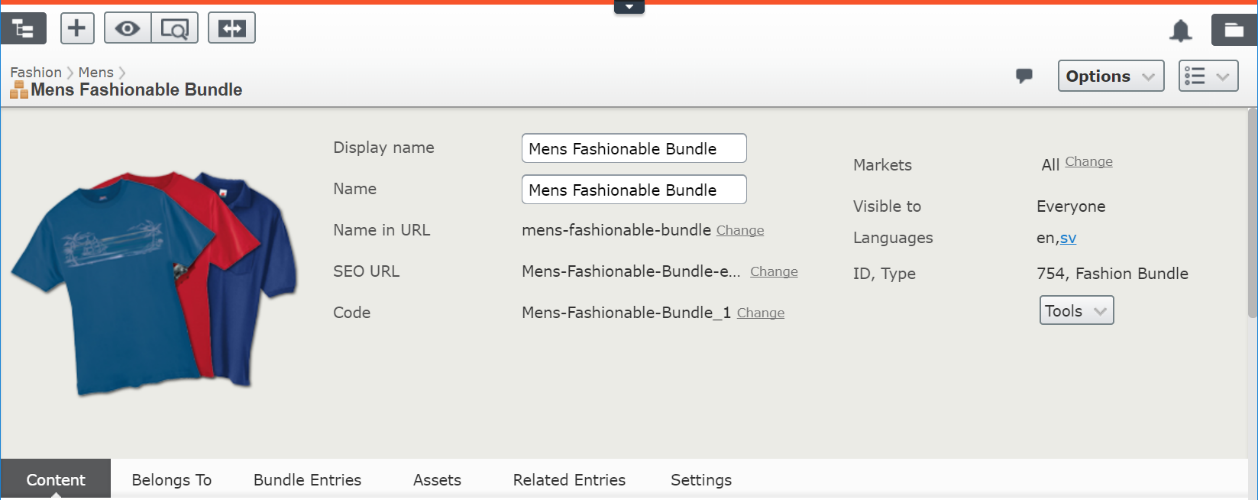


1. Change the **Market** back to **USA**.
2. Use the menu navigation to select a product and add it to your Wish List, as shown in the screenshot:
3. Use search to find a product and add it to your **Cart**, as shown in the following screenshots:

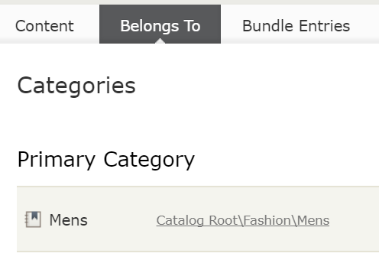
 

### (Optional) Exploring the Quicksilver site as a Commerce admin

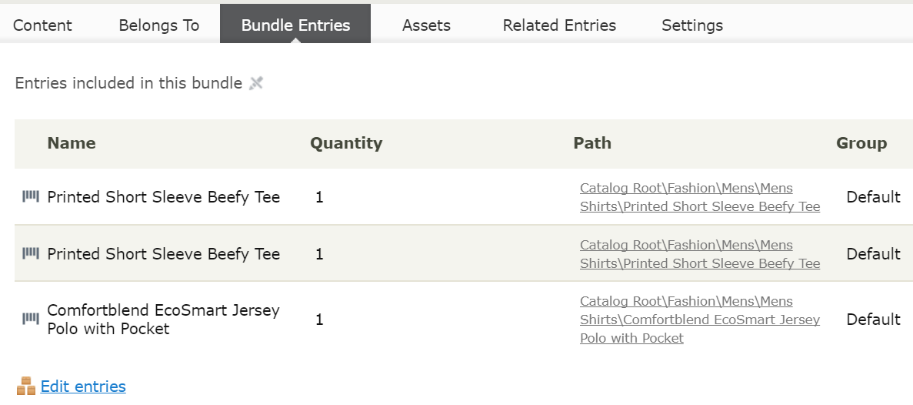
1. Search for the **Mens Fashionable Bundle** and view it.
2. Click the **epi** menu to switch to Edit view, as shown in the following screenshot:



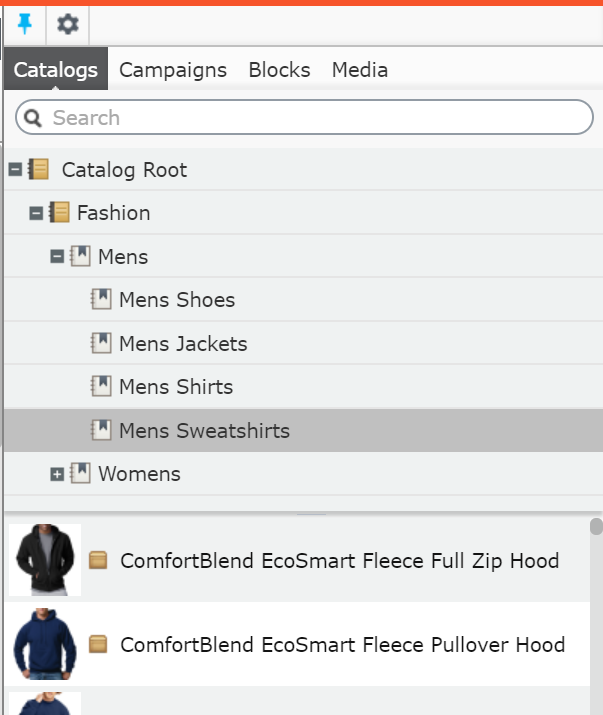
1. Navigate to **Belongs To**, and note the bundle’s primary category, as shown in the following screenshot:



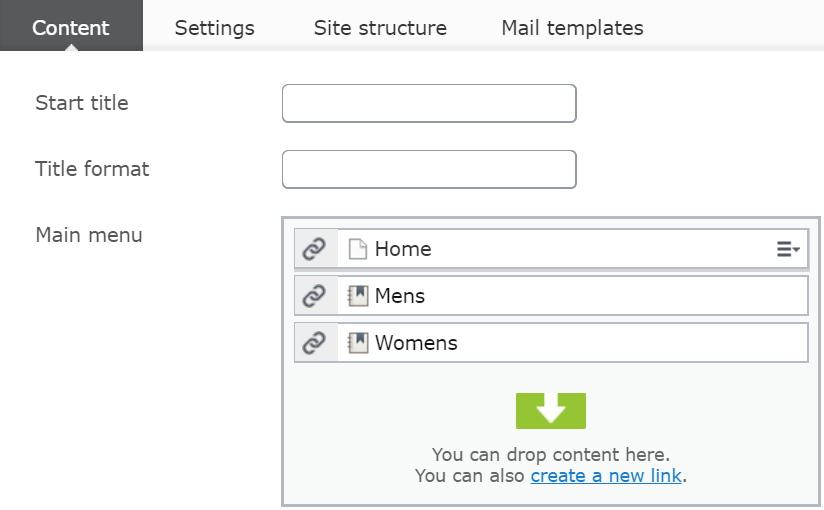
1. Navigate to **Bundle Entries**, and note the three entries, as shown in the following screenshot:



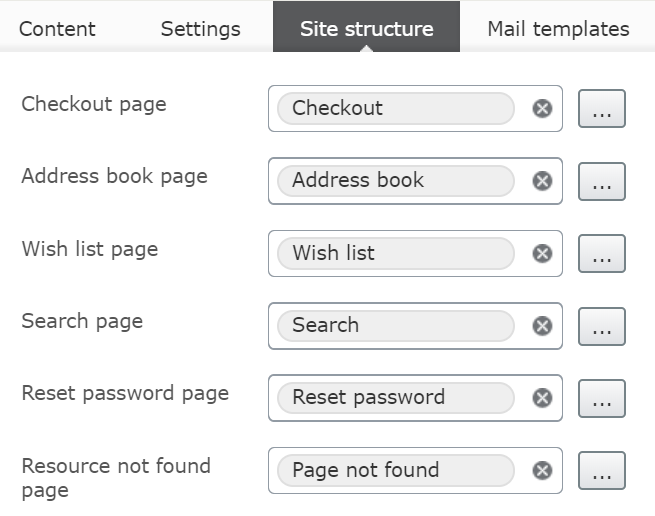
1. In the **Assets** pane, use **Catalog** tab to navigate around the catalog to different categories, products, and variants, as shown in the following screenshot:



1. In the **Navigation** pane, select **Pages**, select the **Start** page, switch to **All Properties** view, and note the page has a **Main menu** property with links to the **Home** page, and to the **Mens** and **Womens** categories in the Commerce catalog, as shown in the following screenshot:



1. Select the **Site structure** tab, and note the references to special pages within the site like a Checkout page and a Search page, as shown in the following screenshot:



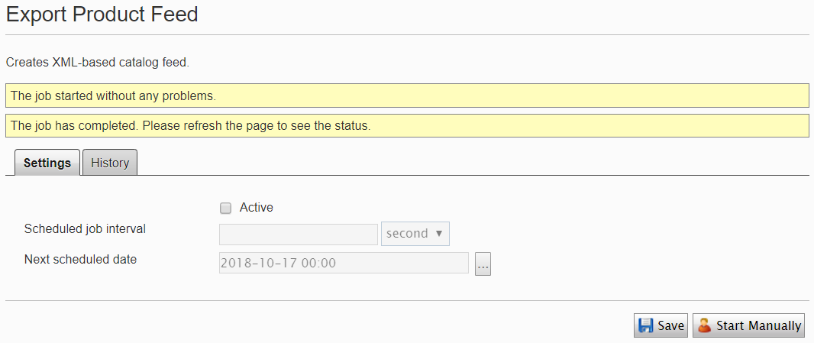
## Exercise 2 – Product Feed and Diagnostics

### Exporting the Quicksilver product feed

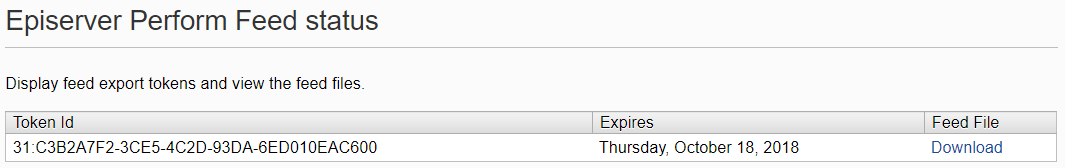
1. Navigate to **Package Manager Console**, and enter the following command:

Install-Package PeformFeedStatus -ProjectName EPiServer.Reference.Commerce.Site

1. Start the website without debugging.
2. Sign in as a returning customer using an e-mail of **admin@example.com** and a password of **store**.
3. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Export Product Feed**.
4. Click Start Manually, and note the job should complete successfully, as shown in the following screenshot:



1. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Peform Feed Status**, and note a security token was created and sent to Episerver Perform that expires in 24 hours, as shown in the following screenshot:

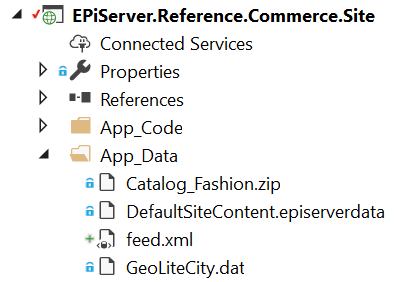


* When exported from a localhost site, the Episerver Perform engine will fail to download the export feed because it cannot access a non-publically hosted site, but you can still look at the product feed file.

1. Click **Download**, and note the URL to the exported feed, as shown in the following screenshot:



* A typical exported product feed uses the configured BLOB provider: http://localhost:50244/episerver/PerformFeedStatus/DownloadPerformFeed/?fileid=epi.fx.blob://default/d4a76096689649908bce5881979b7c1a/d25c49541e9d43fdafe99aff249958b9.zip

1. Open the folder the **Feed.zip** downloaded to and extract it.
2. Drag and drop the extracted **feed.xml** into the **App\_Data** folder, as shown in the screenshot:
3. Open **feed.xml**, and navigate to **Edit** | **Advanced** | **Format Document** or press *Ctrl* + *E*, *D*.
4. Review the product feed, referring to the online Episerver World documentation at the following link:

<https://world.episerver.com/documentation/developer-guides/personalization/data-import-guides2/catalog-feed-specification/rss-feed-content/mandatory-item-elements/>

### Adding diagnostics information to the Start page

1. Open the **EPiServer.Reference.Commerce.Site** project.
2. Open the Features\Start\Controllers\StartController.cs file.
3. In the class, add statements to import the following namespaces:

using EPiServer.Personalization.Commerce;

using EPiServer.Personalization.Commerce.Widgets;

using EPiServer.Tracking.Core;

1. In the class, add statements to declare fields to store personalization configuration service, widgets service, and tracking data interceptors, as shown in the following code:

private readonly PersonalizationClientConfiguration

personalizationClientConfiguration;

private readonly WidgetService widgetService;

private readonly IEnumerable<ITrackingDataInterceptor> trackingDataInterceptors;

1. In the constructor, add statements to pass the registered personalization configuration service, widgets service, and tracking data interceptors as parameters and to set the fields to those parameters.
2. In the Index method, add statements to get and store the widgets and tracking data interceptors in the ViewData dictionary keys named **widgets**, with its scope and page, and **interceptors**, sorted by SortOrder, as shown in the following code:

var widgets = Enumerable.Empty<Tuple<string, string, string, bool>>();

foreach (var scope in personalizationClientConfiguration.GetScopes())

{

WidgetsResponseData response = widgetService.GetWidgets(scope);

foreach (var page in response.EpiPerPage.Pages)

{

widgets = widgets.Concat(page.Widgets.Select(w =>

Tuple.Create(scope, page.PageType, w.WidgetName, w.Active)));

}

}

ViewData["widgets"] = widgets;

ViewData["interceptors"] = trackingDataInterceptors  
 .OrderBy(item => item.SortOrder);

* We have configured Quicksilver to be “scopeless” so there will only be one scope with an empty name.

1. Drag and drop from **perdevfun-exercisefiles\Module C\C1\TrainingExtensions.cs** into the **EPiServer.Reference.Commerce.Site** project.
2. Open the Views\Start\Index.cshtml file.

* To save you time, you can copy and paste from a solution for this file.

1. Add statements to import the EPiServer.Tracking.Core and EPiServer.Reference.Commerce.Site namespaces.
2. Add statements to output a collapsible panel with tables of widgets and tracking data interceptors including their SortOrder, Name (with FullName as title attribute), and description using the TrainingExtensions TrackingDataInterceptorDescriptions dictionary, as shown in the following markup:

<div class="panel-group">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title"><a data-toggle="collapse"

href="#collapseDiagnostics">Diagnostics</a></h3>

</div>

<div id="collapseDiagnostics" class="panel-collapse collapse">

<div class="panel-body">

@if (ViewData["widgets"] != null)

{

<div class="container-fluid">

<div class="row">

<div class="col-xs-12">

<h4>Widgets</h4>

<table class="table table-bordered table-condensed table-responsive">

<tr>

<th>Scope</th>

<th>Page</th>

<th>WidgetName</th>

<th>Active</th>

</tr>

@foreach (var widget in ViewData["widgets"] as

IEnumerable<Tuple<string, string, string, bool>>)

{

<tr>

<td>@widget.Item1</td>

<td>@widget.Item2</td>

<td>@widget.Item3</td>

<td>@widget.Item4</td>

</tr>

}

</table>

</div>

</div>

</div>

}

@if (ViewData["interceptors"] != null)

{

<div class="container-fluid">

<div class="row">

<div class="col-xs-12">

<h4>Tracking Data Interceptors</h4>

<table class="table table-bordered table-condensed table-responsive">

<tr>

<th>SortOrder</th>

<th>Type</th>

<th>Description</th>

</tr>

@foreach (var interceptor in ViewData["interceptors"] as IEnumerable<ITrackingDataInterceptor>)

{

<tr>

<td>@interceptor.SortOrder</td>

<td title="@interceptor.GetType()

.FullName">@interceptor.GetType().Name</td>

<td>

@if (TrainingExtensions

.TrackingDataInterceptorDescriptions.ContainsKey(interceptor.GetType().FullName))

{

<span>@TrainingExtensions

.TrackingDataInterceptorDescriptions[interceptor.GetType().FullName]</span>

}

</td>

</tr>

}

</table>

</div>

</div>

</div>

}

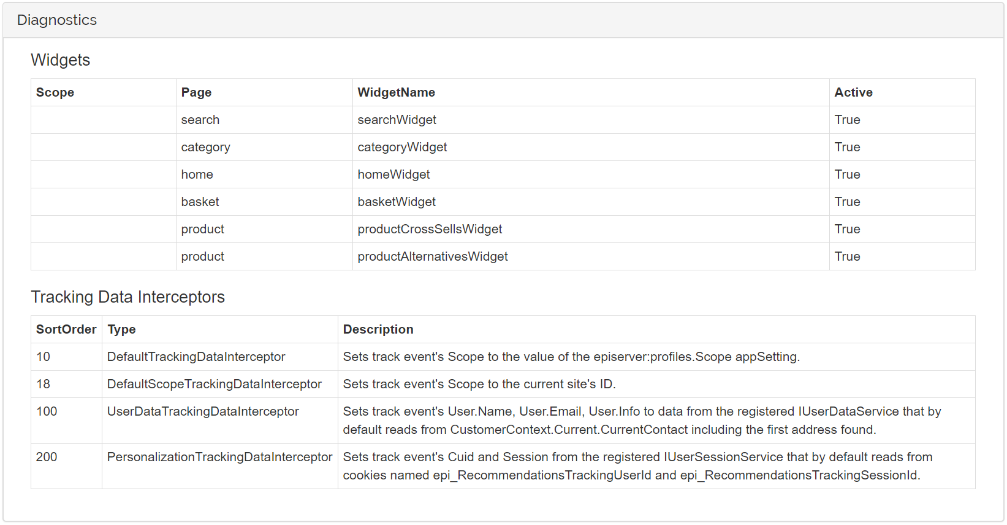
</div>

</div>

</div>

</div>

1. Start the website without debugging, click **Diagnostics**, and note the table showing the activated widgets and registered tracking data interceptors, and shown in the following screenshot:



1. Close the browser.

## Exercise 3 – Getting product recommendations in web pages

The Quicksilver site already has recommendations implemented on various pages, with the most current version using JavaScript and the client-side API. To make it easier to understand the code required for Product recommendations you are going to build new pages. This will allow you to see everything condensed into a single controller, view model and a few view templates. Note that this is just for learning purposes and is not an example of best practices. For best practice examples you should explore the Quicksilver’s default configuration.

For this exercise you are going to first see a couple ways to implement server-side API code and then, as an alternative, an example of the client-side API

### Implementing Server-side API Code with Tracking Attribute (Home Page Recommendations)

1. In the **Episerver.Reference.Commerce.Site** project, under the **Features** folder, create a new folder and name it “RecDemo”.
2. Create 3 subfolders under the new **RecDemo** folder and name them, “Controllers”, “Pages” and “ViewModels”. The folder structure should look like the following:  
     
   A close up of a sign

   Description automatically generated
3. Starting with a new page type, right-click the **Pages** folder and choose **Add -> New Item…**
4. In the **Add New Item** dialog, select **Episerver** from the left navigation panel and then choose **Page Type**.
5. Name the page type “RecDemoPage.cs” and click the **Add** button to create it.
6. Uncomment the **MainBody** property and attribute decorators and save the page.  
     
   **Remarks:** The main reason for creating the page type is to have a clean controller class, that you’ll create later, to use for adding the server-side recommendations code. For the page type you don’t need anything else defined for this exercise.
7. Next you’ll add a view model, right-click the **ViewModels** folder and choose **Add -> Class…**
8. Name the new class “RecDemoViewModel.cs” and click the **Add** button to create it.
9. Add the following class definition below the existing one, be sure to stay inside the namespace curly brace:  
     
   public class RecommendedProduct

{

public string Title { get; set; }

public string Url { get; set; }

public string ImageUrl { get; set; }

public string UnitPrice { get; set; }

public string SalePrice { get; set; }

public string Code { get; set; }

public long recommendationId { get; set; }

}  
   
**Remarks:** This class will be used to make it easier to pass recommendation information into the view template.

1. In the body of the original class **RecDemoViewModel**, add the following properties:

public string CodeValue { get; set; }

public RecDemoPage RecDemoPage { get; set; }

public IEnumerable<RecommendedProduct> HomeRecommendations { get; set; }

public IEnumerable<RecommendedProduct> ProductAltRecommendations { get; set; }

public IEnumerable<RecommendedProduct> ProductCrossSellRecommendations { get; set;

1. Fix the missing namespace for the page type.

**Remarks:** The three enumerable properties will be used to pass recommendation sets to the view templates.

1. For the controller, right-click the **Controllers** folder and choose **Add -> New Item…**
2. In the **Add New Item** dialog, select **Episerver** from the left navigation panel and then choose **Page Controller (MVC)**.
3. Name the page type “RecDemoController.cs” and click the **Add** button to create it.
4. Change the page type of the generic controller type to **RecDemoPage** and repeat that for the **Index** action method’s **currentPage** parameter type and fix the missing namespace. The result should look like the following:

public class RecDemoController : PageController<RecDemoPage>

{

public ActionResult Index(RecDemoPage currentPage)

{

return View(currentPage);

}

}

1. Add the following additional namespaces to the class definition:

using EPiServer.Reference.Commerce.Site.Features.RecDemo.ViewModels;

using EPiServer.Tracking.Commerce;

using EPiServer.Personalization.Commerce.Extensions;

using EPiServer.Personalization.Commerce.Tracking;

using EPiServer.Commerce.Catalog;

using Mediachase.Commerce.Pricing;

using EPiServer.Globalization;

using EPiServer.Commerce.Catalog.ContentTypes;

using Mediachase.Commerce;

using Mediachase.Commerce.Catalog;

using System;

using EPiServer.Commerce.Marketing;

using EPiServer.Tracking.Core;  
using EPiServer.Commerce.Order;

using Mediachase.Commerce.Customers;

1. Add the following private fields to the class definition:

private LanguageResolver \_languageResolver;

private ICurrentMarket \_currentMarket;

private IContentLoader \_contentLoader;

private AssetUrlResolver \_assetUrlResolver;

private IPriceService \_priceService;

private IPromotionEngine \_promotionEngine;

private ITrackingService \_trackingService;

private TrackingDataFactory \_trackingDataFactory;

private ReferenceConverter \_referenceConverter;

private IOrderRepository \_orderRepository;

private IOrderGroupFactory \_orderGroupFactory;

1. Add the following constructor to the class:

public RecDemoController(LanguageResolver languageResolver, ICurrentMarket currentMarket, IContentLoader contentLoader, AssetUrlResolver assetUrlResolver, IPriceService priceService, IPromotionEngine promotionEngine, ITrackingService trackingService, TrackingDataFactory trackingDataFactory, ReferenceConverter referenceConverter, IOrderRepository orderRepository, IOrderGroupFactory orderGroupFactory)

{

\_languageResolver = languageResolver;

\_currentMarket = currentMarket;

\_contentLoader = contentLoader;

\_assetUrlResolver = assetUrlResolver;

\_priceService = priceService;

\_promotionEngine = promotionEngine;

\_trackingService = trackingService;

\_trackingDataFactory = trackingDataFactory;

\_referenceConverter = referenceConverter;

\_orderRepository = orderRepository;

\_orderGroupFactory = orderGroupFactory;

}

1. Add the following private method member to the body of the **RecDemoController** class definition:

private IEnumerable<RecommendedProduct> FillRecsModel(IEnumerable<Recommendation> recommendations)

{

var recs = new List<RecommendedProduct>();

var catRefs = recommendations.Select(s => s.ContentLink);

var catItems = \_contentLoader.GetItems(catRefs, \_languageResolver.GetPreferredCulture());

var market = \_currentMarket.GetCurrentMarket();

var currency = \_currentMarket.GetCurrentMarket().DefaultCurrency;

foreach (var catItem in catItems)

{

var product = catItem as ProductContent;

var variantRef = product.GetVariants().FirstOrDefault();

var variant = \_contentLoader.Get<VariationContent>(variantRef);

var price = \_priceService.GetDefaultPrice(market.MarketId, DateTime.Now, new CatalogKey(variant.Code), currency);

var discountPrice = \_promotionEngine.GetDiscountPrices(variantRef, market, currency).FirstOrDefault()?.DiscountPrices.FirstOrDefault();

var recProd = new RecommendedProduct

{

Title = product.DisplayName,

UnitPrice = price?.UnitPrice.ToString(),

SalePrice = discountPrice?.Price.ToString(),

ImageUrl = \_assetUrlResolver.GetAssetUrl(product),

Code = product.Code,

recommendationId = recommendations.Where(r => r.ContentLink == product.ContentLink).FirstOrDefault().RecommendationId

};

recs.Add(recProd);

}

return recs;

}

1. Add the following recommendations attribute decorator to the top of the **Index** action result method:

[CommerceTracking(TrackingType.Home)]

**Remarks:** This is arguably the easiest way to get tracking started for the **Home** page in this example.

1. Add the following code into the body of the **Index** action result method:

var viewModel = new RecDemoViewModel();

viewModel.RecDemoPage = currentPage;

var recs = this.GetRecommendationGroups()

.Where(x => x.Area == "homeWidget")

.SelectMany(x => x.Recommendations);

viewModel.HomeRecommendations = FillRecsModel(recs);

return View(viewModel);

1. Right-click inside the **Index** action method body and choose **Add View…** from the menu.
2. Set the **Template** dropdown to **Empty**, the **Model class** to **RecDemoViewModel** and **un-check** all of the **Options**, it should look like the following:  
   A screenshot of a cell phone

   Description automatically generated
3. Click the **Add** button to complete the creation of the view template.
4. Add the following **using** statement to the top of the file:  
   @using System.Web.Optimization
5. Modify the contents of the **<head>** section with the following:  
     
   <meta name="viewport" content="width=device-width" />

<title>Recommendations Testing</title>

@Styles.Render("~/styles/bundled")

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

1. Replace the **<div>** element in the **<body>** with the following:

<div class="container">

@{ Html.RenderAction("Index", "Navigation"); }

<div class="h2">Testing Product Recommendations</div>

<div>

@Html.PropertyFor(m => m.RecDemoPage.MainBody)

</div>

<div class="row">

<div class="h3">Home Page Recommendations</div>

<div class="col-xs-2">

@foreach (var rec in Model.HomeRecommendations)

{

<div class="row text-center">@rec.Title</div>

<div class="product-row\_\_item\_\_image">

<img src="@rec.ImageUrl" class="img-responsive" />

</div>

<div class="row text-center">Retail: @rec.UnitPrice</div>

if (rec.SalePrice != null)

{

<div class="row text-center">Sale: @rec.SalePrice</div>

}

}

</div>

</div>

</div>

1. Before running the site, right-click the **EPiServer.Reference.Commerce.Site** project in Solution Explorer and click **Properties** from the menu.
2. On the left navigation panel choose **Web** and then select the **Specific Page** radio button option for the **Start Action**, leave the field blank and save the setting and close the tab.

A screenshot of a social media post

Description automatically generated  
**Remarks:** This ensures that when you run the site with a razor file in the editor’s for-front Visual Studio won’t try to be “helpful” and automatically navigate to the view.

1. **Run** the site without debugging.
2. Log in as **Admin** use the orange drop-down in the upper right corner of the page to navigate to the **CMS Edit** view.
3. In the navigation pane, click the “hamburger” icon next to the **Start** page and choose **New Page** from the menu.  
   A screenshot of a cell phone

   Description automatically generated
4. From the page template choices, select the **RecDemoPage** and name it “RecTesting”, click **OK** to save it.
5. Enter some descriptive text into the **Main Body** field such as “This is the home page for testing Product Recommendations.”
6. **Publish** the page.
7. Click the **Options** dropdown menu and then click the **View on website** link.  
   A screenshot of a cell phone

   Description automatically generated
8. The browser should show the **http://localhost:50244/en/RecTesting** page that includes recommendations from the service.

### Implementing Server-side API Code without Tracking Attribute (Product Recommendatioins)

1. In the **RecDemoController** class add the following:

public PartialViewResult ProductRecommendations(string code, long recommendationId)

{

var viewModel = new RecDemoViewModel();

viewModel.CodeValue = code;

return PartialView(viewModel);

}

1. Add the following lines into the new **ProductionRecommendations** partial view result method before the **return** line.

var trackData = \_trackingDataFactory.CreateProductTrackingData(code, recommendationId, HttpContext);

var prodRef = \_referenceConverter.GetContentLink(code);

var prod = \_contentLoader.Get<EntryContentBase>(prodRef);

var data = \_trackingService.Track(trackData, HttpContext, prod);

**Remarks:** The first line creates an object that will be used to pass information on to the service, note the recommendation ID. The call to the **Track** method is what actually makes a call to the Product Recommendations service passing in the tracking data object along with the HTTP context and **IContent** object you want to get recommendations back for. Note that there is an **TrackAsync** version of this method you could use instead.

1. Add the following below the previously added lines:

var prodAlts = data.GetRecommendationGroups()

.Where(x => x.Area == "productAlternativesWidget")

.SelectMany(x => x.Recommendations);

viewModel.ProductAltRecommendations = FillRecsModel(prodAlts);

var prodCrossSell = data.GetRecommendationGroups()

.Where(x => x.Area == "productCrossSellsWidget")

.SelectMany(x => x.Recommendations);

viewModel.ProductCrossSellRecommendations = FillRecsModel(prodCrossSell);

**Remarks:** If the call to **Track** was successful, the results can be easily obtained by using the same **GetRecommendationGroups** that was used in the **Index** action method with the attribute tracking. You do have to know what the name of the “widgets” are to pass into the **Area** property. In the case of product tracking, there are two results, “cross sells” and “alternatives”.

1. Right-click inside the **ProductRecommendations** partial view result method body and choose **Add View…** from the menu.
2. Set the **Template** dropdown to **Empty**, the **Model class** to **RecDemoViewModel** and **check** the **Create as a partial view option**, it should look like the following:

A screenshot of a cell phone

Description automatically generated

1. Click the **Add** button to complete the creation of the view template.
2. Add the following code to the new **ProductRecommendations** view template razor file:

<div class="row">

<div class="h3">Product Recommendations</div>

<div>Refrerring Product Code: @Model.CodeValue</div>

</div>

<div class="row">

<div class="col-xs-6">

<div class="h4">Product Alternative Recommendations</div>

<div class="col-xs-3">

@foreach (var rec in Model.ProductAltRecommendations)

{

<div class="row " style="border: 2px solid grey; margin-bottom: 2px">

<div class="text-center">@rec.Title</div>

<div class="product-row\_\_item\_\_image">

<img src="@rec.ImageUrl" class="img-responsive" />

</div>

<div class="row text-center">Retail: @rec.UnitPrice</div>

@if (rec.SalePrice != null)

{

<div class="row text-center">Sale: @rec.SalePrice</div>

}

</div>

}

</div>

</div>

<div class="col-xs-6">

<div class="h4">Product Cross-Sell Recommendations</div>

<div class="col-xs-3">

@foreach (var rec in Model.ProductCrossSellRecommendations)

{

<div class="row " style="border: 2px solid grey; margin-bottom: 2px">

<div class="text-center">@rec.Title</div>

<div class="product-row\_\_item\_\_image">

<img src="@rec.ImageUrl" class="img-responsive" />

</div>

<div class="row text-center">Retail: @rec.UnitPrice</div>

@if (rec.SalePrice != null)

{

<div class="row text-center">Sale: @rec.SalePrice</div>

}

</div>

}

</div>

</div>

</div>

**Remarks:** I want to retrieve this new partial view from the Index view template, and to make it a bit more efficient, I’m going add an Ajax package to the project so I can call it from the client browser asynchronously.

1. In Solution Explorer right-click the **EPiServer.Reference.Commerce.Site** project the select the **Manage NuGet Packages…** from the menu.
2. Make sure **nuget.org** is the Package source and then use the **Browse** tab to find and install the **Microsoft.jQuery.Unobtrusive.Ajax** package and any required dependencies.
3. In Solution Explorer, open the **Scripts** folder and delete the **jquery-1.8.0.xxxx** files that where added as dependencies for the Ajax package.  
   A screenshot of text

   Description automatically generated  
   **Remarks:** There was already a newer version of the jquery script in the folder so removing these will eliminate any possible confusion.
4. Navigate back into the **index.cshtml** razor view template for the **RecDemoController**.
5. At the top of the razor page, inside the **<head>** section add the following new script reference for the Ajax calls:

<script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>

1. Add the following code inside the **foreach** loop of the view model’s **HomeRecommendations** property at the end of the code block just below the **if** condition block:

<div class="row">

@Ajax.ActionLink("Get Product Recommendations", "ProductRecommendations", new { rec.Code, rec.recommendationId }, new AjaxOptions { UpdateTargetId = "productRecs" }, new { @class = "btn btn-primary" })

</div>

**Remarks:** The Ajax version of the **ActionLink** helper method will asynchronously call back to the server and retrieve the HTML fragment returned in the partial view. The results are then injected into an element with the **id** attribute of “productRecs” that was targeted in the Ajax options. We need to now add an element with that **id** value.

1. Add the following **div** block to the end of the view template just before the **last** **</div>** end element:

<div class="row">

<div id="productRecs"></div>

</div>

1. Run the site and manually navigate to <http://localhost:50244/en/RecTesting>.
2. You should see at leas one recommendation (try refreshing the page if you do not), click the **Get Product Recommendations** button on any of the home page recommendations you have, for example:

A screenshot of a social media post

Description automatically generated

1. Scroll down the page and verify that you are getting both cross sell and alternative recommendations for the product you clicked, example:

A screenshot of a social media post

Description automatically generated

### Implementing Client-side API Code (JavaScript)

1. Open the **ProductRecommendations.cshtml** razor view template and add the following code inside the **foreach** loop for both the cross-sell and alternative recommendations, it should be the last inner **div** of the top level **div** in the loop.

<div class="row">

@Html.ActionLink("Add to cart", "FakeCArt", new { rec.Code, rec.recommendationId }, new { @class = "btn btn-primary" })

</div>

**Remarks:** This will call back to the **RecDemoController** and to an action method named **“FakeCart”** that we’ll add next.

1. Navigate to the **RecDemoController.cs** file and add the following action result method to the class definition:

public ActionResult FakeCart(string code, long recommendationId)

{

var cart = \_orderRepository.LoadOrCreateCart<ICart>(CustomerContext.Current.CurrentContactId, "Default");

var prodRef = \_referenceConverter.GetContentLink(code);

var prod = \_contentLoader.Get<ProductContent>(prodRef);

var varRef = prod.GetVariants().FirstOrDefault();

var varContent = \_contentLoader.Get<VariationContent>(varRef);

var lineItem = \_orderGroupFactory.CreateLineItem(varContent.Code, cart);

lineItem.PlacedPrice = \_priceService.GetDefaultPrice(\_currentMarket.GetCurrentMarket().MarketId, DateTime.Now, new CatalogKey(varContent.Code), \_currentMarket.GetCurrentMarket().DefaultCurrency).UnitPrice;

lineItem.Quantity = 1;

cart.AddLineItem(lineItem);

\_orderRepository.Save(cart);

ViewBag.ProdCode = code;

ViewBag.RecId = recommendationId;

return View();

}

**Remarks:** I named the method “FakeCart” but it actually adds the product to the default cart for the site. Note that there is nothing dealing with the Product Recommendations service in this code, it is just plain Commerce code.

1. Right-click inside the **FakeCart** action method body and choose **Add View…** from the menu.
2. Set the **Template** dropdown to **Empty (without model)**, all the **Options** should be **un-checked**, it should look like the following:  
   A screenshot of a cell phone

   Description automatically generated
3. Click the **Add** button to complete creating the new view template.
4. Add the following namespaces to the top of the **FakeCart.cshtml** razor file:

@using System.Web.Optimization

@using EPiServer.Personalization.Commerce.Extensions

1. Add the following code inside the **<head>** section:

@Styles.Render("~/styles/bundled")

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@Html.LoadTrackingAPI()

**Remarks:** It’s the **LoadTrackingAPI** extension method that will render the client-side tracking script links.

1. Add the following code into the **<body>** of the page:

<div class="h1">You've added product code: @ViewBag.ProdCode to your cart!</div>

<div class="h2">Basket Recommendations</div>

<div class="recommendations recommend-stuff"></div>

@Scripts.Render("~/bundles/mustache")

@Scripts.Render("~/bundles/js")

**Remarks:** The **div** element with the class attribute set to **recommend-stuff** is the place-holder for where the recommendations returned from the service. The “mustache” script reference is a standard template system that the client-side Product Recommendations uses and the “js” represents a folder with various helper JavaScript files.

1. Add the following scripts blocks to the view template:

<script id="epiRecommendationListTemplate" type="x-tmpl-mustache">

**{{#recs}}**

<div class="jsProductTile product-row\_\_tile" data-recommendation-id="**{{**id**}}**">**{{>** epiRecommendationItemTemplate**}}**</div>

**{{/recs}}**

</script>

<script id="epiRecommendationItemTemplate" type="x-tmpl-mustache">

**{{#hasDiscount}}**

<div class="product has-discount">

**{{/hasDiscount}}**

**{{^hasDiscount}}**<div class="product">

**{{/hasDiscount}}**

<a href="**{{#attributes}}{{**url**}}{{/attributes}}**&recommendationId=**{{**id**}}**" class="link--black">

<div class="view-details"></div>

<img src="**{{#attributes}}{{**img**}}{{/attributes}}**" alt="**{{**refCode**}}**" class="img-responsive" />

<h3 class="product-title">**{{#attributes}}{{**title**}}{{/attributes}}**</h3>

<div>

**{{#hasDiscount}}**

<h4 class="product-price">**{{#attributes}}{{**unitPrice**}}{{/attributes}}**</h4>

<h4 class="product-price product-price--discount">**{{#attributes}}{{**salePrice**}}{{/attributes}}**</h4>

**{{/hasDiscount}}**

**{{^hasDiscount}}**

<h4 class="product-price">**{{#attributes}}{{**salePrice**}}{{/attributes}}**</h4>

**{{/hasDiscount}}**

</div>

</a>

</div>

<div class="quick-view-btn-container">

<button type="button" data-toggle="modal" data-target="#ModalDialog" data-url="**{{#attributes}}{{**url**}}{{/attributes}}**&recommendationId=**{{**id**}}**" class="btn btn-block btn-sm quickview-button">@Html.Translate("/Product/Quickview")</button>

</div>

</div>

</script>

<script>

$(document).ready(function () {

var refCode = '@ViewBag.ProdCode';

var basketTrackingData = TrackingDataFactory.createBasketTrackingData();

var recommendationId = parseInt(isNaN('@Request.QueryString["recommendationId"]') ? '0' : '@Request.QueryString["recommendationId"]');

if (recommendationId > 0) {

basketTrackingData["recommendationId"] = recommendationId;

}

basketTrackingData["skipRecommendations"] = false;

epiRecommendations.track(

basketTrackingData,

null,

'web',

Recommendations.render,

{

sectionMappings: [

{ area: "basketWidget", selector: ".recommend-stuff" }

]

}

);

});

</script>

1. Run the site and manually navigate to <http://localhost:50244/en/RecTesting>.
2. You should see at least one recommendation (try refreshing the page if you do not), click the **Get Product Recommendations** button on any of the home page recommendations you have.
3. Scroll down the page and click one of the **Add to cart** buttons:  
   A screenshot of a cell phone

   Description automatically generated
4. Verify that you get basket recommendations similar to the following:  
   A screenshot of a cell phone

   Description automatically generated

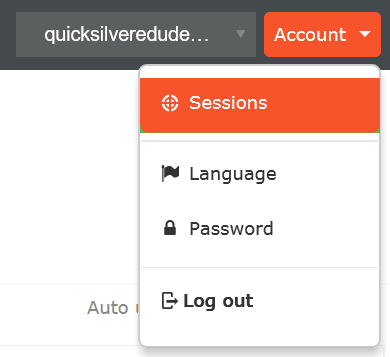
## Exercise 4 – Trouble Shooting and Session Viewer

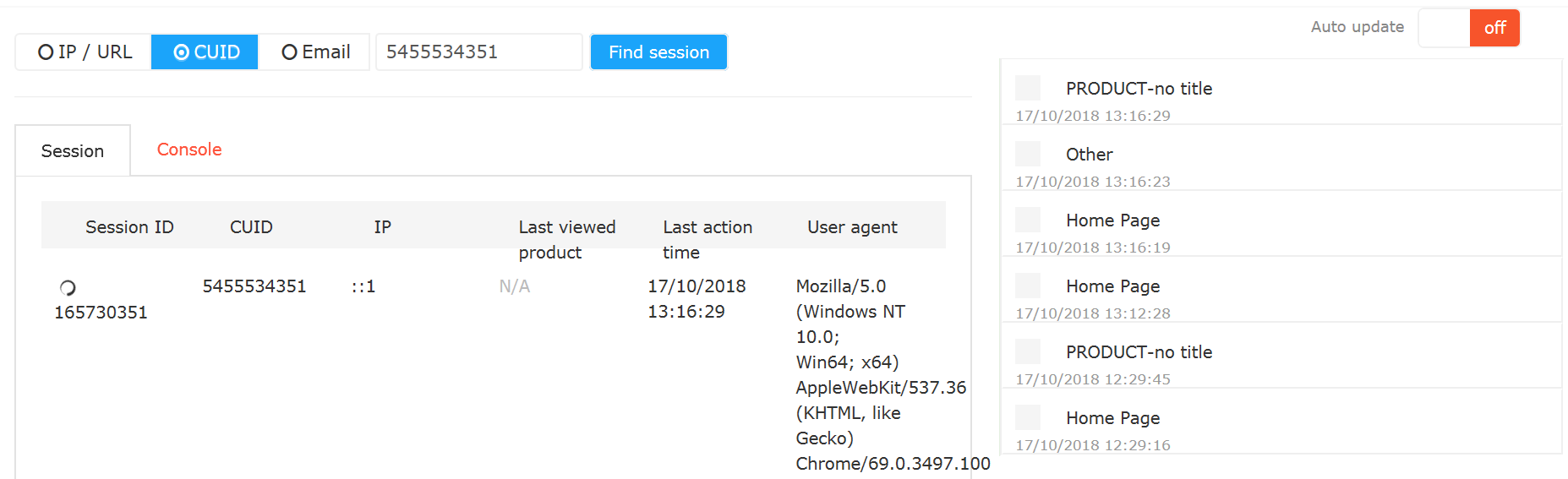
1. Run the site and manually navigate to <http://localhost:50244/en/RecTesting>.
2. Click on one of the products on the page.
3. In Chrome, open the developer tools (Ctrl + Shift + I).
4. In Chrome’s developer tools click the **Application** tab and then click the in the left navigation panel click the [**http**://localhost:50244](http://localhost:50244) cookie under the Cookies storage container:

A screenshot of a cell phone

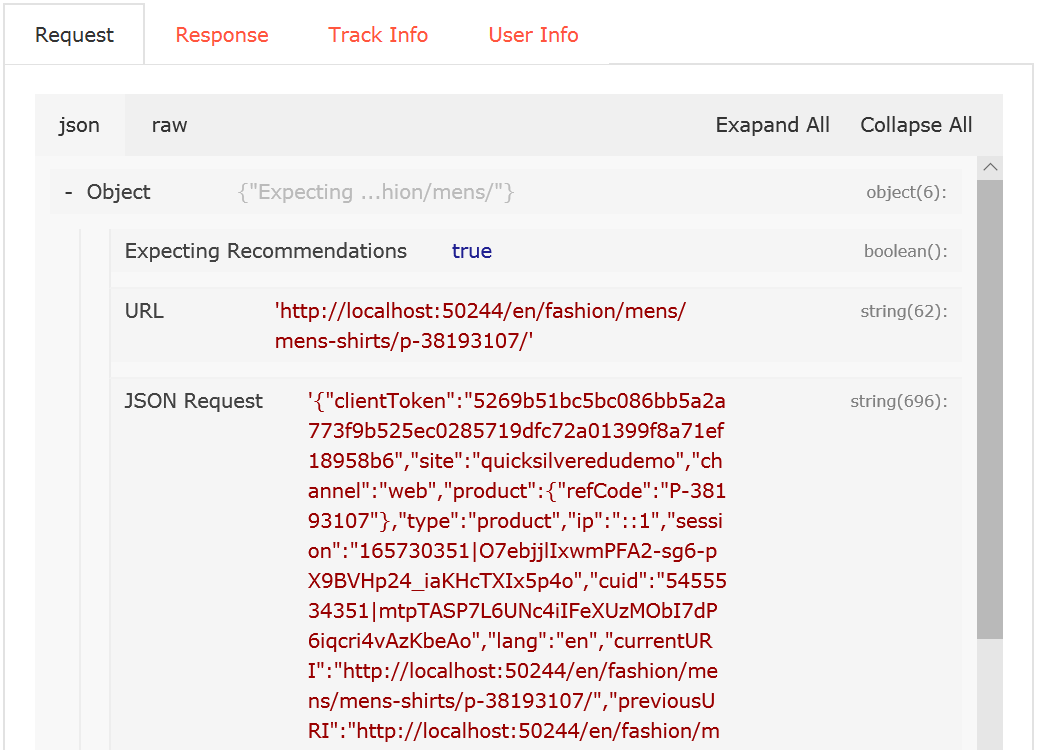
Description automatically generated

1. Click **epi\_RecommendationsTrackingUserId** from the list of cookies.
2. In the content of the cookie, copy the digits of the **cuid** up to the pipe (vertical bar) character into a text file for later use, for example: 6005344501:  
   A screenshot of a cell phone

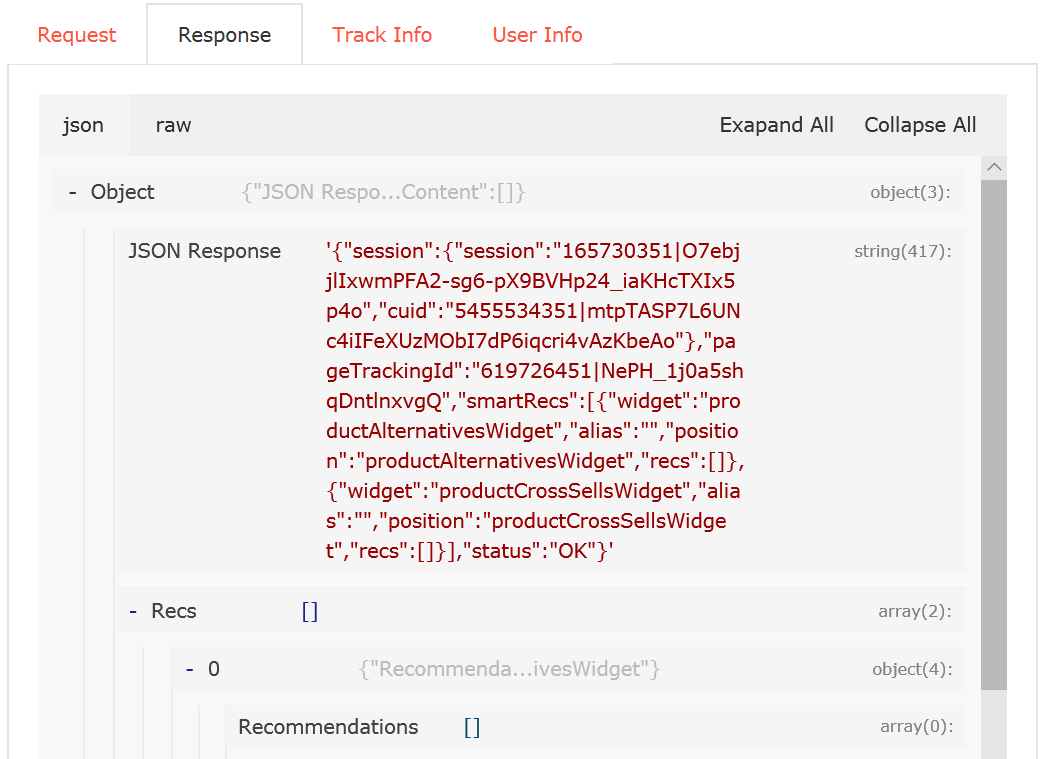
   Description automatically generated
3. Close Chrome’s developer tools window.
4. Start a new browser window and log in to the Episerver Personalization portal.
5. Navigate to **Account** | **Sessions**, as shown in the screenshot:
6. In **Session info**, select **CUID**, enter the cuid number from the cookie, click **Find session**, and then click the found session. You will see a list of the pages that the visitor has viewed, as shown in the following screenshot:



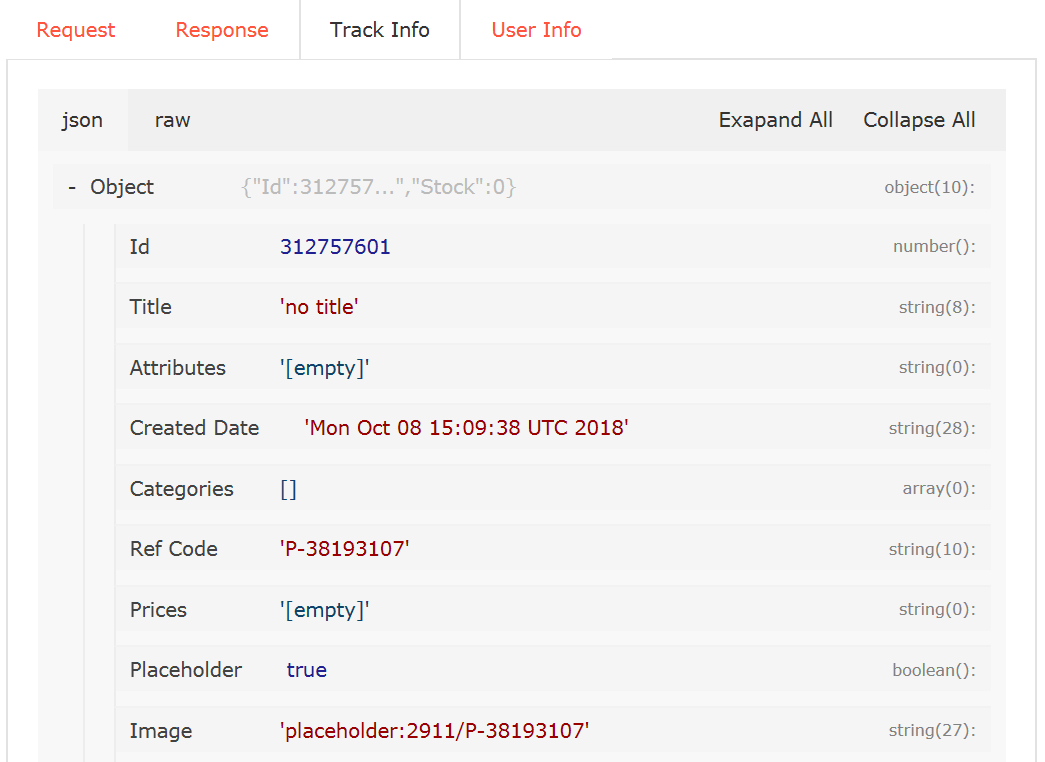
1. Click a product entry, and note the detailed track event information that was recorded, including the **URL** as **localhost**, the **site** as **quicksilveredudemo**, the product’s **refCode**, and the **cuid**, as shown in the following screenshot:



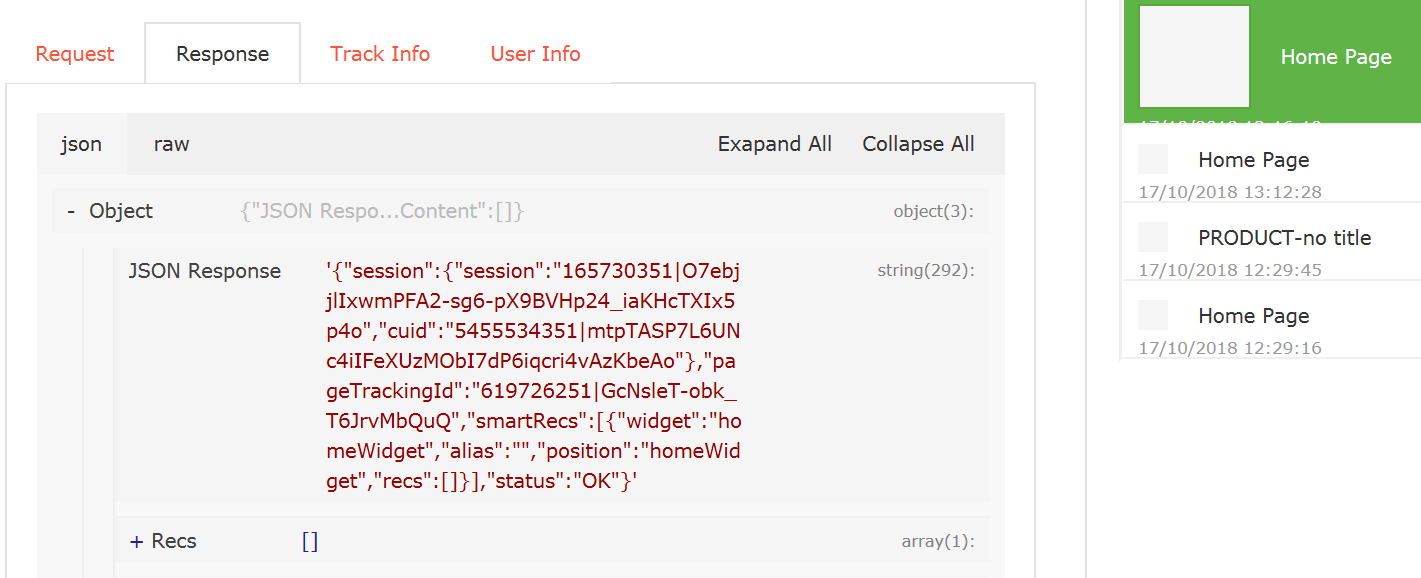
1. Click the **Response** tab, and note the cuid and the arrays of recommendations for two product page widgets, as shown in the following screenshot:



1. Click **Track Info**, and note the product details, as shown in the following screenshot:



1. Click one of the home entries, and note the cuid and the array of recommendations for the home page widget, as shown in the following screenshot:



1. Close the portal.

### Implementing a debugger tracking data interceptor

1. Add a class named DebuggerTrackingDataInterceptor, and modify its code to implement a tracking data interceptor that stores the event type, user, and payload in local variables and then breaks if the debugger is attached, as shown in the following code:

using EPiServer.ServiceLocation;

using EPiServer.Tracking.Core;

using System.Diagnostics;

namespace EPiServer.Reference.Commerce.Site

{

[ServiceConfiguration(ServiceType = typeof(ITrackingDataInterceptor),

Lifecycle = ServiceInstanceScope.Singleton)]

public class DebuggerTrackingDataInterceptor : ITrackingDataInterceptor

{

public int SortOrder => int.MaxValue;

public void Intercept<TPayload>(TrackingData<TPayload> trackingData)

{

if (Debugger.IsAttached)

{

string eventType = trackingData.EventType;

UserData user = trackingData.User;

TPayload payload = trackingData.Payload;

Debugger.Break();

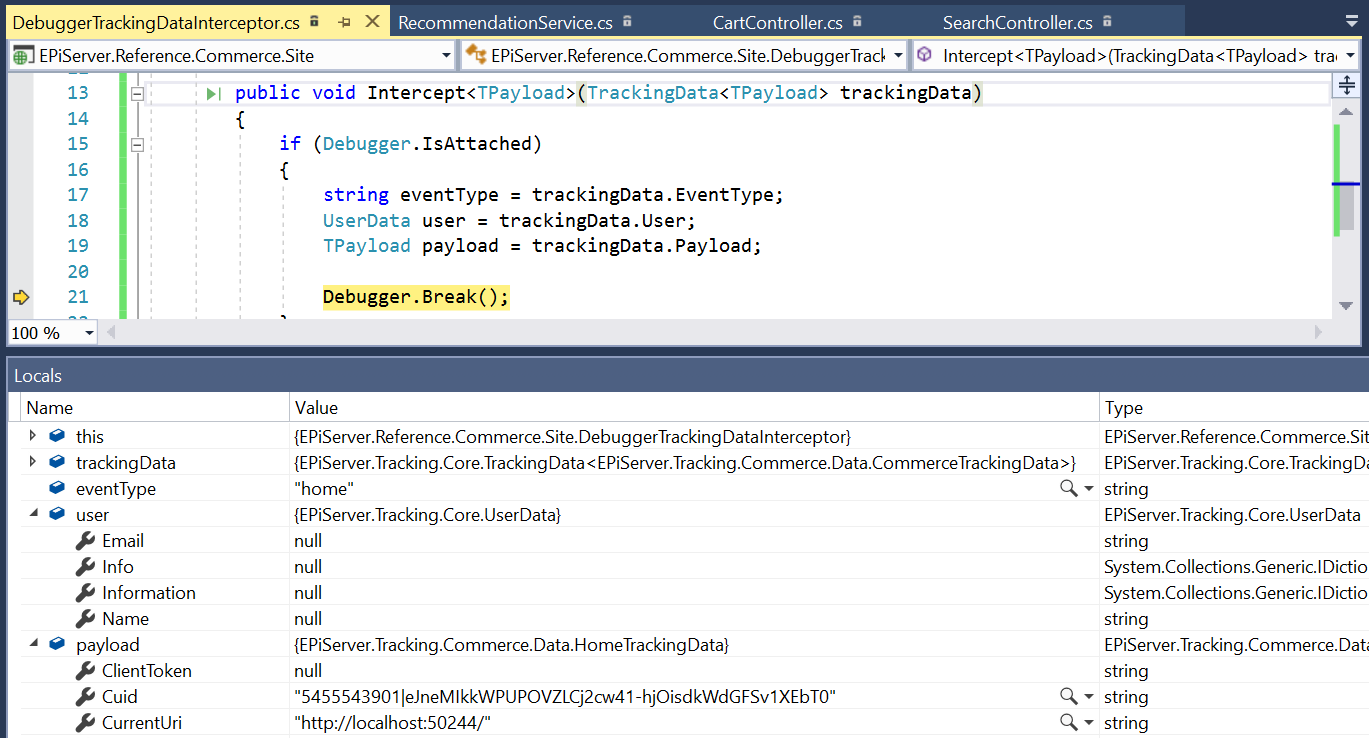
}

}

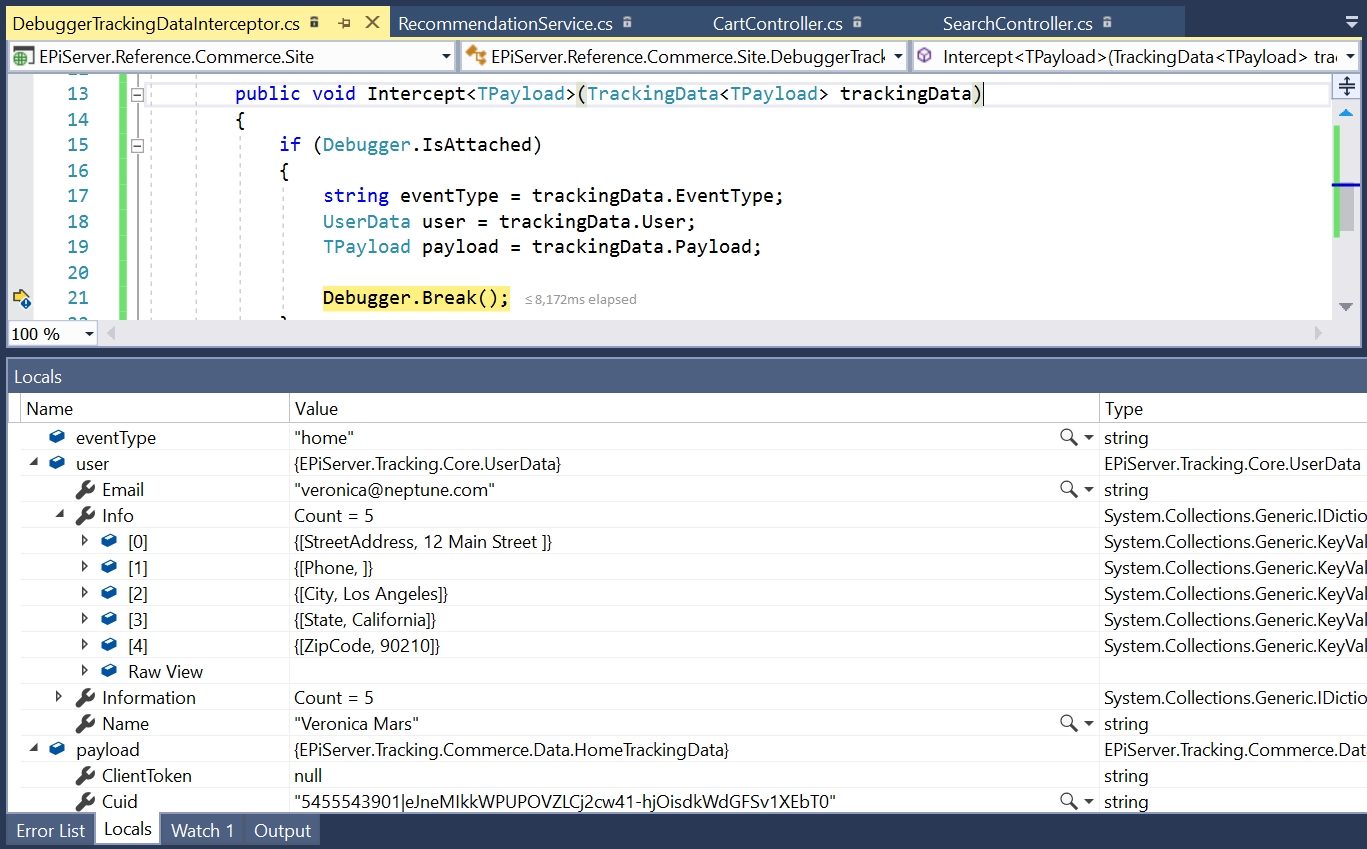
}

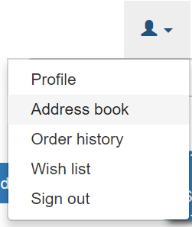
}

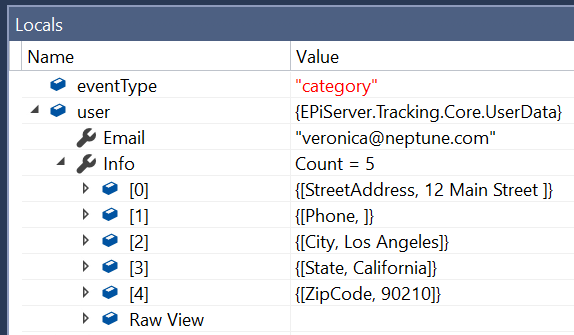
1. Start the website with the debugger attached.
2. When the home page tracks the event, note the breakpoint is hit, and the locals window shows the anonymous user, as shown in the following screenshot:



1. Click **Continue** or press *F5*.
2. Log in as an existing customer or register as a new customer.
3. When the home page tracks the event when you are redirected back to the home page, note the breakpoint is hit, and the locals window shows the details for the user, including their full name in the User.Name, their email, and StreetAddress, City, State, ZipCode, and Phone in User.Info, as shown in the following screenshot:



1. Click **Continue** or press *F5*.
2. Navigate to the customer’s **Address book**, as shown in the screenshot:
3. Click **New**.
4. Enter another address and save it as a **Preferred shipping address**.
5. Navigate to **Womens**.
6. When the category page tracks the event, note the breakpoint is hit, and the locals window shows the first address is still being used, as shown in the following screenshot:



### Implementing a custom user data service

You will now implement a custom user data service to control which address is used during tracking.

1. Create a new class named **PreferredShippingUserDataService**.
2. Modify it to implement and register a user data service that uses the customer’s preferred shipping address, as shown in the following code:

using EPiServer.ServiceLocation;

using EPiServer.Tracking.Commerce;

using Mediachase.Commerce.Customers;

using System.Collections.Generic;

using System.Web;

namespace EPiServer.Reference.Commerce.Site

{

[ServiceConfiguration(ServiceType = typeof(IUserDataService),

Lifecycle = ServiceInstanceScope.Singleton)]

public class PreferredShippingUserDataService : IUserDataService

{

public IDictionary<string, string> GetAdditionalInformation()

{

var address = CustomerContext.Current?.

CurrentContact?.PreferredShippingAddress;

if (address == null) return null;

var info = new Dictionary<string, string>

{

{ "StreetAddress", address.Line1 },

{ "City", address.City },

{ "State", address.State },

{ "ZipCode", address.PostalCode },

{ "Phone", address.DaytimePhoneNumber }

};

return info;

}

public string GetUserEmail(HttpContextBase httpContext)

{

return CustomerContext.Current?.CurrentContact?.Email;

}

public string GetUserName(HttpContextBase httpContext)

{

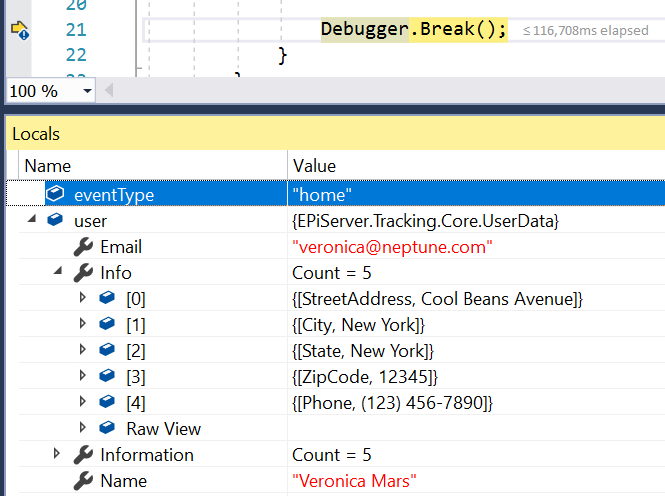
return CustomerContext.Current?.CurrentContact?.FullName;

}

}

}

1. Start the website with the debugger attached.
2. Click **Continue** or press *F5* when the anonymous customer views the home page.
3. Sign in as the customer you used before, and when you are redirected to the home page and the breakpoint is hit, note the preferred shipping address is now being used for tracking, as shown in the following screenshot:



1. Click **Continue** or press *F5*.
2. Close the browser.

### Disable the debugger tracking data interceptor

1. Open DebuggerTrackingDataInterceptor.cs, and modify its code to comment out the call to the Break method, as shown in the following code:

//Debugger.Break();

1. Save and close the class.

## Exercise 5 – Customizing the product feed

In this exercise, you will display catalog feed configuration on the home page, and then implement and register a custom filter for catalog items to exclude them from the exported product feed.

**Prerequisites:** complete Exercise C1.

### Configuring the catalog feed

1. Start Visual Studio.
2. Open the **Quicksilver** solution.
3. Expand the EPiServer.Reference.Commerce.Site project.
4. Open the Features\Start\Controllers\StartController.cs file.
5. Import the EPiServer.Personalization.Commerce.CatalogFeed namespace.
6. Add statements to declare fields to store personalization configuration service, widgets service, and tracking data interceptors, as shown in the following code:

private readonly CatalogFeedSettings catalogFeedSettings;

private readonly ICatalogItemFilter catalogItemFilter;

1. In the constructor, add statements to pass the registered catalog feed settings and catalog item filter as parameters and to set the fields to those parameters.
2. In the Index method, add statements to get and store the catalog feed settings and catalog item filter in the ViewData dictionary with keys named **feedSettings** and **feedFilter**, as shown in the following code:

ViewData["feedSettings"] = catalogFeedSettings;

ViewData["feedFilter"] = catalogItemFilter.GetType().FullName;

1. Open the Views\Start\Index.cshtml file.

* To save you time, you can copy and paste from a solution for this file.

1. Import the EPiServer.Personalization.Commerce.CatalogFeed namespace.
2. Add statements to the existing collapsible panel a table of information about the catalog feed settings including the catalog feed filter, as shown in the following markup:

@if (ViewData["feedSettings"] != null)

{

<div class="container-fluid">

<div class="row">

<div class="col-xs-12">

<h4>Feed Settings</h4>

<table class="table table-bordered table-condensed table-responsive">

<tr>

<th>Name</th>

<th>Value</th>

</tr>

<tr>

<td>AssetGroupName</td>

<td>@((ViewData["feedSettings"] as CatalogFeedSettings).AssetGroupName)</td>

</tr>

<tr>

<td>CatalogContentBatchSize</td>

<td>@((ViewData["feedSettings"] as CatalogFeedSettings).CatalogContentBatchSize)</td>

</tr>

<tr>

<td>DescriptionPropertyName</td>

<td>@((ViewData["feedSettings"] as CatalogFeedSettings).DescriptionPropertyName)</td>

</tr>

<tr>

<td>ExcludedAttributes</td>

<td>@(string.Join(", ", (ViewData["feedSettings"] as CatalogFeedSettings).ExcludedAttributes))</td>

</tr>

<tr>

<td>Feed Filter</td>

<td>@ViewData["feedFilter"]</td>

</tr>

</table>

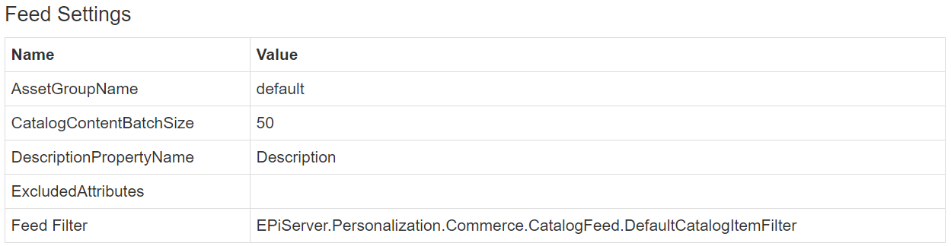
</div>

</div>

</div>

}

1. Start the website.
2. Click **Diagnostics**, and note the information about catalog feed configuration, including your custom feed filter as shown in the following screenshot:



* DefaultCatalogFeedFilter’s ShouldFilter method returns true for unpublished content.

1. Close the browser.

### Implementing a catalog item filter

1. Add a new class named **QuicksilverCatalogItemFilter**.
2. Modify the class to implement the ICatalogItemFilter interface, and use the default implementation to return true for the ShouldFilter method if the content has not been published or its display name contains a bad word, as shown in the following code:

using EPiServer.Commerce.Catalog.ContentTypes;

using EPiServer.Personalization.Commerce.CatalogFeed;

namespace EPiServer.Reference.Commerce.Site

{

public class QuicksilverCatalogItemFilter : ICatalogItemFilter

{

private readonly ICatalogItemFilter defaultImplementation;

public QuicksilverCatalogItemFilter(

ICatalogItemFilter defaultImplementation)

{

this.defaultImplementation = defaultImplementation;

}

public bool ShouldFilter(CatalogContentBase content, string scope)

{

string displayName = content.Property["DisplayName"]?.Value.ToString();

bool containsBadWord = displayName.ToUpper().Contains("BAD WORD");

bool defaultShouldFilter =

defaultImplementation.ShouldFilter(content, scope);

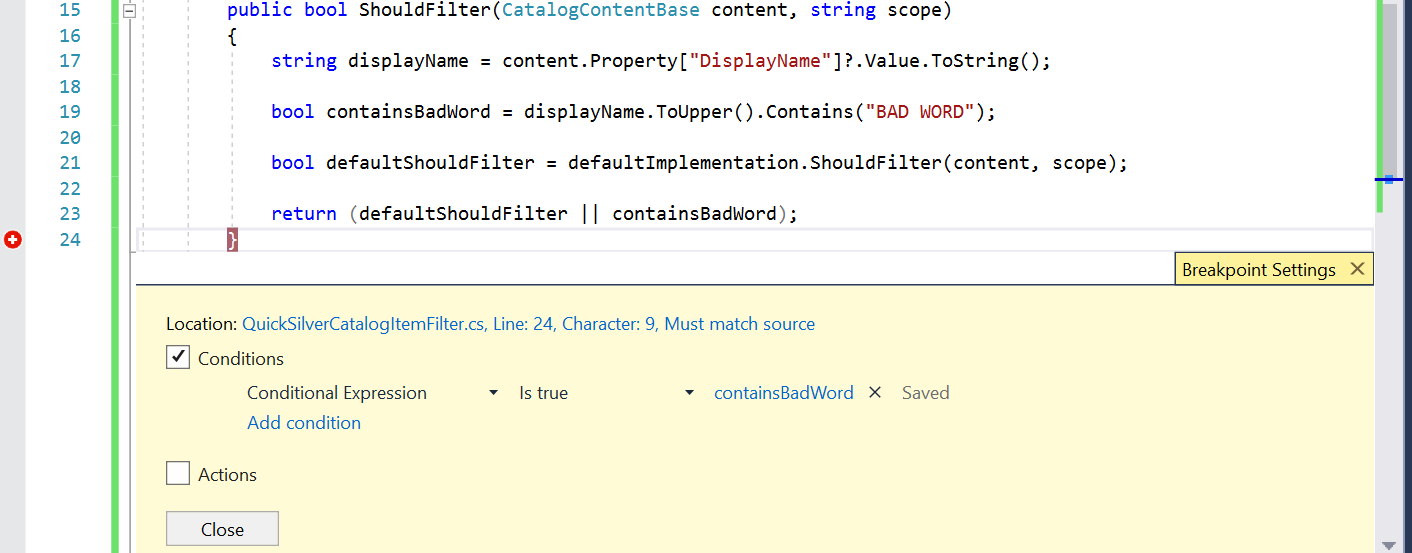
return (defaultShouldFilter || containsBadWord);

}

}

}

1. Set a conditional breakpoint on the close brace that only activates if containsBadWord is **true**, as shown in the following screenshot:



* Only one catalog item filter can be registered, so we must intercept the default one.

1. Open Infrastructure\SiteInitialization.cs.
2. Add a statement to the ConfigureContainer method to intercept the default catalog item filter and replace with your implementation, as shown in the following code:

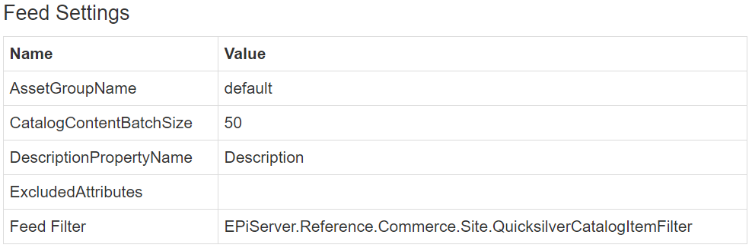
services.Intercept<ICatalogItemFilter>(

(locator, defaultImplementation) =>

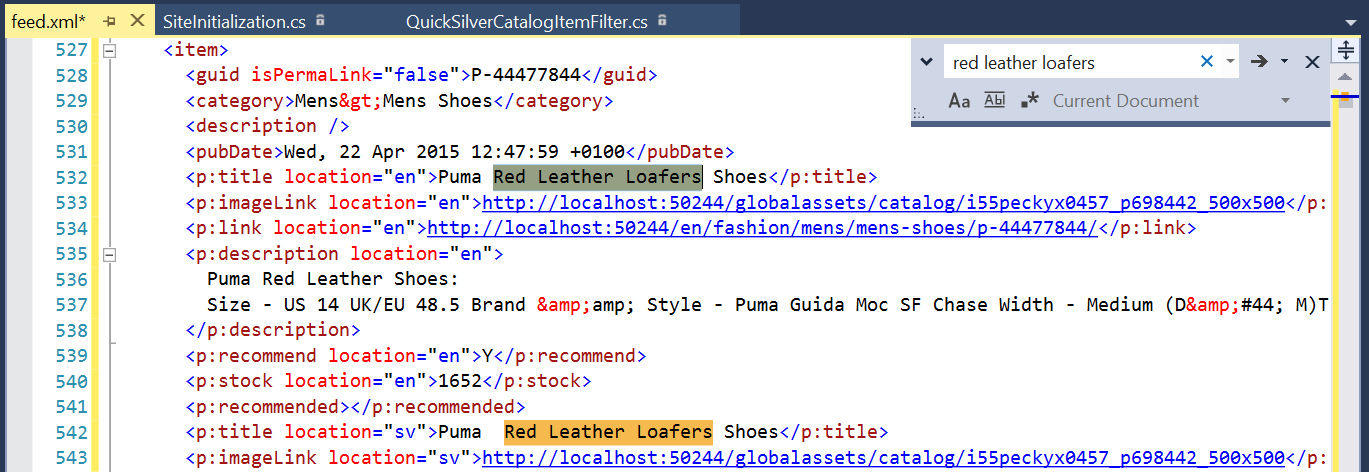
new QuicksilverCatalogItemFilter(defaultImplementation));

### Testing the catalog item filter

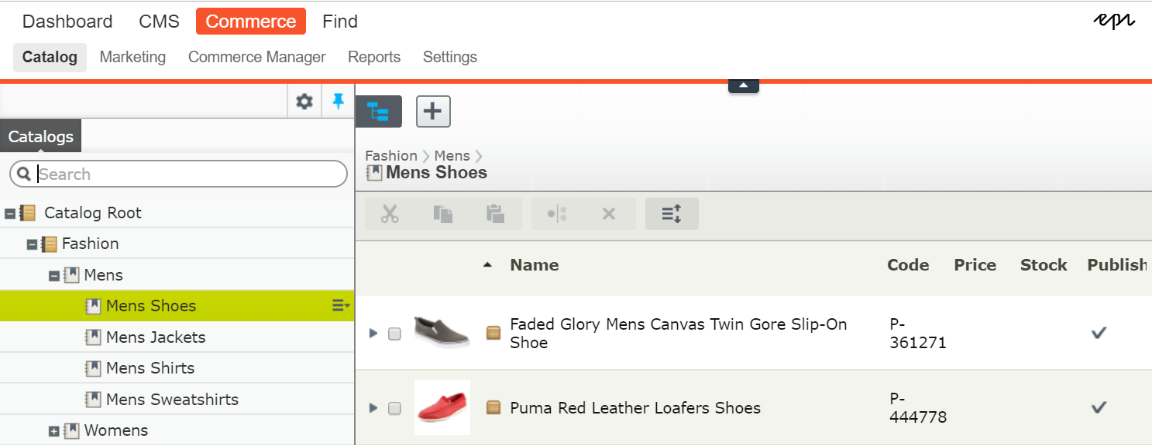
1. Start the website with debugging.
2. Click **Diagnostics**, and note the information about catalog feed configuration, including your custom feed filter as shown in the following screenshot:



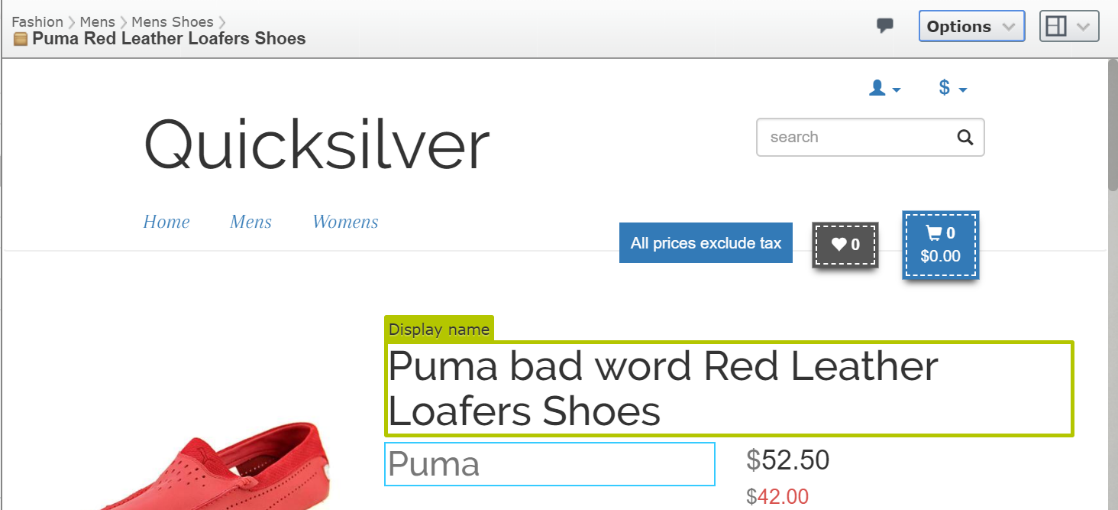
1. Sign in as **admin@example.com**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Export Product Feed**.
3. Start the job manually.
4. Click **History** and make sure the job completed successfully.
5. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Peform Feed Status**.
6. Click **Download** for the last entry in the table.
7. Extract the ZIP file.
8. Open the feed.xml file in Visual Studio.
9. Use **Quick Find** to search for **red leather loafers** and note that it is found, and it has a GUID of **P-44477844**, as shown in the following screenshot:



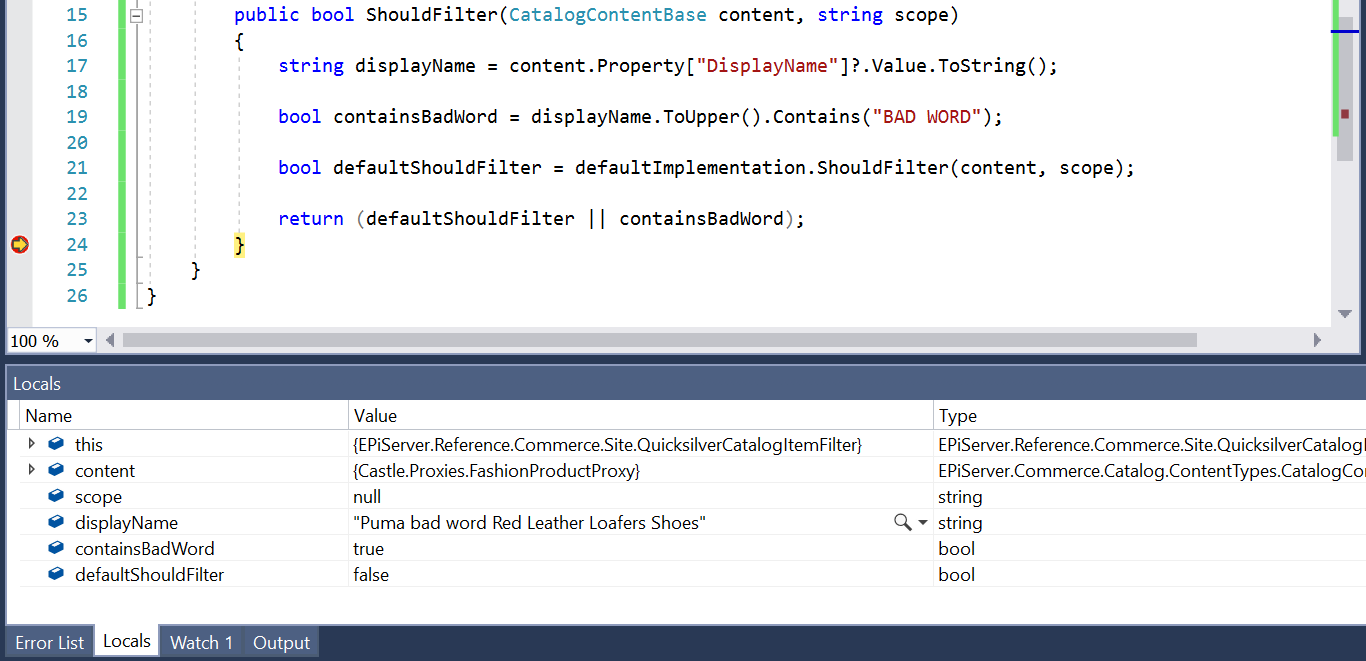
1. Close the feed.xml file.
2. In the browser, in the Quicksilver site, navigate to **Commerce** | **Catalog** | **Fashion** | **Mens Shoes**, as shown in the following screenshot:

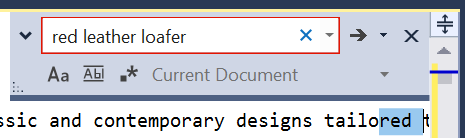


1. Click **Puma Red Leather Shoes**, modify the **Display name** to include bad word, and **publish** the change, as shown in the following screenshot:



1. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Export Product Feed**.
2. Start the job manually.
3. When the breakpoint is hit, note the value of the displayName, containsBadWord, and defaultShouldFilter, as shown in the following screenshot:



1. Click **Continue** or press *F5*.
2. Click **History** and make sure the job completed successfully.
3. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Peform Feed Status**.
4. Click **Download** for the last entry in the table.
5. Extract the ZIP file.
6. Open the feed.xml file in Visual Studio.
7. Use **Quick Find** to search for **red leather loafers**, and note that it is not found, as shown in the screenshot:
8. Close the browser.
9. Disable the breakpoint.