Episerver CMS

Advanced Development Exercise Book

February 2018

Product version: Update 202
Course version: 18.02

Course title: *Episerver CMS – Advanced Development Fundamentals* Course code: 170-3030
Course version: 18.02, 25th February 2018 Product Update 202, 20th February 2018

Episerver CMS Visual Studio Extension version: 11.1.0.326
Episerver CMS packages: EPiServer.CMS.Core 11.3.3, EPiServer.CMS.UI 11.2.6

http://world.episerver.com/releases/

Episerver - update 202

Included packages: Logging 2.2.2, CMS UI 11.2.6

Feb 20 2018

Bug fixes for Episerver Logging (including a critical bug fix) and Episerver CMS UI.

Episerver - update 201

Included packages: CMS UI 11.2.5, Azure 9.4.2, Commerce 11.8.1, Insight 1.3.2, Profile Store 1.3.2, Campaign 6.73, Forms 4.10.0, Mail 10.0.0, UGC 1.1.0

Feb 12 2018

New releases of Episerver Campaign, and the add-ons Episerver User Generated Content (UGC) and Episerver Mail (both with support for Episerver CMS 11). Bug fixes for Episerver CMS UI, Episerver Azure, Episerver Commerce, Episerver Insight, Episerver Profile Store, Episerver Forms, and the third-party add-on Lionbridge connector for Episerver.

Episerver - update 200

Included packages: CMS Core 11.3.3, Logging4Net 2.2.1, CMS UI 11.2.4, Commerce 11.8.0, Find for Commerce 10.1.1, Salesforce 3.3.0

Feb 05 2018

New releases of Episerver Commerce and the Marketing Automation Salesforce connector. Bug fixes for Episerver CMS Core, Episerver CMS UI, Episerver Find for Commerce, and Episerver Logging4Net.

Table of Contents

Episerver CMS – Advanced Development

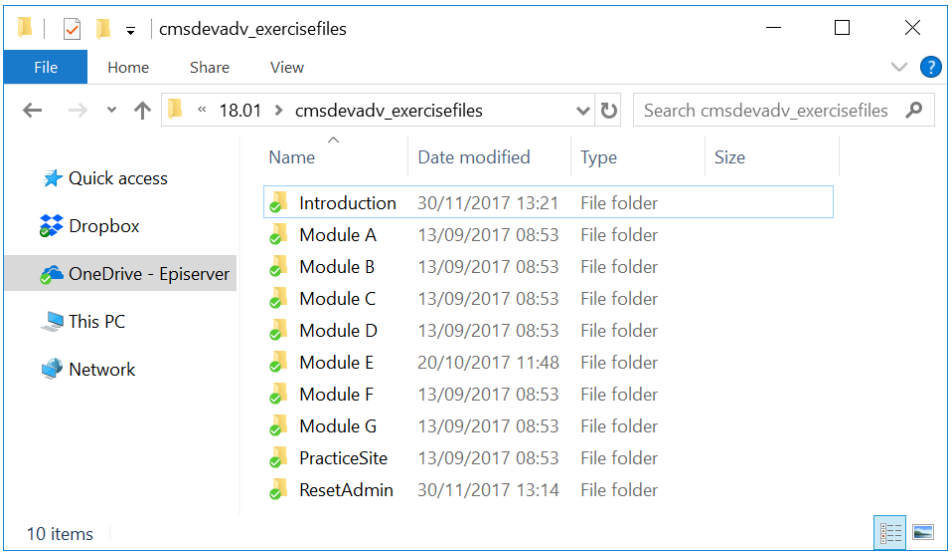
Student prerequisites

You should have experience equivalent to attending the *Episerver CMS Development Fundamentals* course.

Software requirements

To complete these exercises, you will need:

- **Microsoft Visual Studio 2017 version 15.3** or later with latest updates.
- **Episerver CMS Visual Studio Extension** version 11.1.0.326 or later.
- **cmsdevadv_exercisefiles.zip** containing solution files for the exercises.



Introduction

Exercise 0.1 – Setting up the AlloyAdvanced web site

In this exercise, you will set up an Alloy (MVC) site ready to extend during the rest of the exercises.

If you are using an Episerver virtual machine, or you have already installed Visual Studio, Episerver CMS Visual Studio Extension, and set up the Episerver NuGet package source, then you can skip ahead to page 6, *Creating an Alloy (MVC) Episerver website with sample content*.

Minimum development system requirements

http://world.episerver.com/documentation/Items/System-Requirements/System-Requirements---EPiServer/

- Microsoft Windows 8, or later, or Windows Server 2012, or later.
- Microsoft Visual Studio 2015, or later.

These instructions will use our most recent recommended development stack: Microsoft Windows 10, Microsoft Visual Studio 2017 including SQL Server LocalDb, and Episerver CMS Visual Studio Extension 11.1.0.326. Older versions may look and behave slightly differently.

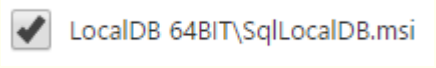
Installing Microsoft Visual Studio Community 2017

If you have already installed **Microsoft Visual Studio 2015** or **Microsoft Visual Studio 2017**, or you are using a virtual machine provided by Episerver, then you can skip this section.

1. Download and install **Microsoft Visual Studio Community 2017** from the following link:

   https://www.visualstudio.com/downloads/
2. At a minimum, choose the workloads: **ASP.NET and web development**, **Azure development**, and **.NET Core cross-platform development**.
3. Add the individual components: **Azure Storage AzCopy**, **Class Designer**, **Git for Windows**, **GitHub extension for Visual Studio**, and **PowerShell tools**.

Microsoft SQL Server Express LocalDb should be included with Visual Studio, but you can also install it separately from the following link: https://www.microsoft.com/en-gb/download/details.aspx?id=42299
Click **Download**, check the box for **LocalDB 64BIT\SqlLocalDB.msi**, and click **Next**.

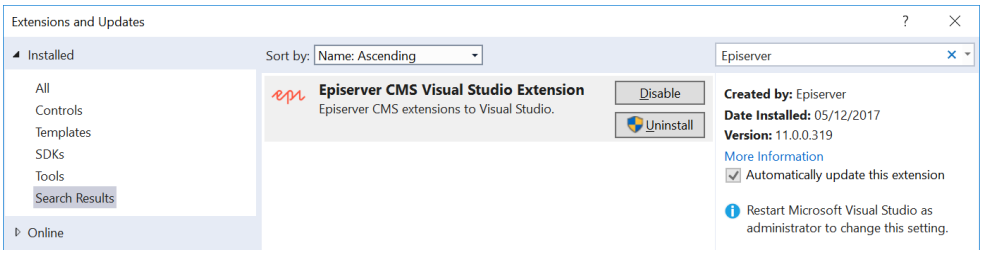


Installing the Episerver CMS Visual Studio Extension

If you have already installed the **Episerver CMS Visual Studio Extension**, or you are using a virtual machine provided by Episerver, then you can skip this section.

1. Start Microsoft Visual Studio.
2. Navigate to **Tools** | **Extensions and Updates...**, and in the section on the left, click **Online**, to show the **Visual Studio Marketplace**.
3. Press *Ctrl* + *E*, or click in the **Search** box, and then enter **Episerver**.

4. Select the **Episerver CMS Visual Studio Extension**, click **Download**, as shown in the following screenshot:



Episerver CMS Visual Studio Extension 11.1.0.326 was released on 18th December 2017.
To install an older version, download from the following link:
https://marketplace.visualstudio.com/items?itemName=EPiServer.EpiserverCMSVisualStudioExtension

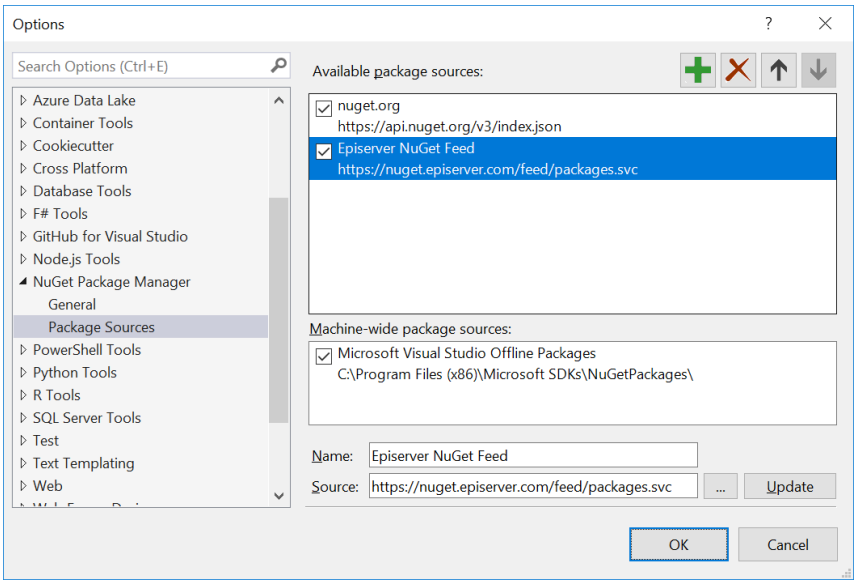
5. Wait for the extension to download, and then click **Close**.
6. Close Visual Studio and wait for the **VSIX Installer** to start.
7. Click **Modify**, wait for the installer to finish, and then click **Close**.

Configuring the Episerver NuGets package source

If you have already configured the **Episerver NuGets** package source, or you are using a virtual machine provided by Episerver, then you can skip this section.

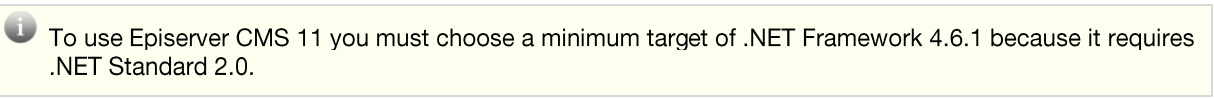
1. Start Microsoft Visual Studio.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Settings**.
3. In the **Options** dialog, in the list on the left, click **Package Sources**.
4. If the Episerver NuGet feed doesn't exist as an available package source, as shown in the following 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/`



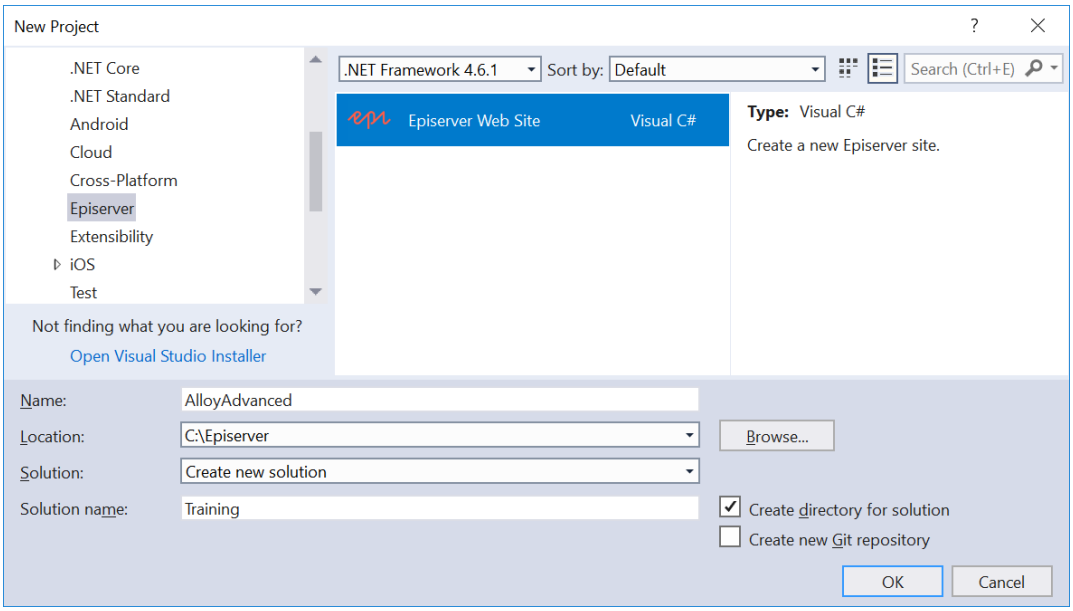
Creating an Alloy (MVC) Episerver website with sample content

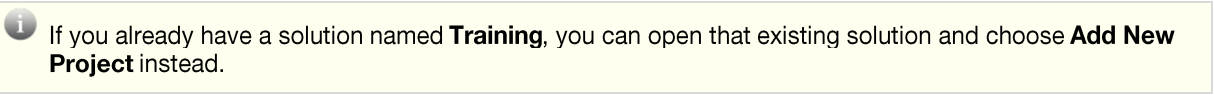
1. In Visual Studio, navigate to **File** | **New** | **Project...**, or press *Ctrl* + *Shift* + *N*.
2. In the left section, navigate to **Installed** | **Visual C#** | **Episerver**.
3. In the middle section, select **Episerver Web Site** project, and enter the following options, as shown in the following screenshot:
    - Target: **.NET Framework 4.6.1** (or a later compatible version).



To use Episerver CMS 11 you must choose a minimum target of .NET Framework 4.6.1 because it requires .NET Standard 2.0.

    - Name: **AlloyAdvanced**
    - Location: **C:\Episerver** (or some other folder).
    - Solution name: **Training** (or something else unique).
    - Create directory for solution: **Yes**

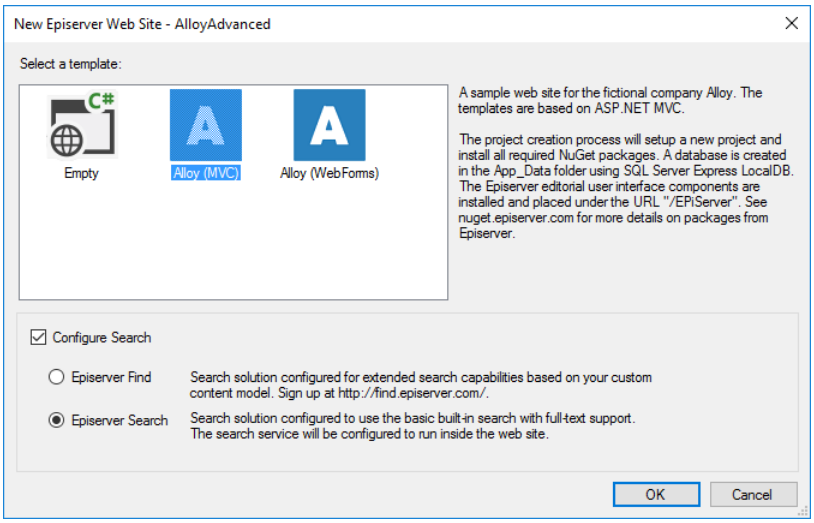




If you already have a solution named **Training**, you can open that existing solution and choose **Add New Project** instead.

4. Click **OK**.

5. Choose the **Alloy (MVC)** template, and in the **Configure Search** section, select **Episerver Search**, as shown in the following screenshot:



6. Click **OK** and wait for the project to be created.

Removing duplicate entries in <runtime>

There is a bug in Episerver CMS Visual Studio Extension 11.1.0.326 that adds duplicate enties to the <runtime> <assemblyBinding> section of Web.config. You must fix this before updating packages.

1. Open **Web.config**.
2. Find the `<runtime>` section.
3. Find the duplicate entry for `<assemblyIdentity name="EPiServer.Data.Cache"` and delete it.
4. Find the duplicate entry for `<assemblyIdentity name="EPiServer.ServiceLocation.StructureMap"` and delete it.

Updating the Episerver NuGet packages and database schema

Episerver CMS Visual Studio Extension 11.1.0.326 uses NuGet packages from 18th December 2017, i.e. EPiServer.CMS.Core 11.3.0. Recommended practice is to update to the latest NuGet packages for your chosen major version number that are usually released on a weekly schedule.

1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. In **Package Manager Console**, set these two options:
   - Package source: **All**
   - Default project: **AlloyAdvanced**

If you cannot see **All** as a **Package source**, then close Visual Studio, use **File Explorer** to open **%appdata%\NuGet**, rename **NuGet.config** to **NuGet.backup**, and restart Visual Studio. That should recreate **NuGet.config**, including the **All** option. You can then copy and paste any additional configuration from the backup, if necessary.

3. Enter one of the following commands to update all packages referenced by the project **AlloyAdvanced**, using restrictions defined in packages.config:

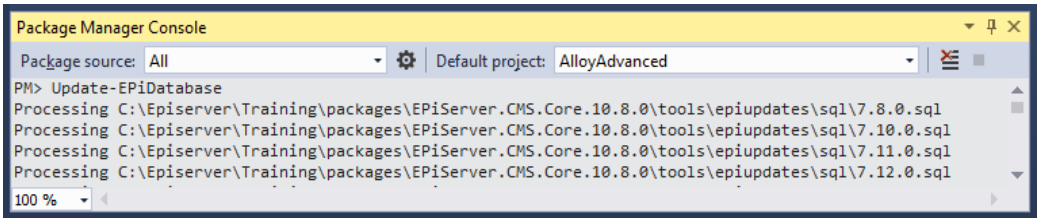
```
Update-Package -ProjectName AlloyAdvanced -ToHighestMinor
```

Doing updates at the command line allows Episerver packages to be safely installed because it won't upgrade to a higher major version that would have breaking changes, and non-Episerver dependencies will

be updated, as well as the EPiServer ones, for example, **Microsoft.Tpl.DataFlow**, but won't accidently install newer but incompatible packages. If you don't have the latest version of NuGet, you can follow instructions here: https://docs.microsoft.com/en-us/nuget/guides/install-nuget

4. Make sure the **Default project** is **AlloyAdvanced**, as shown in the following screenshot, and at the prompt enter:

```
Update-EPiDatabase
```

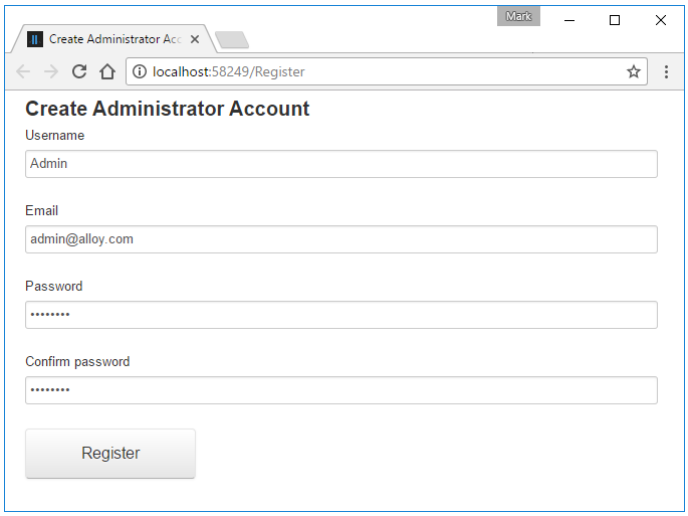


If you get a warning about a missing **packages** folder, exit and restart Visual Studio, and try again.

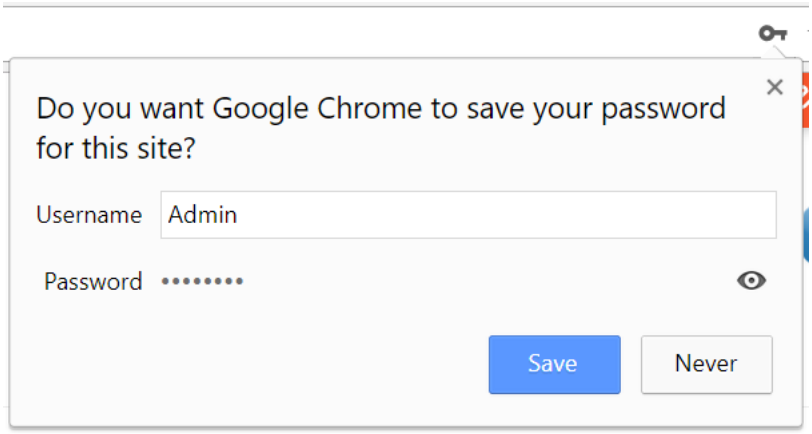
Testing the Episerver website and registering an administrator account

1. Start the site by navigating to **Debug | Start Without Debugging**, or press *Ctrl + F5*.
2. You will be prompted to **Create Administrator Account**, as shown in the following screenshot, so enter the following details:

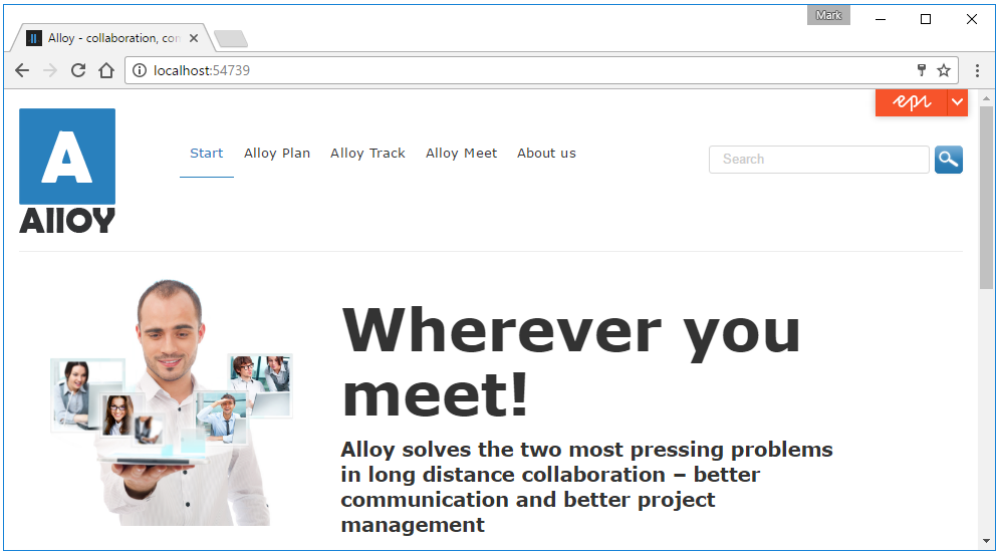
- Username: **Admin** or something else, but make a note of it!
- Email: **admin@alloy.com** or some other e-mail address (it does not need to be real).
- Password: **Pa$$w0rd** or something else, but make a note of it!



3. If you use Chrome, it will prompt to save your **Username** and **Password**. Click **Save**, as shown in the following screenshot:



4. You should now see the **Alloy** sample site's **Start** page, as shown in the following screenshot:



5. Close the browser.

Congratulations! You are now ready to complete the exercises in this course.

Exercise 0.2 – Setting up the Northwind database

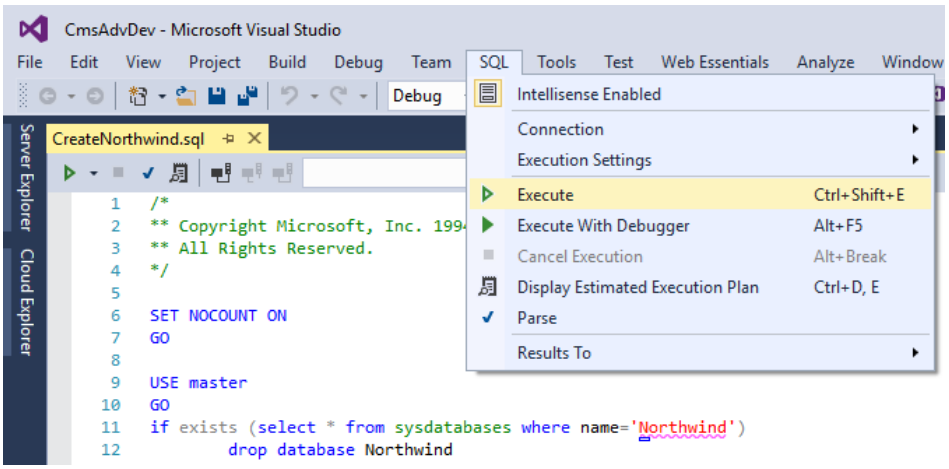
In this exercise, you will set up an SQL Server database for use during some of the other exercises.

Northwind is a sample database originally used by Microsoft in the 1990s.

Prerequisites: complete Exercise 0.1.

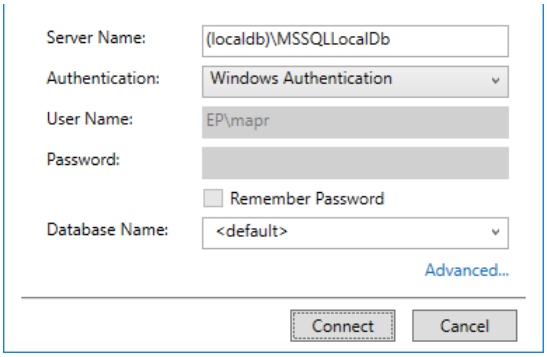
Creating the Northwind database

1. Extract the folders and files in **cmsdevadv_exercisefiles.zip**.
2. In **Visual Studio**, choose **File** | **Open** | **File...**, or press *Ctrl* + *O*, change to the **\cmsdevadv_exercisefiles\Introduction** folder, and open the file named **CreateNorthwind.sql**.
3. To run the SQL script to create the Northwind database, choose **SQL** | **Execute**, or press *Ctrl* + *Shift* + *E*, as shown in the following screenshot:



If you would like to use SQL Server Express or the full version of SQL Server, then enter the appropriate server name instead of **(localdb)\MSSQLLocalDb** in the next step.

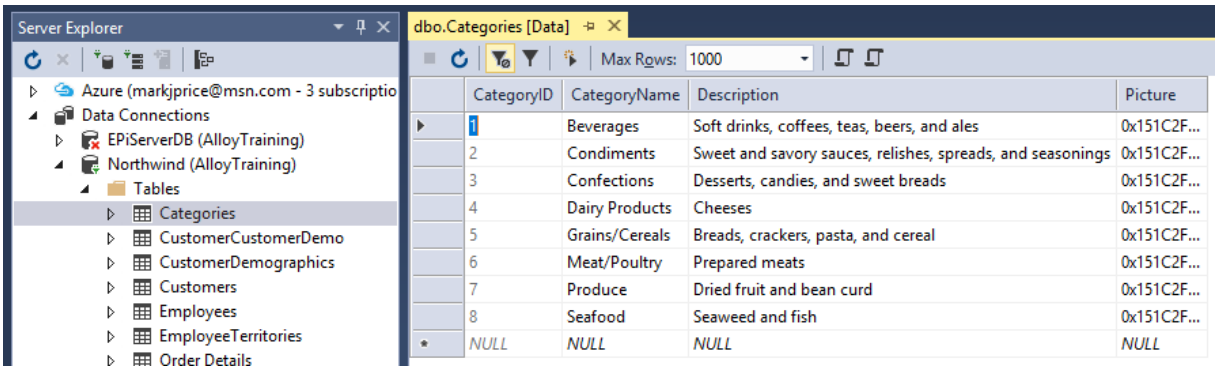
4. Enter the server name: **(localdb)\MSSQLLocalDb**, and click **Connect**, as shown in the following screenshot:



5. After a few seconds, you will see the message:

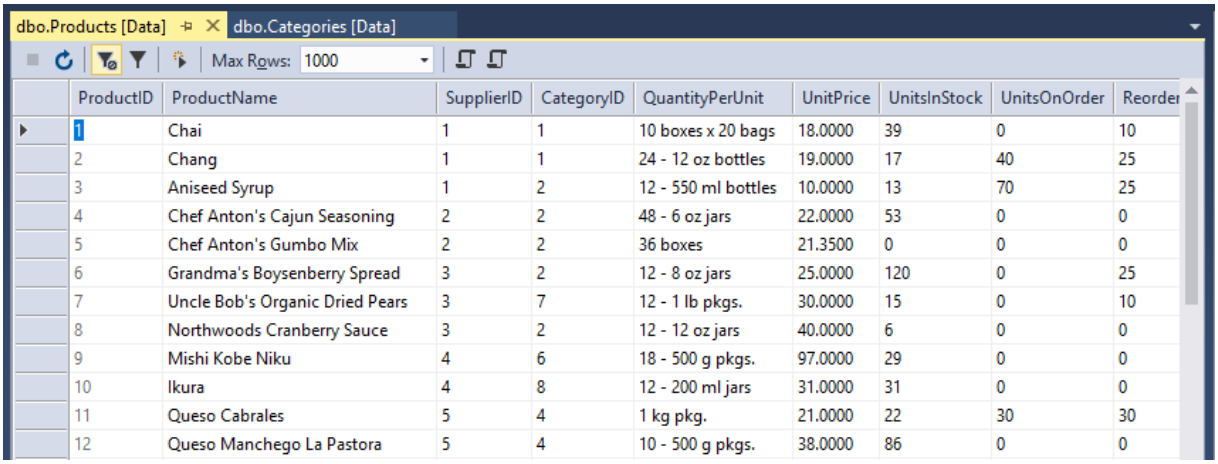
```
Command(s) completed successfully.
```

6. Navigate to **View** | **Server Explorer**.
7. In **Server Explorer**, expand **Data Connections**.
8. If the **Northwind** database is not shown, right-click **Data Connections**, and click **Add Connection...**, in **Choose Data Source**, click **Microsoft SQL Server**, and click **Continue**. In **Add Connection**, enter server name **(localdb)\MSSQLLocalDb**, select or enter the database name **Northwind**, and click **OK**.
9. In **Server Explorer**, under **Data Connections**, expand **Northwind**.
10. Expand **Tables**.
11. Right-click **Categories**, and click **Show Table Data**, and note there are eight rows in the **Categories** table, as shown in the following screenshot:



CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x151C2F...
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	0x151C2F...
3	Confections	Desserts, candies, and sweet breads	0x151C2F...
4	Dairy Products	Cheeses	0x151C2F...
5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x151C2F...
6	Meat/Poultry	Prepared meats	0x151C2F...
7	Produce	Dried fruit and bean curd	0x151C2F...
8	Seafood	Seaweed and fish	0x151C2F...
NULL	NULL	NULL	NULL

12. Right-click **Products**, and click **Show Table Data**, and note there are 77 rows, and each product belongs to a category, as shown in the following screenshot:

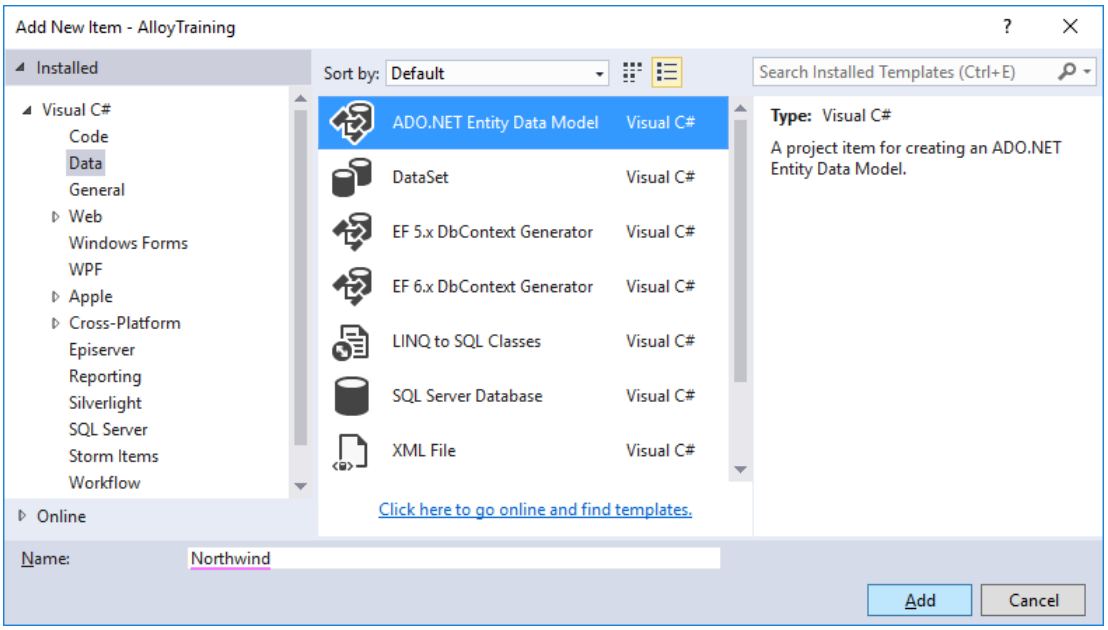


ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	Reorder
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0	10
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40	25
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70	25
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0	0
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.0000	15	0	10
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.0000	6	0	0
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97.0000	29	0	0
10	Ikura	4	8	12 - 200 ml jars	31.0000	31	0	0
11	Queso Cabrales	5	4	1 kg pkg.	21.0000	22	30	30
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38.0000	86	0	0

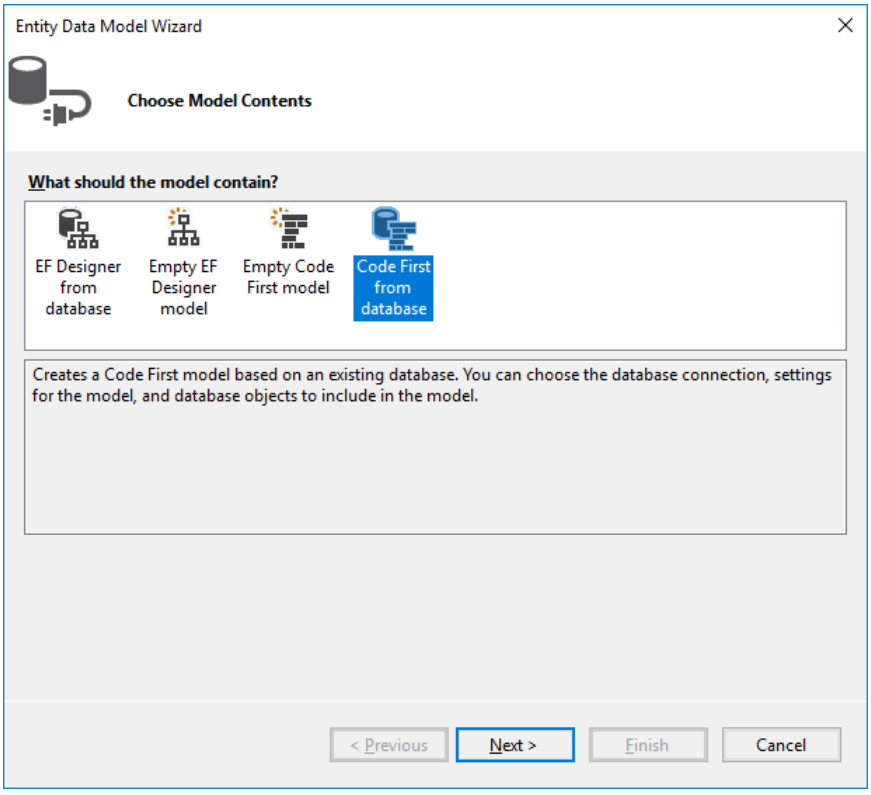
Creating a domain model for the external database

1. Open your solution with the **AlloyAdvanced** project.
2. In **Solution Explorer**, in **~\Models**, add a folder named **NorthwindEntities**.

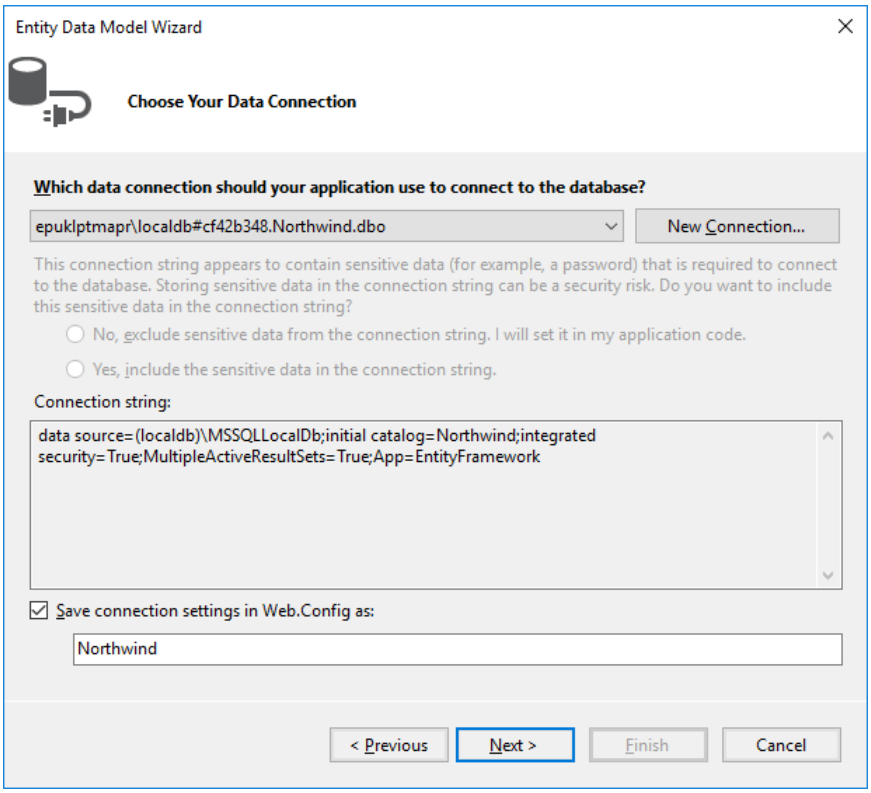
3. In **~\Models\NorthwindEntities**, add an **ADO.NET Entity Data Model** named **Northwind**, as shown in the following screenshot:



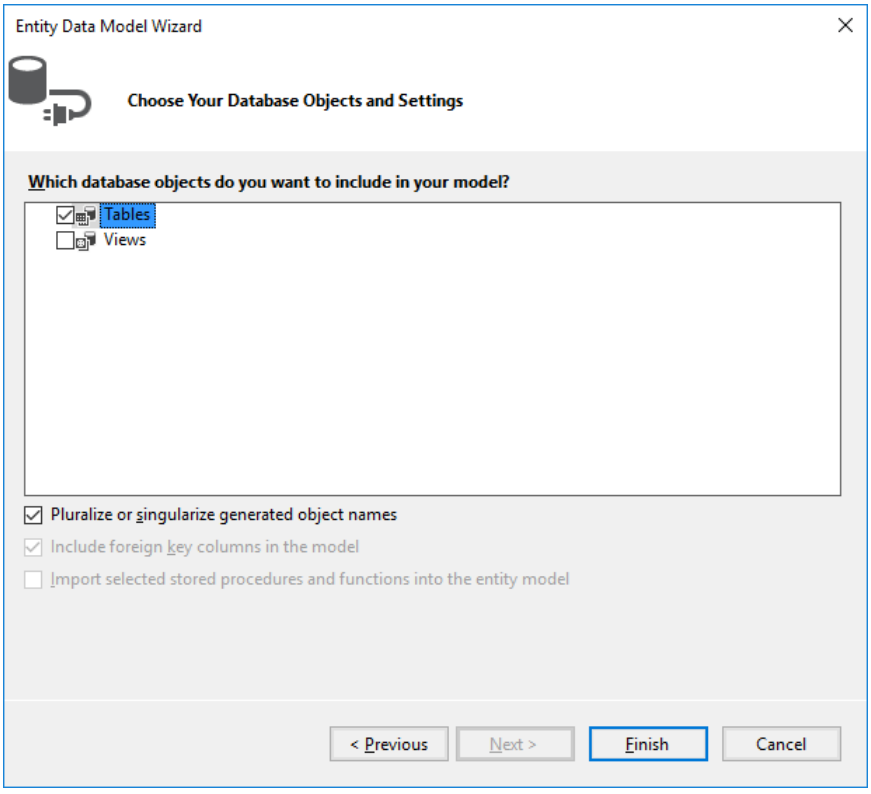
4. Choose **Code First from database**, and then click **Next**, as shown in the following screenshot:



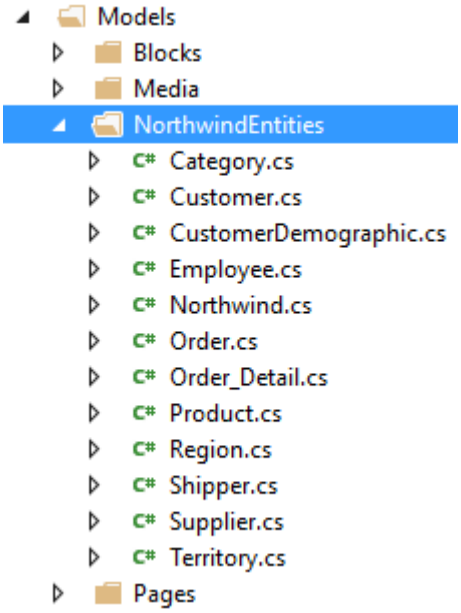
5. Choose an existing connection, or create a new connection to the **Northwind** database, and save the connection string in Web.config, as shown in the following screenshot:



6. Select all tables, pluralize or singularize generated object names, and click **Finish**, as shown in the following screenshot:

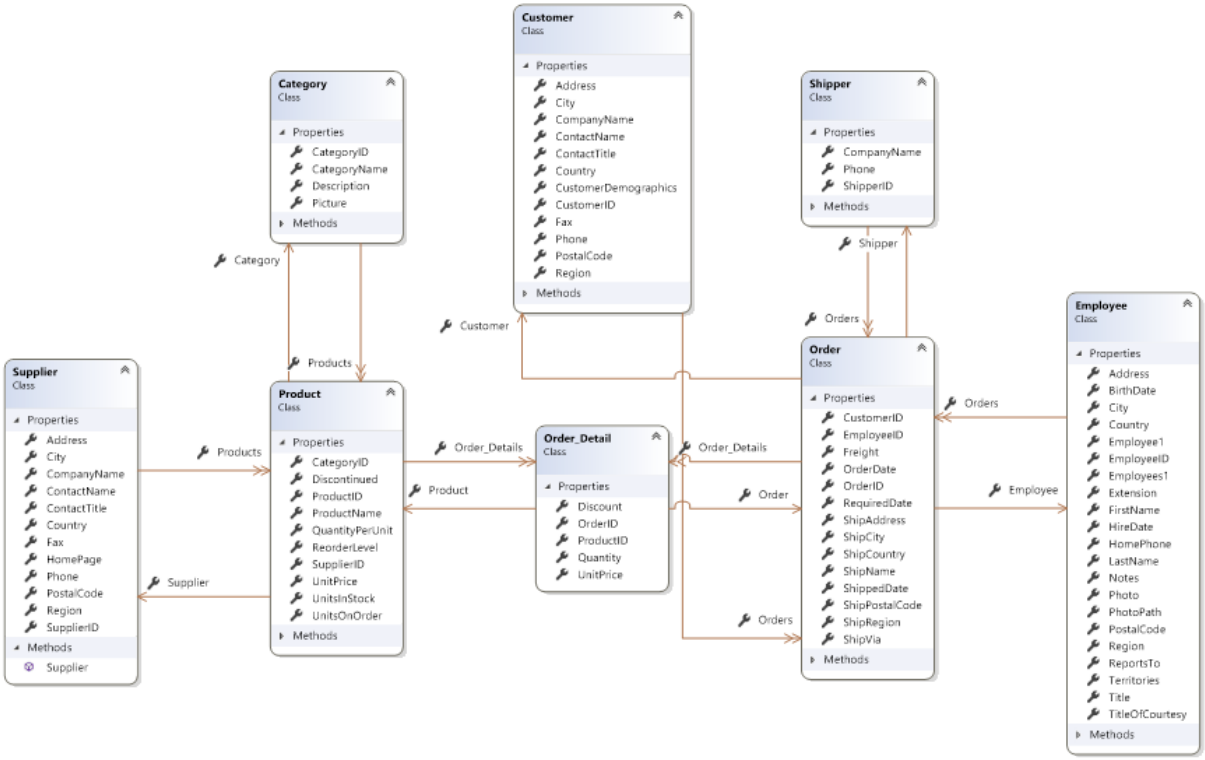


7. Review the classes that the wizard created for you, as shown in the following screenshot:



- **Northwind.cs**: manages the connection to the database and tracks changes to entities.
- **Category.cs**, **Customer.cs**, **Product.cs**, **and so on**: class entities that represent a row in the equivalent table.

The following screenshot shows a class diagram of the associations (relationships) between entities:



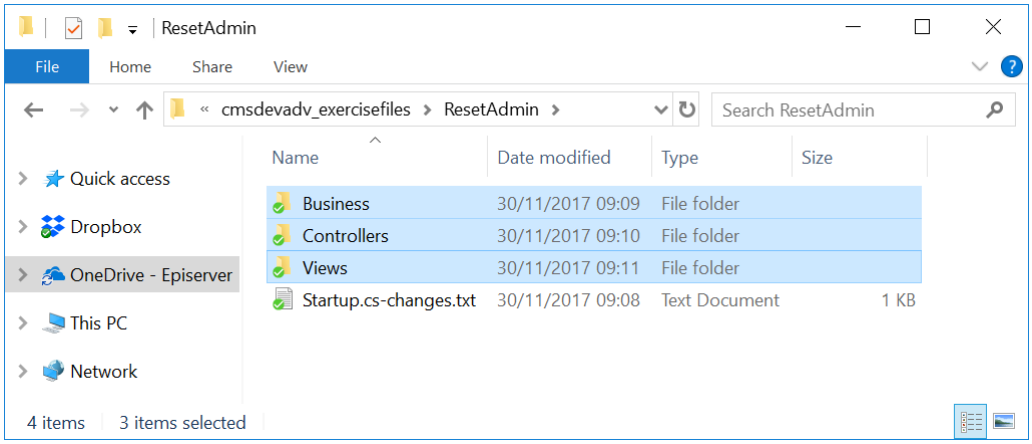
Exercise 0.3 – Resetting the Admin account (optional)

In this exercise, you will add functionality to reset the Admin account (if necessary).

Prerequisites: complete Exercise 0.1.

Adding reset admin functionality

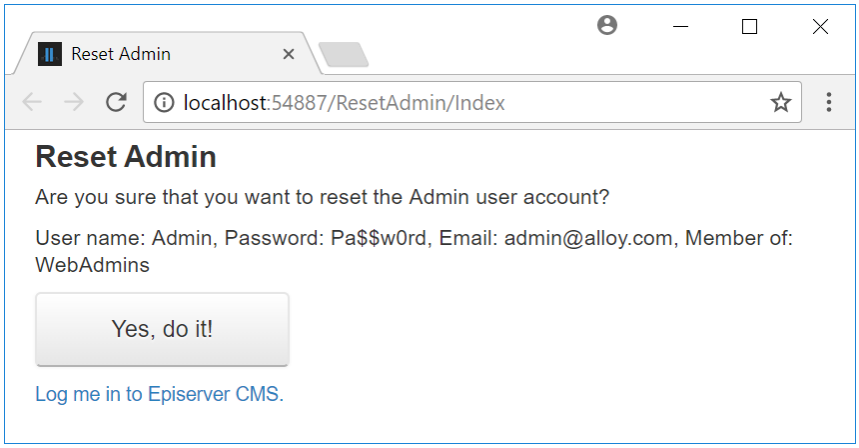
1. Drag and drop the three folders from **ResetAdmin** to **AlloyAdvanced** project, as shown in the following screenshot:



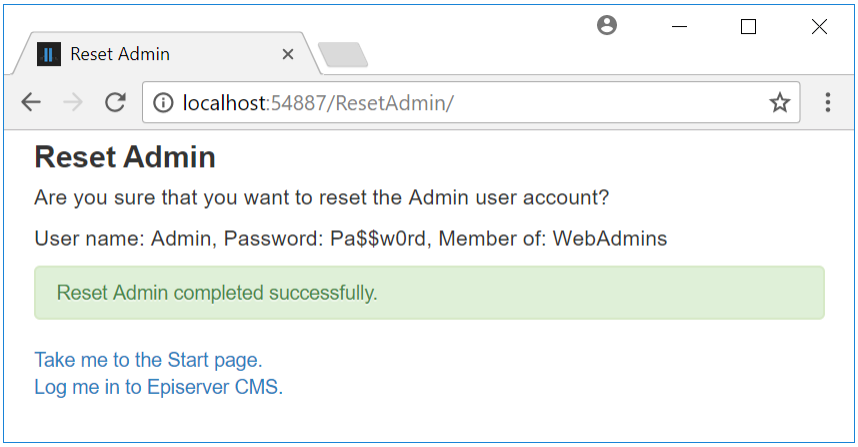
2. Open the **Startup.cs-changes.txt** file, and copy and paste the statements into the appropriate place in the **Startup.cs** file.
3. Open **~/Controllers/ResetAdminController.cs** file and modify the username and password if you want.

Resetting the admin account

1. Start the **AlloyAdvanced** website.
2. On the **Reset Admin** page, click **Yes, do it!**, as shown in the following screenshot:



3. On the **Reset Admin** page, click **Log me in to Episerver CMS**, as shown in the following screenshot:



4. Log in using the new username and password.
5. Close the browser.

Disabling admin reset

1. Open the **Startup.cs** file.
2. Comment out or delete the statement that calls **UseResetAdmin()**.
3. Save changes and close the file.

Module A – Reviewing Episerver CMS Fundamentals

Goal

The overall goal of the exercises in this module is to learn how to use sometimes-overlooked features of the Episerver CMS system. You will:

- Implement validation techniques.
- Explore Episerver developer tools.
- Explore authentication and authorization.
- Complete some challenges.

Exercise A1 – Validating content and properties

In this exercise, you will implement two common validation techniques: (1) a validation attribute, and (2) Episerver's `IValidate<T>` interface.

Prerequisites: complete Exercise 0.1.

Creating properties that must be validated

1. In **~\Models\Pages**, open **StartPage.cs**.
2. Add three properties, as shown in the following code:

```
public virtual string Password { get; set; }
public virtual DateTime StartDate { get; set; }
public virtual DateTime EndDate { get; set; }
```

Creating an attribute to validate passwords

First, we need to validate the single string property named Password to ensure that it is "strong". We can do that by creating an attribute.

1. In **~\Business**, add a new folder named **Validation**.
2. In **~\Business\Validation**, add a new class named **StrongPassword.cs**.
3. Write code for a custom validation attribute that enforces strong password rules, as shown in the following code:

```
using System.ComponentModel.DataAnnotations;
using System.Text;
using System.Text.RegularExpressions;

namespace AlloyAdvanced.Business.Validation
{
    // Example regular expression for strong password:
    // ^                                Start anchor
    // (?=.*[A-Z].*[A-Z])               Ensure string has two uppercase letters.
    // (?=.*[!@#$&*])                   Ensure string has one special case letter.
    // (?=.*[0-9].*[0-9])           Ensure string has two digits.
    // (?=.*[a-z].*[a-z].*[a-z])  Ensure string has three lowercase letters.
    // .{8,}                            Ensure string is of length 8 minimum.
    // $                                End anchor.

    public class StrongPassword : ValidationAttribute
    {
        public string SpecialCharacters { get; set; } = "!@#$&*";

        public int MinimumTotalCharacters { get; set; } = 8;
```

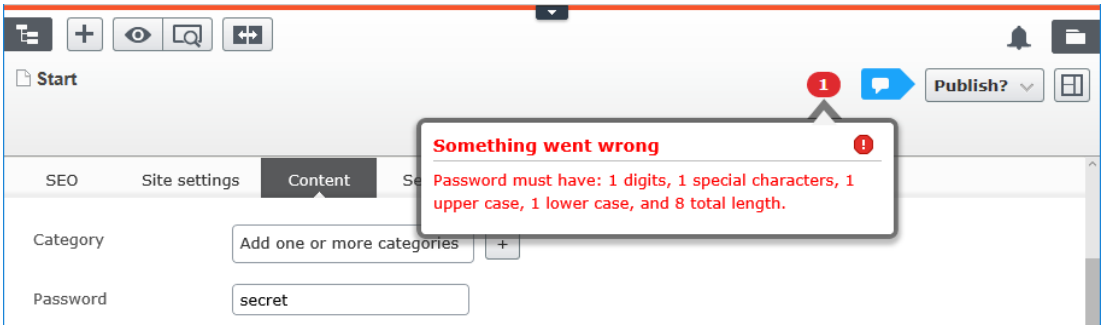
```
        public int MinimumUppercaseLetters { get; set; } = 1;
        public int MinimumLowercaseLetters { get; set; } = 1;
        public int MinimumDigitCharacters { get; set; } = 1;
        public int MinimumSpecialCharacters { get; set; } = 1;

        public override bool IsValid(object value)
        {
            ErrorMessage = $"Password must have: {MinimumDigitCharacters} digits,
{MinimumSpecialCharacters} special characters, {MinimumUppercaseLetters} upper case,
{MinimumLowercaseLetters} lower case, and {MinimumTotalCharacters} total length.";

            string input = value as string;

            var builder = new StringBuilder();

            builder.Append("^");

            if (MinimumUppercaseLetters > 0)
            {
                builder.Append("(?=");
                for (int i = 0; i < MinimumUppercaseLetters; i++)
                {
                    builder.Append(".*[A-Z]");
                }
                builder.Append(")");
            }

            if (MinimumLowercaseLetters > 0)
            {
                builder.Append("(?=");
                for (int i = 0; i < MinimumLowercaseLetters; i++)
                {
                    builder.Append(".*[a-z]");
                }
                builder.Append(")");
            }

            if (MinimumSpecialCharacters > 0)
            {
                builder.Append("(?=");
                for (int i = 0; i < MinimumSpecialCharacters; i++)
                {
                    builder.Append(".*[");
                    builder.Append(SpecialCharacters);
                    builder.Append("]");
                }
                builder.Append(")");
            }

            if (MinimumDigitCharacters > 0)
            {
                builder.Append("(?=");
                for (int i = 0; i < MinimumDigitCharacters; i++)
                {
                    builder.Append(".*[0-9]");
                }
                builder.Append(")");
            }

            builder.Append(".{");
            builder.Append(MinimumTotalCharacters);
```

```
            builder.Append(",}$");

            return Regex.IsMatch(input, builder.ToString());
        }
    }
}
```

4. Open **StartPage.cs**, import the namespace **AlloyAdvanced.Business.Validation**, and apply the **StrongPassword** attribute to the **Password** property, as shown in the following code:

```
[StrongPassword]
public virtual string Password { get; set; }
```

5. Start the site and log in as **Admin**.
6. Edit the **Start** page and set **Password** to a weak value, like **secret**, as shown in the following screenshot:



7. Set **Password** to a strong value, as shown in the following screenshot:

8. Close the browser.
9. In **StartPage.cs**, modify the attribute to make **Password** even stronger, as shown in the following code:

```
[StrongPassword(
    MinimumTotalCharacters = 12,
    MinimumDigitCharacters = 2,
    MinimumSpecialCharacters = 3)]
public virtual string Password { get; set; }
```

10. Start the site, log in as **Admin**, edit the **Start** page, and test the stronger password rule.

Creating a content-level validator

Next, we need to validate the two date/time properties, but we cannot use an attribute because they can only apply to a single property, and we need to compare start date to end date an ensure end is after start. We can do this by creating a class that implements `IValidate<T>`. The example will also show another benefit of this technique compared to attributes: different levels of severity.

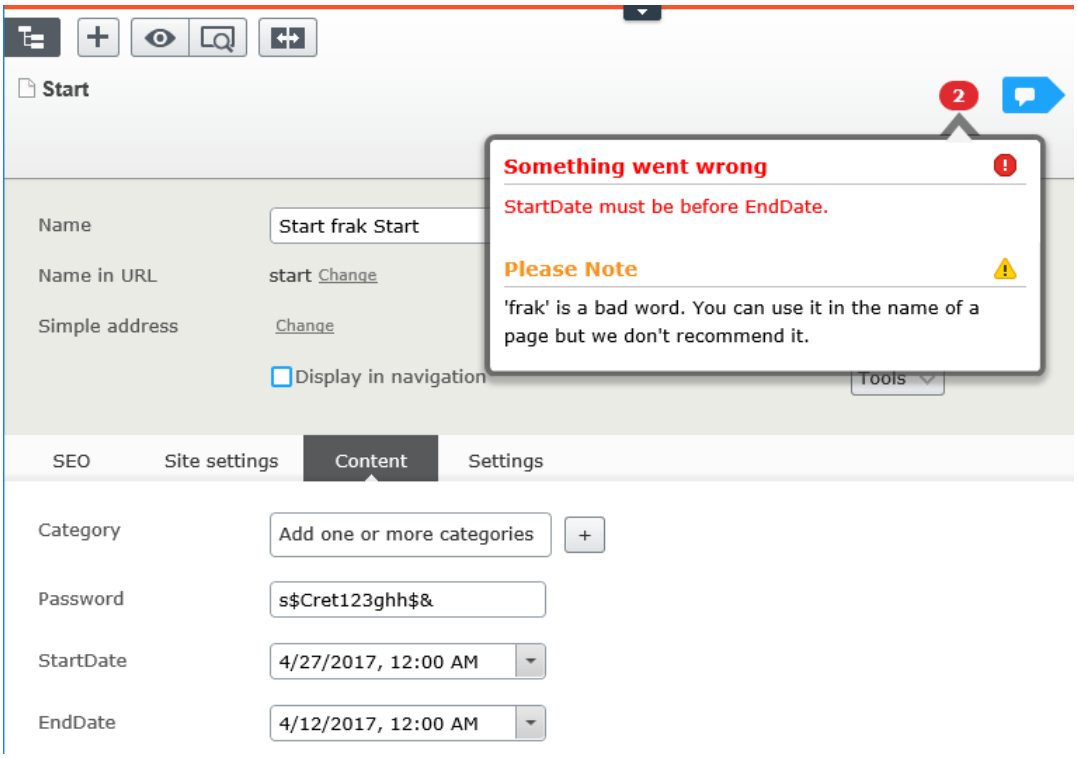
1. In **~\Business\Validation**, add a new class named **StartPageValidator.cs**.
2. Write a class, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using EPiServer.Validation;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.Validation
{
    public class StartPageValidator : IValidate<StartPage>
```

```
    {
        public IEnumerable<ValidationError> Validate(StartPage instance)
        {
            var errors = new List<ValidationError>();

            if (instance.StartDate >= instance.EndDate)
            {
                errors.Add(new ValidationError
                {
                    ErrorMessage = "StartDate must be before EndDate.",
                    PropertyName = "StartDate",
                    RelatedProperties = new string[] { "EndDate" },
                    Severity = ValidationErrorSeverity.Error
                });
            }

            if (instance.Name.ToLower().Contains("frak"))
            {
                errors.Add(new ValidationError
                {
                    ErrorMessage = "'frak' is a bad word. You can use it in the name
of a page but we don't recommend it.",
                    PropertyName = "Name",
                    Severity = ValidationErrorSeverity.Warning
                });
            }

            return errors;
        }
    }
}
```

3. Start the site and log in as **Admin**.

4. Edit the **Start** page and set **StartDate** to a value after **EndDate**, and add the word frak into the **Name**, as shown in the following screenshot:



5. Fix the date issue, but leave the frak, and note that you can save and publish despite the warning.

Exercise A2 – Exploring authentication and authorization

In this exercise, you will create a page type and template that shows security configuration:

- System security including which provider is configured, lists of stored and virtual roles.
- Current user security including if they have special access (admin or editor), the roles they belong too, and their claims.

Prerequisites: complete Exercise 0.1.

Reviewing authentication and authorization in an Alloy (MVC) site

1. Open the **Training** solution with the **AlloyAdvanced** project.
2. Open **~/Web.config**.
3. Find the `<authentication>` element, as shown in the following configuration:

```
<authentication mode="None">
  <forms name=".EPiServerLogin"
         loginUrl="Util/login.aspx"
         timeout="120"
         defaultUrl="~/" />
</authentication>
```

Alloy (MVC) project template disables authentication in the configuration file. This does not mean the Alloy site does not authenticate. For historical reasons, you must set authentication mode to **None** to enable federated authentication systems like **ASP.NET Identity**.

4. Find the `<membership>` and `<rolemanager>` elements, as shown in the following configuration:

```
<membership>
  <providers>
    <clear />
  </providers>
</membership>
<roleManager>
  <providers>
    <clear />
  </providers>
</roleManager>
```

Alloy (MVC) project template configures no **ASP.NET Membership** aka **Forms** providers.

5. Open **~/Startup.cs**. and review the **Configuration** method, as partially shown in the following code:

```
// Add CMS integration for ASP.NET Identity
app.AddCmsAspNetIdentity<ApplicationUser>();

// Remove to block registration of administrators
app.UseAdministratorRegistrationPage(
    () => HttpContext.Current.Request.IsLocal);

// Use cookie authentication
app.UseCookieAuthentication(new CookieAuthenticationOptions
```

Alloy (MVC) project template uses **ASP.NET Identity** with accounts stored in Episerver CMS database to authenticate users and authorize roles, and uses cookies for re-authentication.

Reviewing virtual roles and changing the default mapping for CmsEditors

1. Open the **Training** solution with the **AlloyAdvanced** project.
2. Open **~/Web.config**.
3. Find the `<virtualRoles>` element, as shown in the following configuration:

```
<episerver.framework>
  <appData basePath="App_Data" />
  <scanAssembly forceBinFolderScan="true" />
  <virtualRoles addClaims="true">
    <providers>
      <add name="Administrators"
           type="EPiServer.Security.WindowsAdministratorsRole, EPiServer.Framework" />
      <add name="Everyone"
           type="EPiServer.Security.EveryoneRole, EPiServer.Framework" />
      <add name="Authenticated"
           type="EPiServer.Security.AuthenticatedRole, EPiServer.Framework" />
      <add name="Anonymous"
           type="EPiServer.Security.AnonymousRole, EPiServer.Framework" />
      <add name="CmsAdmins"
           type="EPiServer.Security.MappedRole, EPiServer.Framework"
           roles="WebAdmins, Administrators" mode="Any" />
      <add name="CmsEditors"
           type="EPiServer.Security.MappedRole, EPiServer.Framework"
           roles="WebEditors" mode="Any" />
      <add name="Creator"
           type="EPiServer.Security.CreatorRole, EPiServer" />
      <add name="PackagingAdmins"
           type="EPiServer.Security.MappedRole, EPiServer.Framework"
           roles="WebAdmins, Administrators" mode="Any" />
```

In this default configuration, users in the Windows group named **Administrators**, or users in the database-stored role named **WebAdmins**, will become members of the virtual role named **CmsAdmins**, which gives access to the Global menu and Admin view, but not necessarily content. Users in the database-stored role named **WebEditors**, will become members of the virtual role named **CmsEditors**, which gives access to **Edit** and **Reports** in the Global menu.

4. Modify the entry for **CmsEditors**, so that a stored role named **ContentEditors** maps to **CmsEditors**, instead of **WebEditors**, as shown in the following markup:

```
<add name="CmsEditors"
     type="EPiServer.Security.MappedRole, EPiServer.Framework"
     roles="ContentEditors" mode="Any" />
```

5. Click after the end of the <virtualRoles> element, press *Ctrl* + *F* to find, and enter **WebAdmins**.
6. In the `<location path="EPiServer">` element, note the `<authorization>` element allows members of **WebEditors**, **WebAdmins**, or **Adminstrators** to access the **EPiServer** path, as shown in the following markup:

```
<location path="EPiServer">
  <system.web>
    ...
    <authorization>
      <allow roles="WebEditors, WebAdmins, Administrators" />
      <deny users="*" />
    </authorization>
```

7. Modify the **roles** attribute to use virtual roles instead of stored roles, as shown in the following markup:

```
<allow roles="CmsEditors, CmsAdmins" />
```

8. Press *Ctrl* + *F* to find, and enter **WebAdmins**.
9. In the `<location path="EPiServer/CMS/admin">` element, note the `<authorization>` element allows members of **WebEditors**, **WebAdmins**, or **Adminstrators** to access the **Views/Plugins** path, as shown in the following markup:

```
<location path="EPiServer/CMS/admin">
  <system.web>
    <authorization>
      <allow roles="WebAdmins, Administrators" />
      <deny users="*" />
    </authorization>
```

10. Modify the **roles** attribute to use the virtual role instead of stored roles, as shown in the following markup:

```
<allow roles="CmsAdmins" />
```

11. In the `<location path="Views/Plugins">` element, note the `<authorization>` element allows members of **WebEditors**, **WebAdmins**, or **Adminstrators** to access the **Views/Plugins** path, as shown in the following markup:

```
<location path="Views/Plugins">
  <system.web>
    <authorization>
      <allow roles="WebAdmins, WebEditors, Administrators" />
      <deny users="*" />
    </authorization>
```

12. Modify the **roles** attribute to use virtual roles instead of stored roles, as shown in the following markup:

```
<allow roles="CmsEditors, CmsAdmins" />
```

You have now removed any dependency on the existance of a stored group/role named **WebEditors**. Instead, any member of **ContentEditors** will have access to the Edit view in Episerver CMS.

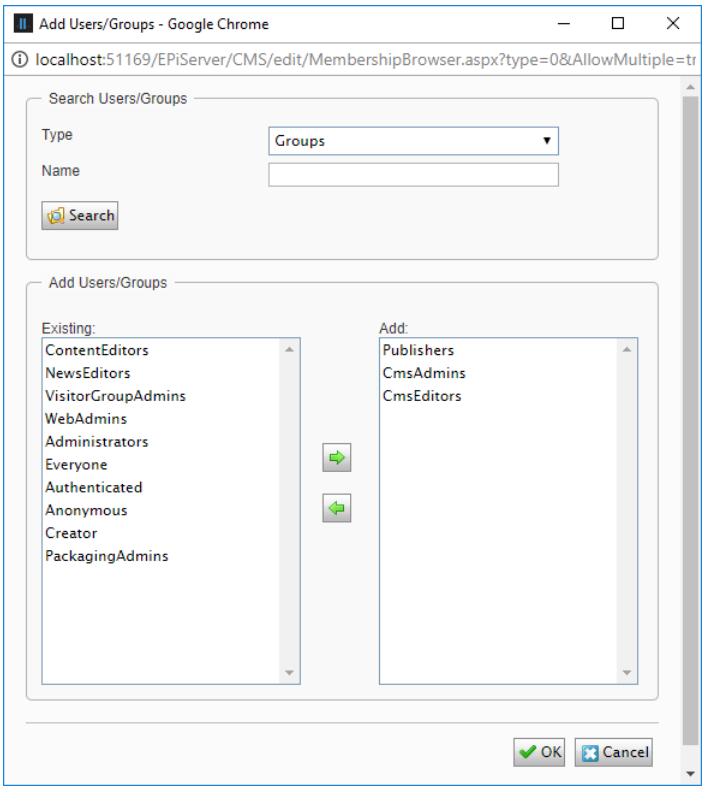
Creating some stored roles and users

1. Start the **AlloyAdvanced** site, and log in as **Admin**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Administer Groups**.
3. Create the following groups:
   - **NewsEditors**
   - **ContentEditors**
   - **Publishers**
4. Create the following users and make them members of the groups:
   - **Alice** is member of **WebAdmins**.
   - **Bob** is a member of **ContentEditors**, **NewsEditors**.
   - **Eve** is a member of **ContentEditors**, **Publishers**.

Assign access rights

1. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Set Access Rights**, and click **Root**.
2. Click **Add Users/Groups**.

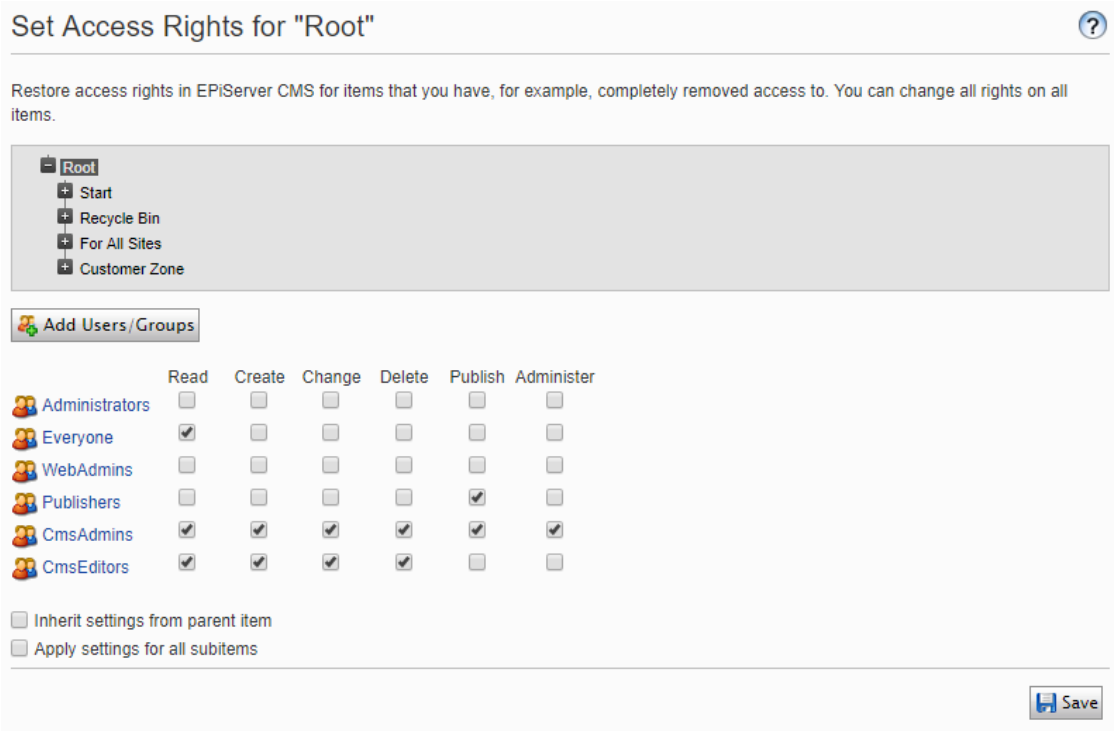
3. Search for **Groups**, add **Publishers**, **CmsAdmins**, and **CmsEditors**, and click **OK**, as shown in the following screenshot:



It is good practice to assign access rights to **CmsAdmins** and **CmsEditors** instead of **WebAdmins** and **WebEditors/ContentEditors** because it uses the mapping in configuration and therefore provides flexibility.

4. Modify the following roles and their access rights, as shown in the following screenshot:
    a. **Administrators**: clear all access rights
    b. **WebAdmins**: clear all access rights
    c. **Publishers**: Read, Publish
    d. **CmsAdmins**: set all access rights
    e. **CmsEditors**: Read, Create, Change, Delete

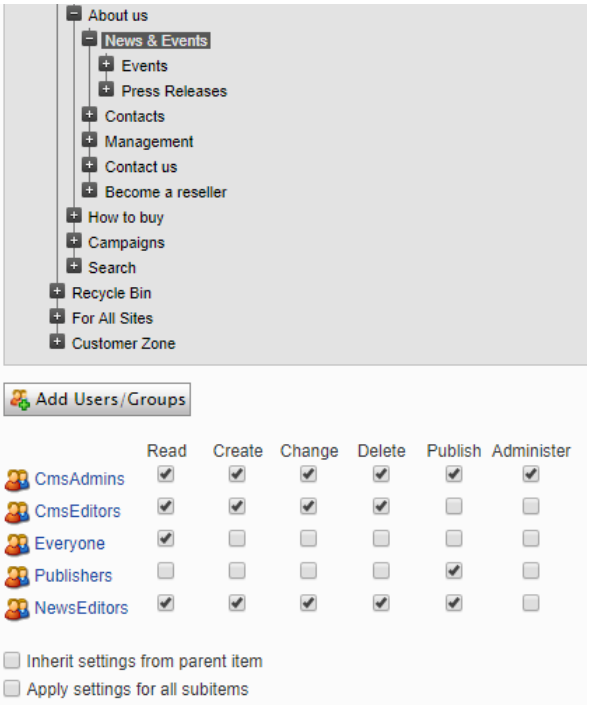
Clearing all access rights will remove the access control entry from the list when you click **Save**.



Five of the six access rights (all except **Create**), apply to the content item that is selected, for example, "Root" in the previous screenshot. Applying **Create** access right to **"Root"** does not mean you can create the *root*. It means you can create children *underneath* **"Root"**.

5. Click **Save**.
6. In the **Set Access Rights** content tree, expand **Start**, expand **About us**, and click **News & Events**.
7. Clear the **Inherit settings from parent item** check box.
8. Click **Add Users/Groups**.
9. Search for **Groups**, add the **NewsEditors** group, and click **OK**.

10. Assign the following access rights to **NewsEditors**: Read, Create, Change, Delete, Publish, as shown in the following screenshot:



11. Click **Save**.
12. Close the browser.

Creating a security page type and template

1. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **SecurityPage.cs**.
2. Modify its statements, as shown in the following code, and note the following:
    - SecurityPage is included as an allowed child on StartPage.
    - The class has two ignored properties, so we don't need to define a custom view model with those properties, but those values will be cached, so we must be careful to always clear them in the controller.

```
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;
using System.Security.Claims;

namespace AlloyAdvanced.Models.Pages
{
    [ContentType(DisplayName = "Security",
        GroupName = Global.GroupNames.Specialized,
        Description = "Use this page to see security details of the system and
current user.")]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class SecurityPage : SitePageData
    {
        [Ignore]
        public User SecurityUser { get; set; }

        [Ignore]
        public System SecuritySystem { get; set; }

        public class System
```

```
            {
                public string Provider { get; set; }

                public string[] VirtualRoles { get; set; }
                public string[] StoredRoles { get; set; }
            }

            public class User
            {
                public string Name { get; set; }
                public bool IsAnonymous { get; set; }
                public bool IsAdministrator { get; set; }
                public bool IsEditor { get; set; }
                public bool HasAccessToPlugins { get; set; }

                public Claim[] Claims { get; set; }
                public string[] Roles { get; set; }
            }
        }
    }
```

3. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **SecurityPageController**.
4. Modify its contents, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Framework.Configuration;
using EPiServer.Security;
using EPiServer.Shell.Security;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Web.Mvc;
using System.Web.Security;

namespace AlloyAdvanced.Controllers
{
    public class SecurityPageController : PageControllerBase<SecurityPage>
    {
        private readonly UIRoleProvider roles;

        public SecurityPageController(UIRoleProvider roles)
        {
            this.roles = roles;
        }

        public ActionResult Index(SecurityPage currentPage)
        {
            currentPage.SecuritySystem = new SecurityPage.System();
            currentPage.SecurityUser = new SecurityPage.User();

            // get current user security information

            var principal = PrincipalInfo.CurrentPrincipal;

            if (principal is RolePrincipal) // ASP.NET Membership
            {
                currentPage.SecuritySystem.Provider = "ASP.NET Membership";
            }
```

```
            else if (principal is ClaimsPrincipal) // ASP.NET Identity
            {
                currentPage.SecuritySystem.Provider = "ASP.NET Identity";
            }

            currentPage.SecurityUser.Claims =
                (principal.Identity as ClaimsIdentity).Claims.ToArray();

            currentPage.SecurityUser.Roles = currentPage.SecurityUser
                .Claims.Where(c => c.Type ==
"http://schemas.microsoft.com/ws/2008/06/identity/claims/role")
                .Select(c => c.Value).ToArray();

            currentPage.SecurityUser.Name = principal.Identity.Name;
            currentPage.SecurityUser.IsAnonymous =
                !principal.Identity.IsAuthenticated;
            currentPage.SecurityUser.HasAccessToPlugins =
                PrincipalInfo.Current.HasPathAccess("Views/Plugins");
            currentPage.SecurityUser.IsAdministrator =
                principal.IsInRole("CmsAdmins");
            currentPage.SecurityUser.IsEditor = principal.IsInRole("CmsEditors");

            // or use following that check access to paths /admins /edit
            currentPage.SecurityUser.IsAdministrator = PrincipalInfo.HasAdminAccess;
            currentPage.SecurityUser.IsEditor = PrincipalInfo.HasEditAccess;

            // get system security information
            currentPage.SecuritySystem.StoredRoles =
                roles.GetAllRoles().Select(r => r.Name).ToArray();
            ProviderSettingsCollection virtualRoles =
                EPiServerFrameworkSection.Instance.VirtualRoles.Providers;
            var list = new List<string>();
            foreach (var setting in virtualRoles.Cast<ProviderSettings>())
            {
                string item = setting.Name;
                if (setting.ElementInformation.Properties
                    .Cast<PropertyInformation>().Any(pi => pi.Name == "roles"))
                {
                    item += " <-- " + setting.ElementInformation
                        .Properties["roles"].DefaultValue;
                }
                list.Add(item);
            }
            currentPage.SecuritySystem.VirtualRoles = list.ToArray();

            // create view model
            var viewmodel = PageViewModel.Create(currentPage);
            return View(viewmodel);
        }
    }
}
```

5. In **Solution Explorer**, right-click **~/Views**, add a folder named **SecurityPage**.
6. In **~\Views\SecurityPage**, add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
7. Modify its contents, as shown in the following markup:

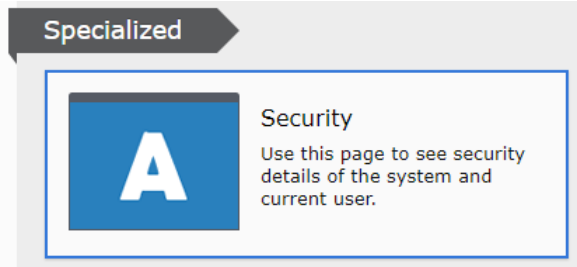
```
@using AlloyAdvanced.Models.Pages
@model PageViewModel<SecurityPage>
<div class="row well well-large">
    <div class="span3">
        <h2>System</h2>
        <div class="alert alert-heading">
```

```
                @Model.CurrentPage.SecuritySystem.Provider
            </div>
        </div>
        <div class="span2">
            <h3>Stored Roles</h3>
            <ul>
                @foreach (string storedRole
                    in Model.CurrentPage.SecuritySystem.StoredRoles)
                {
                    <li>@storedRole</li>
                }
            </ul>
        </div>
        <div class="span2">
            <h3>Virtual Roles</h3>
            <ul>
                @foreach (string virtualRole
                    in Model.CurrentPage.SecuritySystem.VirtualRoles)
                {
                    <li>@virtualRole</li>
                }
            </ul>
        </div>
    </div>
    <div class="row well well-large">
        <div class="span3">
            <h2>Current User</h2>
            <div class="alert alert-heading">
                @if (Model.CurrentPage.SecurityUser.IsAnonymous)
                {
                    @Html.Raw("User is anonymous.")
                }
                else
                {
                    @Html.Raw(Model.CurrentPage.SecurityUser.Name + " is ")
                    if (Model.CurrentPage.SecurityUser.IsAdministrator)
                    {
                        <text>a CMS administrator.</text>
                    }
                    else if (Model.CurrentPage.SecurityUser.IsEditor)
                    {
                        <text>a CMS editor.</text>
                    }
                }
            </div>
            @if (Model.CurrentPage.SecurityUser.HasAccessToPlugins)
            {
            <div class="alert alert-heading">
                User has access to /Views/Plugins
            </div>
            }
        </div>
        <div class="span2">
            <h3>Roles</h3>
            <ul>
                @foreach (var role in Model.CurrentPage.SecurityUser.Roles)
                {
                    <li>@role</li>
                }
            </ul>
        </div>
        <div class="span6">
```

```
            <h3>Claims</h3>
            <table class="table table-condensed table-striped table-bordered">
                <tr><th>Type</th><th>Value</th></tr>
                @foreach (var claim in Model.CurrentPage.SecurityUser.Claims)
                {
                    <tr><td>@claim.Type</td><td>@claim.Value</td></tr>
                }
            </table>
        </div>
    </div>
```

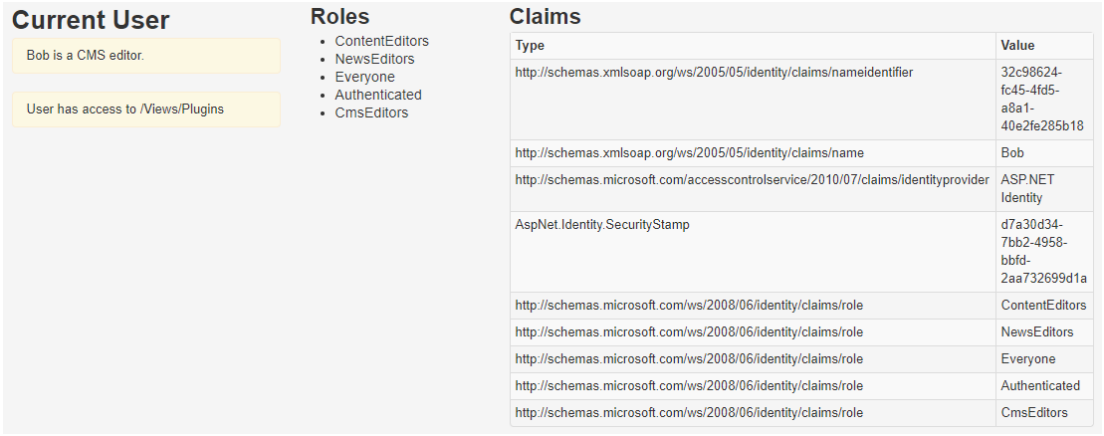
Testing the security page

1. Start the **AlloyAdvanced** site, and log in as **Admin**.
2. Create a new page under **Start**.
3. Choose the **Security** page type, and name the new page **Security**.



4. Publish the page.
5. Switch to **Live** view, and log out.
6. View the **Security** page, and note what is displayed for a visitor, as shown in the following screenshot:

7. Log in as **Bob**, as shown in the following screenshot:



Current User

Bob is a CMS editor.

User has access to /Views/Plugins

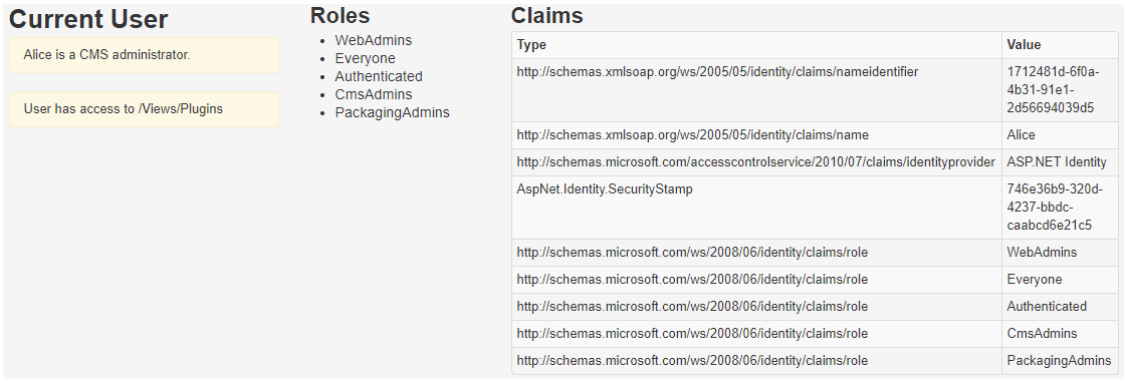
Roles

- ContentEditors
- NewsEditors
- Everyone
- Authenticated
- CmsEditors

Claims

Type	Value
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier	32c98624-fc45-4fd5-a8a1-40e2fe285b18
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	Bob
http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider	ASP.NET Identity
AspNet.Identity.SecurityStamp	d7a30d34-7bb2-4958-bbfd-2aa732699d1a
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	ContentEditors
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	NewsEditors
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	Everyone
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	Authenticated
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	CmsEditors

8. Log in as **Alice**, as shown in the following screenshot:



Current User

Alice is a CMS administrator.

User has access to /Views/Plugins

Roles

- WebAdmins
- Everyone
- Authenticated
- CmsAdmins
- PackagingAdmins

Claims

Type	Value
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier	1712481d-6f0a-4b31-91e1-2d56694039d5
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	Alice
http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider	ASP.NET Identity
AspNet.Identity.SecurityStamp	746e36b9-320d-4237-bbdc-caabcd6e21c5
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	WebAdmins
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	Everyone
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	Authenticated
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	CmsAdmins
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	PackagingAdmins

Exercise A3 – Completing some challenges

In this exercise, you will create several extensions to the Alloy site including:

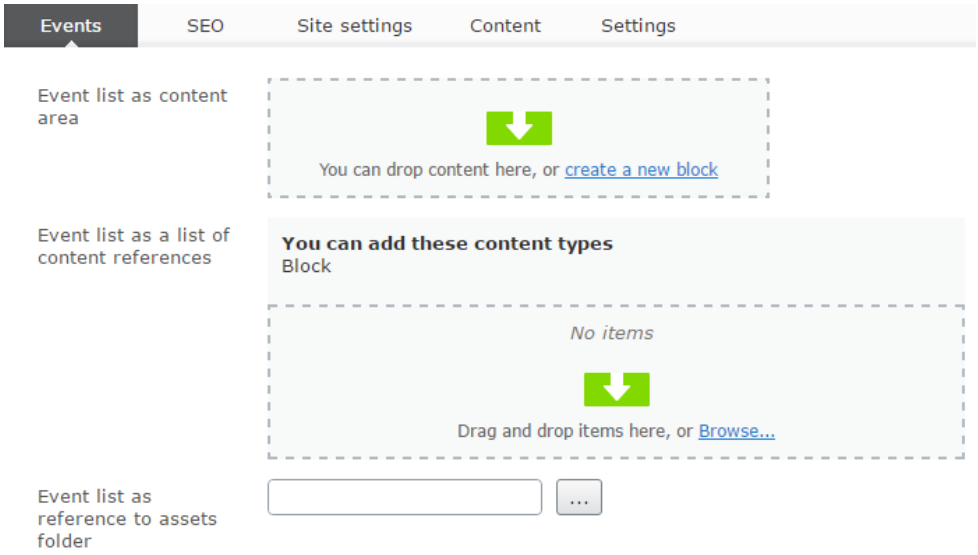
- Some block types and some properties that can contain those blocks.
- Listening for content events to provide complex validation.
- Adding a site-level page property.

Prerequisites: complete Exercise 0.1.

Creating some block types

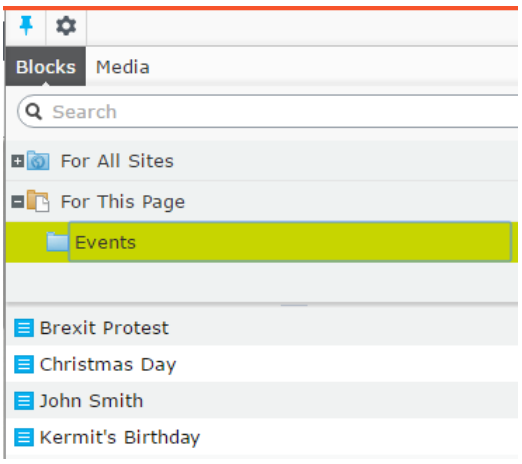
When creating content types in the Alloy site, you should apply the site-specific attributes named `[SiteContentType]` and `[SiteImageUrl]`, instead of the usual Episerver attributes named `[ContentType]` and `[ImageUrl]`.

1. Open the solution with the **AlloyAdvanced** project.
2. Create a block type named **EventBlock** with properties: **Title**, **When**, **Where**, **Description**.
3. Create a block type named **PersonBlock** with: **FirstName**, **LastName**, **BirthDate**, **Summary**.
4. Create controller-less block templates (aka views) for both EventBlock and PersonBlock.
5. Add three properties to the **StartPage**, all of them should be able to contain any number of instances of the EventBlock, but not other content types, and put them in a tab named **Events**:
   - **EventListAsContentArea**: a content area into which an editor could drag and drop one or more shared/global EventBlock instances.
   - **EventListAsListOfContentReferences**: a list of content links to one or more shared/global EventBlock instances.
   - **EventListAsReferenceToAssetsFolder**: a reference to an assets folder that could contain one or more shared/global EventBlock instances.
6. Explore the user experiences that a content editor has for each mechanism for creating a list of events, as shown in the following screenshot:



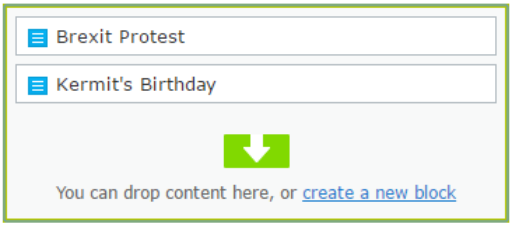
7. Modify the **~/Views/StartPage/Index.cshtml** view to output the events in each of the three properties.
8. Start the site, and log in as **Admin**.

9. In the **Assets** pane, in the **For This Page** folder (while Start is selected), create a folder named **Events**.
10. In the **Events** folder, add three event blocks, and one person block, as shown in the following screenshot:



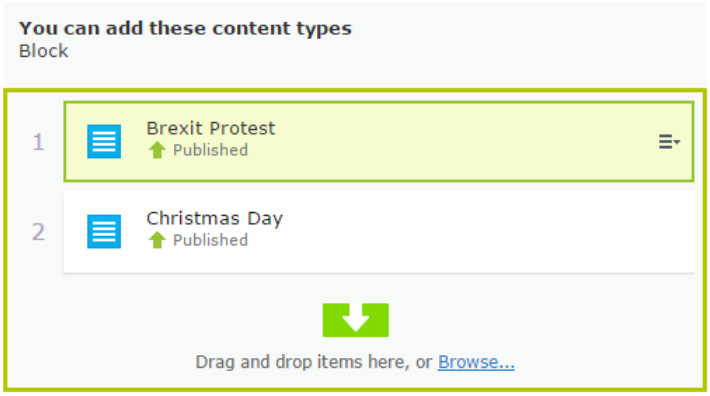
11. Edit the **Start** page, and switch to **All Properties** view.
12. Drag and drop blocks from the **Events** folder into the content area property. If you have written your code correctly, only event blocks should be allowed to be dropped. The editor controls the order that the blocks appear in the content area, as shown in the following screenshot:

Event list as content area

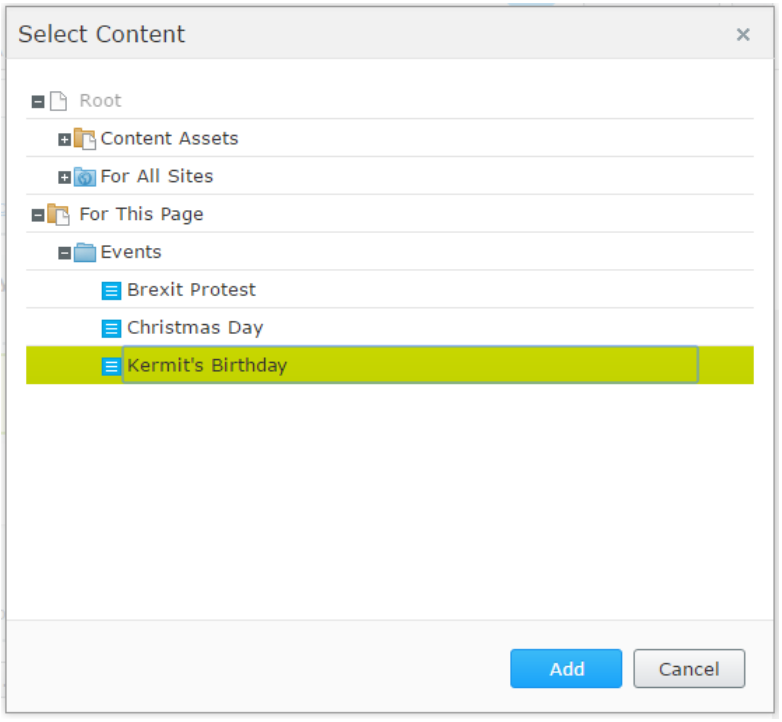


13. Drag and drop blocks from the **Events** folder into the list of content references property. If you have written your code correctly, only event blocks should be allowed to be dropped, as shown in the following screenshot:

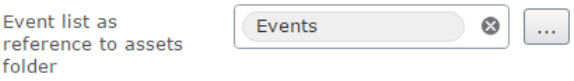
Event list as a list of content references



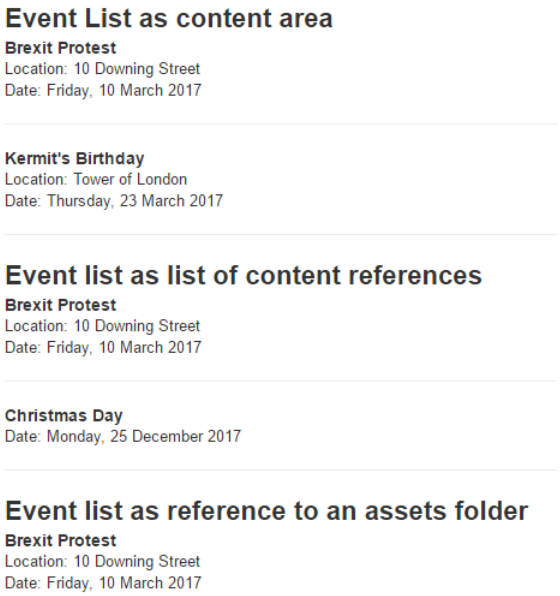
14. Click **Browse...**, and note the **Select Content** dialog box is pre-filtered to only show event blocks, as shown in the following screenshot:



15. Set the third property to reference the **Events** folder, as shown in the following screenshot:



16. **Publish** the changes to the Start page.
17. View the Start page as a visitor, as shown in the following screenshot:



Which mechanism is easiest for the content editor? Which mechanism is easiest for the developer to implement? Which gives the most control? Which do you personally prefer?

Listening for content events

1. Write code that listens for when content is published and prevents non-administrators from using the phrase “iPhone 8” until after 1st September 2017.

Hint: create an Episerver **Initialization Module**, and get an instance of the **IContentEvents** service.

Creating a site-level property to enable prefetching

Modern browsers that support HTML5 allow a page to specify URLs that should be prefetched. This can improve perceived performance for a site for pages that are commonly requested.

1. Add a new property to **SitePageData** named **PrefetchLinks** that a content editor can manually add one or more links and URLs to.
2. When the PrefetchLinks property is not empty, it should be rendered by inserting `<link>` elements into the `<head>` of a layout. For example, on the **Start** page, visitors may commonly next click **About us**, so the `<head>` should look contain the following markup (in this case multiple languages have been enabled):

```
<link rel="prefetch" href="/en/about-us">
```

Hint: the **LinkItemCollection** type allows an editor to manually maintain a list of links.

Exercise A4 – Exploring Episerver developer tools

In this exercise, you will explore some of the features of the experimental Episerver developer tools.

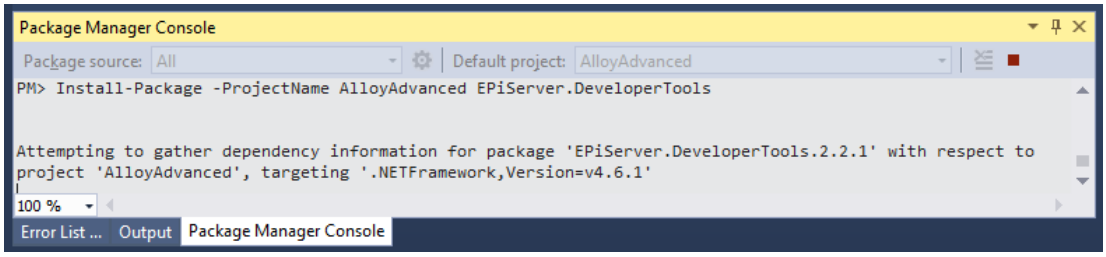
Prerequisites: complete Exercise 0.1.

Installing Episerver Developer Tools

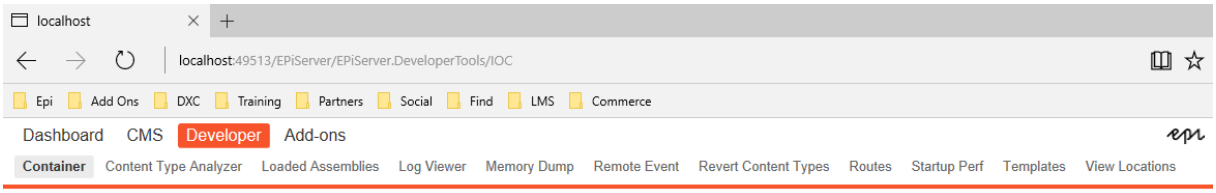
Episerver Developer Tools is experimental. We recommend that you create a backup copy of an Episerver project before installing its NuGet package so that you can recover the project if it causes problems.

1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. In **Package Manager Console**, set Package source to **All**, and then enter the following command to install the developer tools package:

   ```
   Install-Package -ProjectName AlloyAdvanced EPiServer.DeveloperTools
   ```

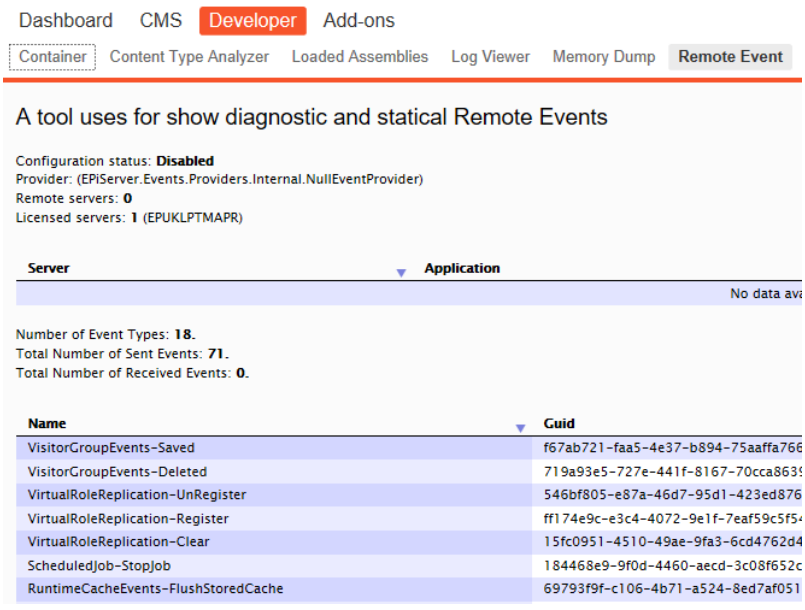


3. Start the site and log in as **Admin**.
4. Show the **Global** menu and note the **Developer** menu, as shown in the following screenshot:

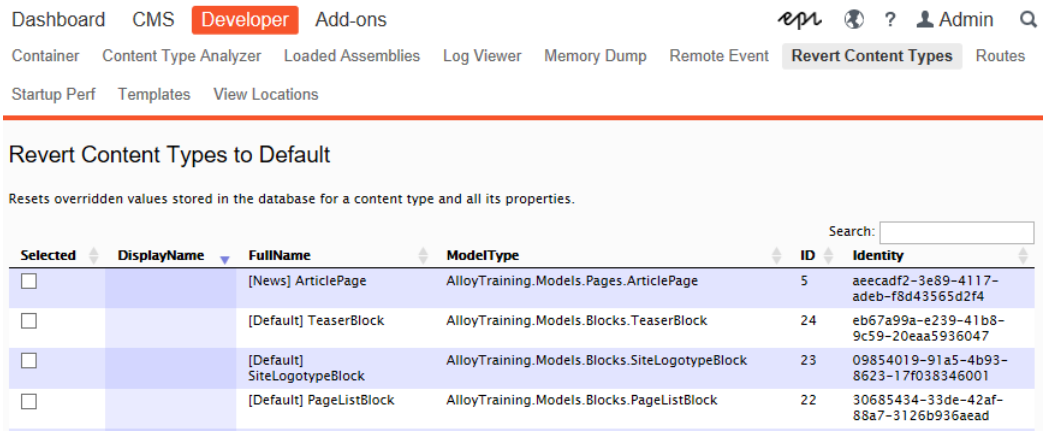


5. Click each of the submenus to review what is available in this developer tool.

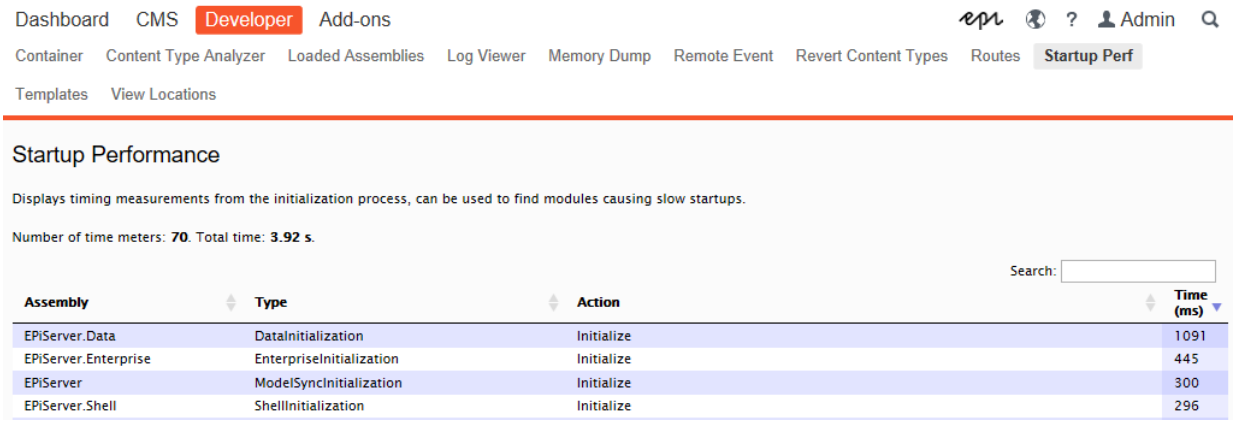
Remote Event



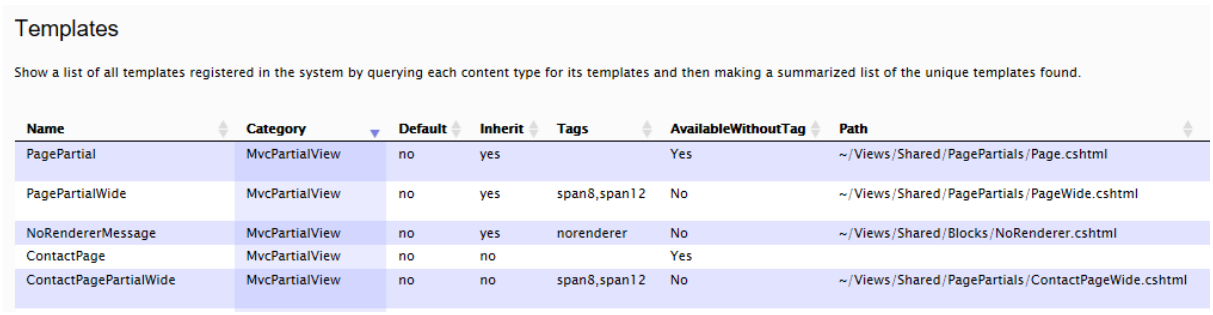
Revert Content Types



Startup Perf



Templates



Module B – Working with Content using APIs

Goal

The overall goal of the exercises in this module is to see how you can work programmatically with Episerver CMS APIs to automate and control the creation of content, and to integrate with external social media sites and sources of content.

Exercise B1 – Implementing a Share This block

In this exercise, you will create a block to allow visitors to share a link to the current page within the Alloy site on Facebook, Twitter, or LinkedIn.

Prerequisites: complete Exercise 0.1.

Creating a sharing block and view model

When creating content types in the Alloy site, you should apply the site-specific attributes named `[SiteContentType]` and `[SiteImageUrl]`, instead of the usual Episerver attributes named `[ContentType]` and `[ImageUrl]`.

1. Open the solution with the **AlloyAdvanced** project.
2. In **~\Models\Blocks**, add a new **Block Type** named **ShareThisBlock**.

```
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;
using System.ComponentModel.DataAnnotations;

namespace AlloyAdvanced.Models.Blocks
{
    [SiteContentType(
        DisplayName = "Share This",
        Description = "Allows a visitor to share a link to a page via Twitter,
Facebook, or LinkedIn.")]
    [SiteImageUrl]
    public class ShareThisBlock : BlockData
    {
        [Display(
            Name = "Display Facebook share",
            GroupName = SystemTabNames.Content,
            Order = 100)]
        public virtual bool ShareToFacebook { get; set; }

        [Display(
            Name = "Display Twitter share",
            GroupName = SystemTabNames.Content,
            Order = 200)]
        public virtual bool ShareToTwitter { get; set; }

        [Display(
            Name = "Display LinkedIn share",
            GroupName = SystemTabNames.Content,
            Order = 300)]
        public virtual bool ShareToLinkedin { get; set; }

        public override void SetDefaultValues(ContentType contentType)
        {
            base.SetDefaultValues(contentType);
```

```
            ShareToFacebook = true;
            ShareToTwitter = true;
            ShareToLinkedin = true;
        }
    }
}
```

3. In **~\Models\ViewModels**, add a class named **ShareThisBlockViewModel**.

```
using AlloyAdvanced.Models.Blocks;

namespace AlloyAdvanced.Models.ViewModels
{
    public class ShareThisBlockViewModel
    {
        public ShareThisBlock Settings { get; set; }

        public string FriendlyUrl { get; set; }
    }
}
```

Creating the block template (controller and view)

1. In **~\Controllers**, add a **Block Controller (MVC)** named **ShareThisBlockController**.

```
using System.Web.Mvc;
using EPiServer;
using EPiServer.Core;
using EPiServer.Web.Mvc;
using AlloyAdvanced.Models.Blocks;
using EPiServer.Web.Routing;
using AlloyAdvanced.Models.ViewModels;

namespace AlloyAdvanced.Controllers
{
    public class ShareThisBlockController : BlockController<ShareThisBlock>
    {
        private readonly IPageRouteHelper pageRouteHelper;
        private readonly UrlResolver urlResolver;

        public ShareThisBlockController(IPageRouteHelper pageRouteHelper,
                                        UrlResolver urlResolver)
        {
            this.pageRouteHelper = pageRouteHelper;
            this.urlResolver = urlResolver;
        }

        public override ActionResult Index(ShareThisBlock currentBlock)
        {
            var model = new ShareThisBlockViewModel();
            PageData page = pageRouteHelper.Page;
            model.FriendlyUrl = UriSupport.AbsoluteUrlBySettings(
                urlResolver.GetUrl(page.ContentLink));
            model.Settings = currentBlock;
            return PartialView(model);
        }
    }
}
```

2. Add a folder to **~\Views** named **ShareThisBlock**.
3. In **~\Views\ShareThisBlock**, add a partial view named **Index.cshtml**.

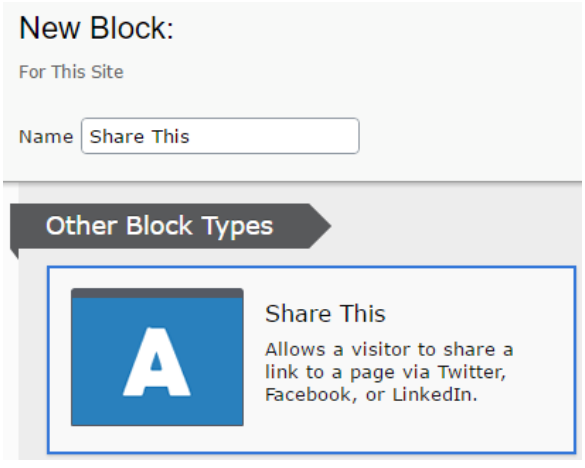
```
@model ShareThisBlockViewModel
```

```
@if (Model.Settings.ShareToFacebook)
{
    <a href="https://www.facebook.com/sharer/sharer.php?u=@Model.FriendlyUrl"
       target="_blank">Share on Facebook</a>
}
@if (Model.Settings.ShareToTwitter)
{
    <script src="//platform.twitter.com/widgets.js" type="text/javascript"></script>
    <a href="https://twitter.com/share?url=@Model.FriendlyUrl"
       class="twitter-share-button" datacount="none">Tweet</a>
}
@if (Model.Settings.ShareToLinkedin)
{
    <script src="http://platform.linkedin.com/in.js" type="text/javascript"></script>
    <script type="IN/Share" data-url="@Model.FriendlyUrl">
    </script>
}
@if (!((Model.Settings.ShareToFacebook) || (Model.Settings.ShareToTwitter)
    || (Model.Settings.ShareToLinkedin)))
{
    <div class="alert alert-info">
        At least one Share To option must be switched on!
    </div>
}
```

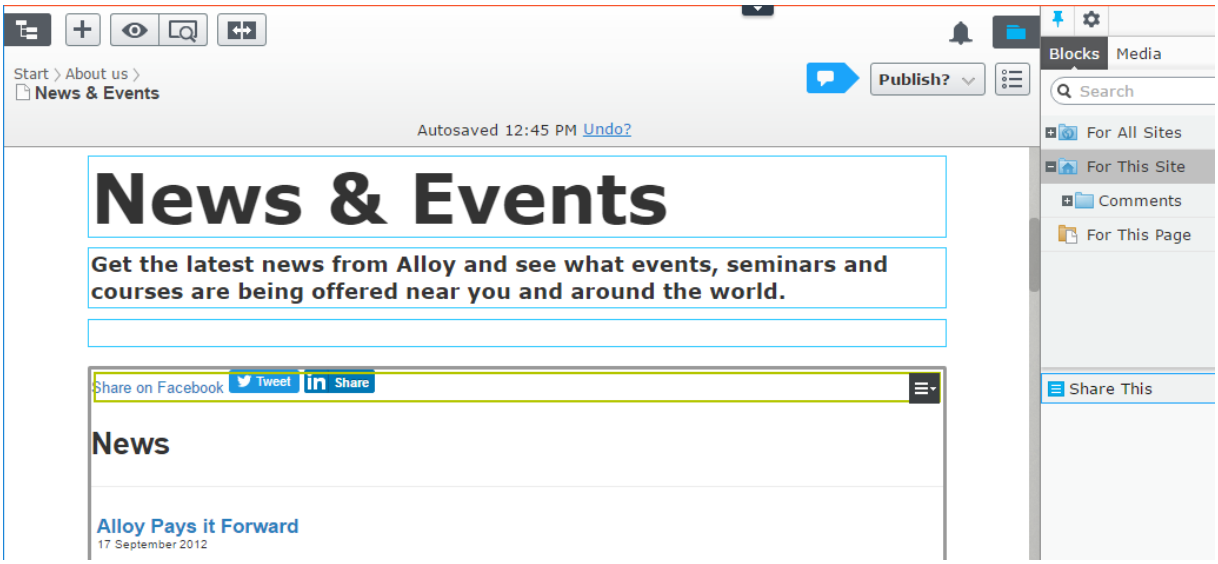
To share stuff on Facebook, a **Like** button is usually used. However, Facebook needs to be able to crawl the page (reading title and such) to get this button working. When running on localhost, Facebook cannot access the page and the box is removed. In a real scenario, when testing a Like button you can use a local tunnel or updated hosts file to give Facebook access to the test site. For simplicity, we use a text link instead in this exercise.

Testing the Share This block

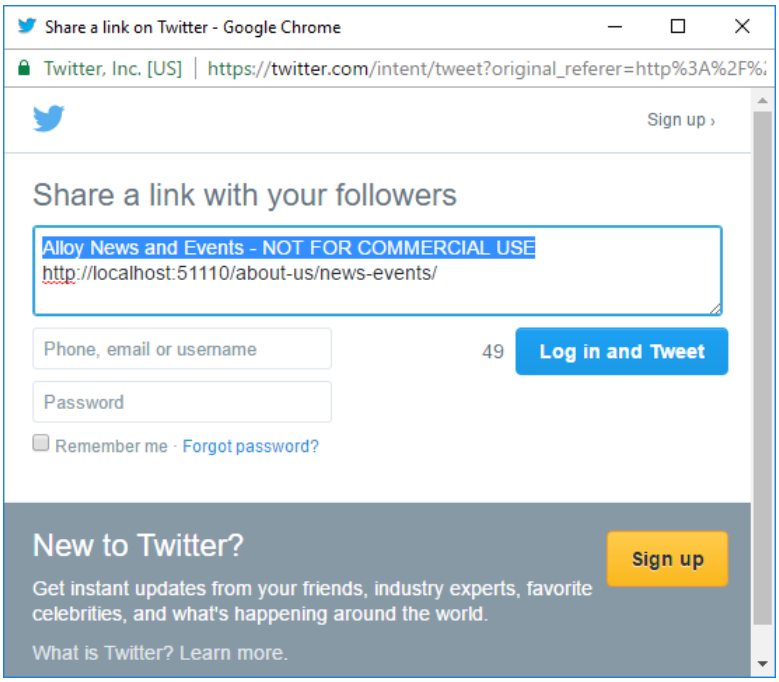
1. Start the site, and log in as **Admin**.
2. Create an instance of the **Share This** block in the **For This Site** folder.



3. Publish the block instance, and then add it to one of the pages with a content area, for example, **News & Events**, as shown in the following screenshot:



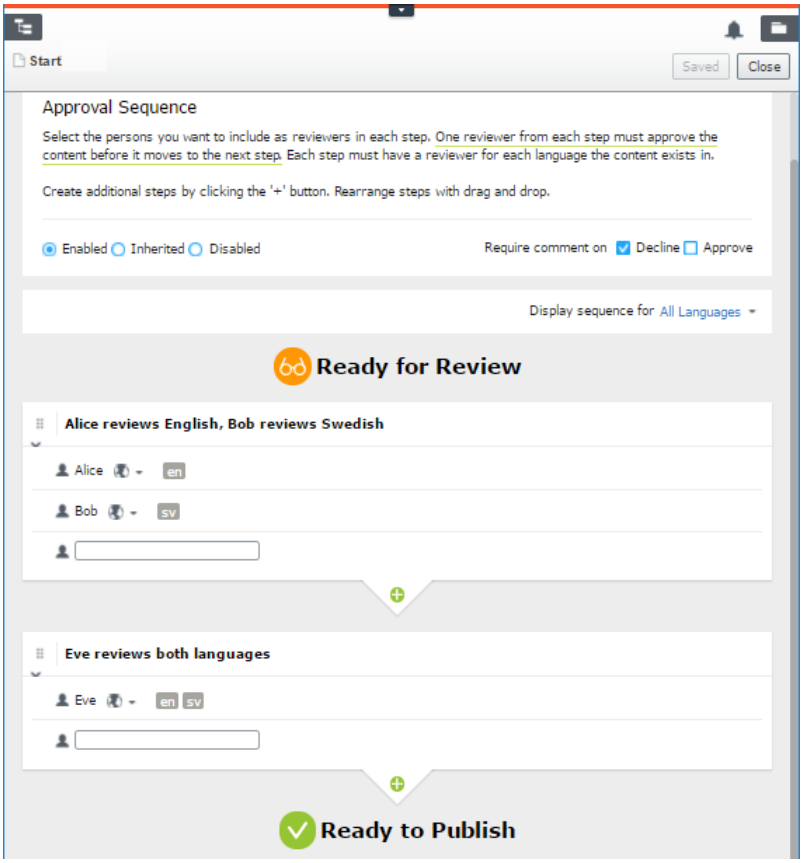
4. **Publish** the **News & Events** page and view it as an anonymous visitor. Click **Tweet** to see a pop up window allowing you to post a link to Twitter with a preview, as shown in the following screenshot:



Exercise B2 – Programming content approvals

In this exercise, you will programmatically define a content approval sequence, and programmatically manage approving and rejecting. Users will be programmatically created for demonstrating the feature.

Prerequisites: complete Exercise 0.1.



Creating a page type and view model for managing content appovals

This page only exists for demonstrating the features. In a real site, you would implement this as a plug-in or gadget.

8. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **ContentApprovalManagerPage.cs**.
9. Modify its statements, as shown in the following code:

```
namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(DisplayName = "Content Approvals Manager",
        Description = "This page demonstrates how to programmatically manage
content approvals.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class ContentApprovalsManagerPage : SitePageData
    {
    }
}
```

10. In **Solution Explorer**, right-click **~/Models/ViewModels**, and add a class named **ContentApprovalsManagerPageViewModel.cs**.
11. Modify its contents, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using EPiServer.Approvals.ContentApprovals;

namespace AlloyAdvanced.Models.ViewModels
{
    public class ContentApprovalsManagerPageViewModel
        : PageViewModel<ContentApprovalsManagerPage>
    {
        public ContentApprovalDefinition ApprovalDefinition { get; set; }
        public ContentApproval Approval { get; set; }

        public ContentApprovalsManagerPageViewModel(
            ContentApprovalsManagerPage currentPage) : base(currentPage)
        {
        }
    }
}
```

Creating a page template for managing content appovals

When creating controllers in the Alloy site, you should inherit from the site-specific class named `PageControllerBase<T>`, instead of the usual Episerver class named `PageController<T>`.

1. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **ContentApprovalManagerPageController**.
2. Modify its contents, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer;
using EPiServer.Approvals;
using EPiServer.Approvals.ContentApprovals;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.DataAccess;
using EPiServer.Security;
using EPiServer.Shell.Security;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class ContentApprovalsManagerPageController
        : PageControllerBase<ContentApprovalsManagerPage>
    {
        private const string admins = "WebAdmins";
        private const string editors = "WebEditors";

        private const string userName1 = "Alice";
        private const string userName2 = "Bob";
        private const string userName3 = "Eve";

        private const string password = "Pa$$w0rd";
        private const string emailBase = "@alloy.com";

        private readonly IApprovalDefinitionRepository repoDefinitions;
        private readonly IContentRepository repoContent;
        private readonly IApprovalRepository repoApprovals;
```

```
        private readonly IApprovalEngine engine;

        public ContentApprovalsManagerPageController(
            IApprovalDefinitionRepository repoDefinitions,
            IContentRepository repoContent,
            IApprovalRepository repoApprovals,
            IApprovalEngine engine,
            UIRoleProvider roles,
            UIUserProvider users,
            IContentSecurityRepository repoSecurity)
        {
            this.repoDefinitions = repoDefinitions;
            this.repoContent = repoContent;
            this.repoApprovals = repoApprovals;
            this.engine = engine;

            // if the editors role does not exist, create it and assign access rights
            if (!roles.RoleExists(editors))
            {
                roles.CreateRole(editors);

                var permissions = repoSecurity.Get(ContentReference.RootPage)
                    .CreateWritableClone() as IContentSecurityDescriptor;

                permissions.AddEntry(new AccessControlEntry(editors,
                    AccessLevel.Create | AccessLevel.Edit | AccessLevel.Delete |
                    AccessLevel.Read | AccessLevel.Publish));

                repoSecurity.Save(ContentReference.RootPage,
                    permissions, SecuritySaveType.Replace);
            }

            // create three users and add them to roles

            UIUserCreateStatus status;
            IEnumerable<string> errors = Enumerable.Empty<string>();

            if (users.GetUser(userName1) == null)
            {
                users.CreateUser(
                    userName1, password,
                    email: userName1.ToLower() + emailBase,
                    passwordQuestion: null, passwordAnswer: null,
                    isApproved: true, status: out status, errors: out errors);

                roles.AddUserToRoles(userName1, new string[] { admins });
            }

            if (users.GetUser(userName2) == null)
            {
                users.CreateUser(
                    userName2, password, userName2.ToLower() + emailBase,
                    null, null, true, out status, out errors);

                roles.AddUserToRoles(userName2, new string[] { editors });
            }

            if (users.GetUser(userName3) == null)
            {
                users.CreateUser(
                    userName3, password, userName3.ToLower() + emailBase,
                    null, null, true, out status, out errors);
```

```
            roles.AddUserToRoles(userName3, new string[] { editors });
        }
    }

    public async Task<ActionResult> Index(
        ContentApprovalsManagerPage currentPage,
        string task, int? stepIndex, string user, string decision)
    {
        var viewmodel = new ContentApprovalsManagerPageViewModel(currentPage);

        if (!string.IsNullOrWhiteSpace(task))
        {
            switch (task)
            {
                case "createDefinition":

                    var langEN = new[] { CultureInfo.GetCultureInfo("en") };
                    var langSV = new[] { CultureInfo.GetCultureInfo("sv") };

                    var def = new ContentApprovalDefinition
                    {
                        ContentLink = ContentReference.StartPage,
                        Steps = new List<ApprovalDefinitionStep>
                        {
                            new ApprovalDefinitionStep(
                                "Alice reviews English, Bob reviews Swedish",
                            new[]
                            {
                                new ApprovalDefinitionReviewer(userName1, langEN),
                                new ApprovalDefinitionReviewer(userName2, langSV),
                            }),
                            new ApprovalDefinitionStep(
                                "Eve reviews both languages", new[]
                            {
                                new ApprovalDefinitionReviewer(userName3,
                                    langEN.Union(langSV))
                            })
                        },
                        RequireCommentOnReject = true
                    };

                    await repoDefinitions.SaveAsync(def);

                    break;

                case "modifyStart":

                    var start = repoContent.Get<StartPage>(
                        ContentReference.StartPage)
                        .CreateWritableClone() as StartPage;

                    start.Name += "X";

                    repoContent.Save(content: start,
                        action: SaveAction.RequestApproval,
                        access: AccessLevel.NoAccess);

                    break;

                case "processStep":
```

```
                        var approval = await repoApprovals
                            .GetAsync(ContentReference.StartPage);

                        if (decision == "Approve")
                        {
                            await engine.ApproveStepAsync(
                                id: approval.ID,
                                username: user,
                                stepIndex: stepIndex.Value,
                                comment: "I approve: the page looks great!");
                        }
                        else
                        {
                            await engine.RejectStepAsync(
                               id: approval.ID,
                               username: user,
                               stepIndex: stepIndex.Value,
                               comment: "I decline: the page looks horrible!");
                        }
                        break;
                    }
                }

                // GetAsync(ContentReference) extension methods need
                // using EPiServer.Approvals.ContentApprovals

                viewmodel.ApprovalDefinition = await repoDefinitions
                    .GetAsync(ContentReference.StartPage);

                viewmodel.Approval = await repoApprovals
                    .GetAsync(ContentReference.StartPage);

                return View(viewmodel);
            }
        }
    }
}
```

This controller will create the **WebEditors** group (if does not already exist) and assign all access rights except Administer for it to the Root page. It will also create three users (if they don't already exist): **Alice** (a member of WebAdmins), **Bob** (a member of WebEditors), and **Eve** (a member of WebEditors). They will all have a password of **Pa$$w0rd**.

3. In **Solution Explorer**, right-click **~/Views**, add a folder named **ContentApprovalsManagerPage**.
4. In **~\Views\ContentApprovalsManagerPage**, add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
5. Modify its contents, as shown in the following markup:

```
@using EPiServer.Web.Mvc.Html
@model AlloyAdvanced.Models.ViewModels.ContentApprovalsManagerPageViewModel
<div>
    <div class="alert alert-info">Managing Content Approvals</div>
    @if (Model.ApprovalDefinition == null)
    {
        <div class="well well-small">
            @Html.ContentLink(
                text: "Create Approval Definition",
                contentLink: Model.CurrentPage.ContentLink,
                routeValues: new { task = "createDefinition" },
                htmlAttributes: new { @class = "btn" })
        </div>
    }
```

```
    else
    {
        if (Model.Approval == null)
        {
            <div class="well well-small">
                @Html.ContentLink(
                            text: "Modify Start and Request Approval",
                            contentLink: Model.CurrentPage.ContentLink,
                            routeValues: new { task = "modifyStart" },
                            htmlAttributes: new { @class = "btn" })
            </div>
        }
        else
        {
            <h3>Approval</h3>
            <table class="table table-condensed table-bordered table-condensed">
                <tr>
                    <th>Property</th>
                    <th>Value(s)</th>
                </tr>
                <tr>
                    <td>Started</td>
                    <td>@Model.Approval.Started, By: @Model.Approval.StartedBy,
Status: @Model.Approval.Status</td>
                </tr>
                <tr>
                    <td>Steps</td>
                    <td>@Model.Approval.StepCount, Active:
@Model.Approval.ActiveStepIndex</td>
                </tr>
                @if (Model.Approval.Completed.HasValue)
                {
                    <tr>
                        <td>Completed</td>
                        <td>
                            @Model.Approval.Completed by
@Model.Approval.CompletedBy<br/>
                            <small>@Model.Approval.CompletedComment</small>
                        </td>
                    </tr>
                }
            </table>
        }
        <h3>Approval Definition</h3>
        <table class="table table-condensed table-bordered table-condensed">
            <tr>
                <th>Property</th>
                <th>Value</th>
            </tr>
            <tr>
                <td>ID</td>
                <td>@Model.ApprovalDefinition.ID</td>
            </tr>
            <tr>
                <td>Reference</td>
                <td>@Model.ApprovalDefinition.Reference</td>
            </tr>
            <tr>
                <td>Steps</td>
                <td>@Model.ApprovalDefinition.Steps.Count</td>
            </tr>
            <tr>
```

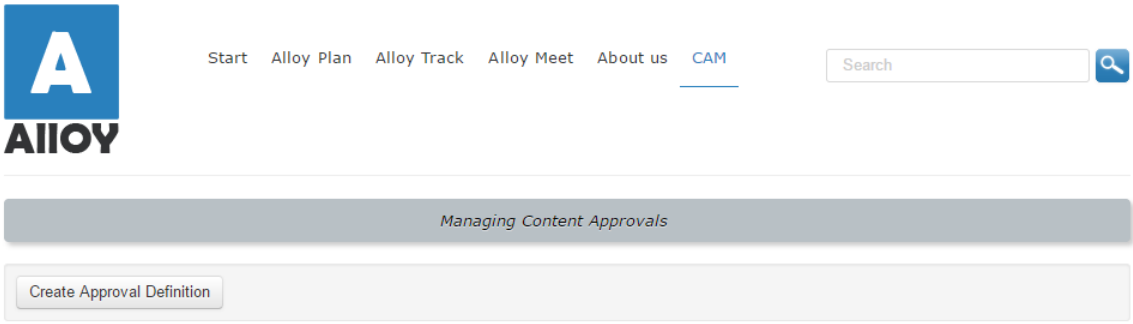
```
            <td>IsEnabled</td>
            <td>@Model.ApprovalDefinition.IsEnabled</td>
        </tr>
        <tr>
            <td>RequireCommentOnApprove</td>
            <td>@Model.ApprovalDefinition.RequireCommentOnApprove</td>
        </tr>
        <tr>
            <td>RequireCommentOnReject</td>
            <td>@Model.ApprovalDefinition.RequireCommentOnReject</td>
        </tr>
    </table>
    <h4>Approval Steps</h4>
    <table class="table table-condensed table-bordered table-condensed">
        <tr>
            <th>Step Name</th>
            <th>Reviewers</th>
            <th></th>
        </tr>
        @for (int index = 0;
            index < Model.ApprovalDefinition.Steps.Count; index++)
        {
            var step = Model.ApprovalDefinition.Steps[index];
            <tr>
                <td>@step.Name</td>
                <td>
                    @foreach (var reviewer in step.Reviewers)
                    {
                        <span class="badge badge-warning">
                        @reviewer.Username</span>
                    }
                </td>
                <td>
                    @foreach (var reviewer in step.Reviewers)
                    {

                        <div class="well well-small">
                            @Html.ContentLink(
        text: "Approve as " + reviewer.Username,
        contentLink: Model.CurrentPage.ContentLink,
        routeValues: new
        {
            task = "processStep",
            stepIndex = index,
            user = reviewer.Username,
            decision = "Approve"
        },
        htmlAttributes: new { @class = "btn btn-success" })

                            @Html.ContentLink(
        text: "Decline as " + reviewer.Username,
        contentLink: Model.CurrentPage.ContentLink,
        routeValues: new
        {
            task = "processStep",
            stepIndex = index,
            user = reviewer.Username,
            decision = "Decline"
        },
        htmlAttributes: new { @class = "btn btn-danger" })
                        </div>
                    }
```

```
                    </td>
                </tr>
            }
        </table>
    }
</div>
```

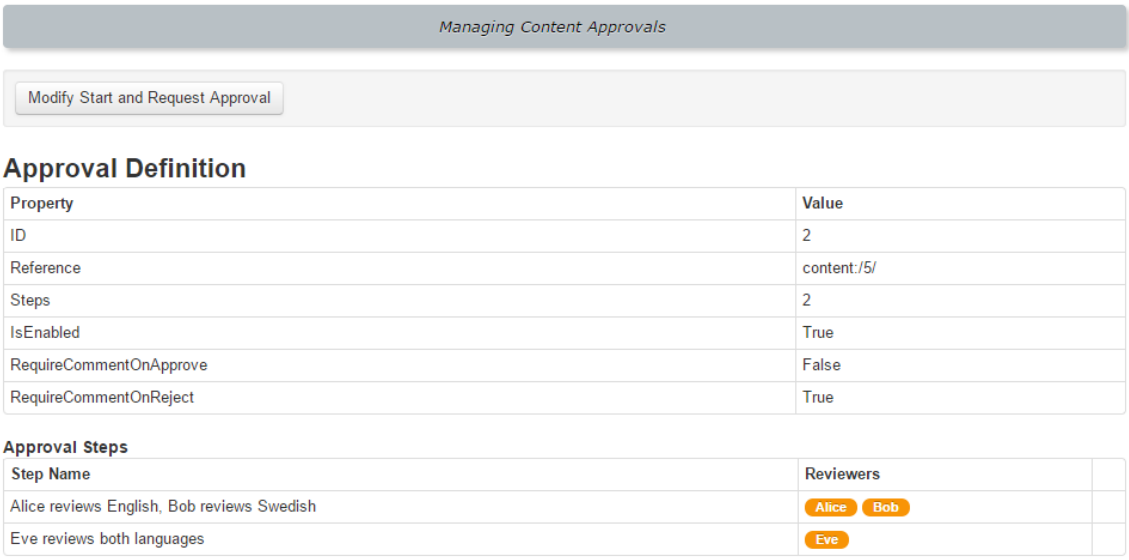
Creating a content approvals sequence

To better understand how the code works, I recommend that you set a breakpoint at the top of the ContentApprovalsManagerPageController's Index method, start the site with debugging, and step through the statements in the Index method one-by-one.

1. Start the site, and log in as **Admin**.
2. Underneath the **Start** page, add a **Content Approvals Manager** page named **CAM**.
3. **Publish** the page.
4. View the site as a visitor.
5. Click **Create Approval Definition**, as shown in the following screenshot:

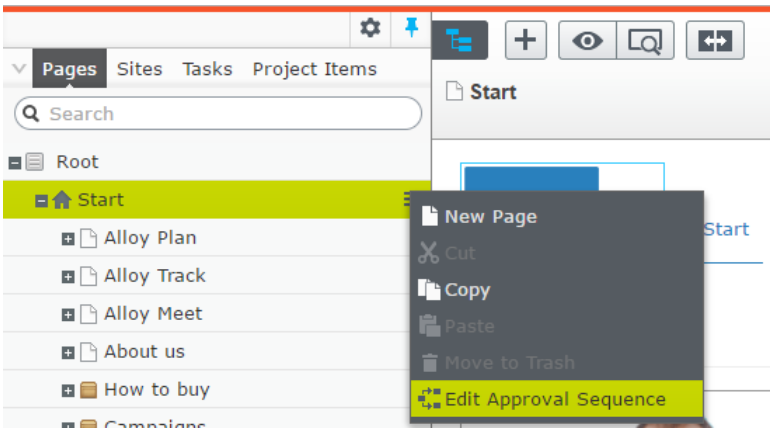


6. Review the approval definition that has been created for the Start page (ID is 5, URI is content:/5/), as shown in the following screenshot:

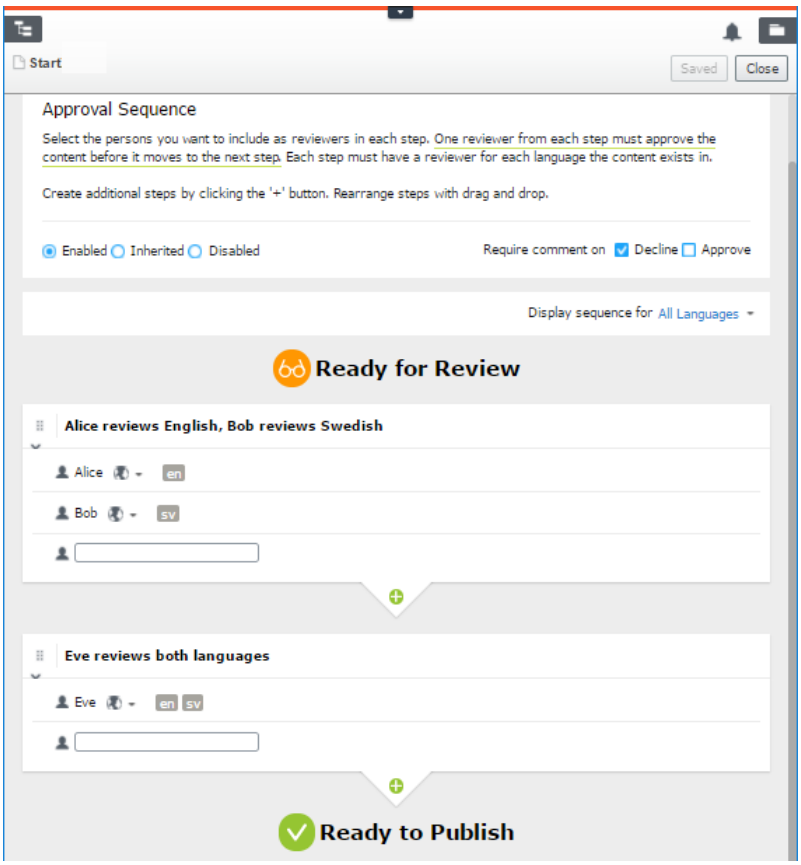


7. Log in as **Admin**.
8. Navigate to **CMS** | **Edit**, and view the **Pages** in the **Navigation** pane.

9. Click the content menu for **Start**, and choose **Edit Approval Sequence**, as shown in the following screenshot, and review the approval sequence to ensure that it has been created correctly:



10. Review the **Approval Sequence** definition, as shown in the following screenshot:



11. Click **Close**.

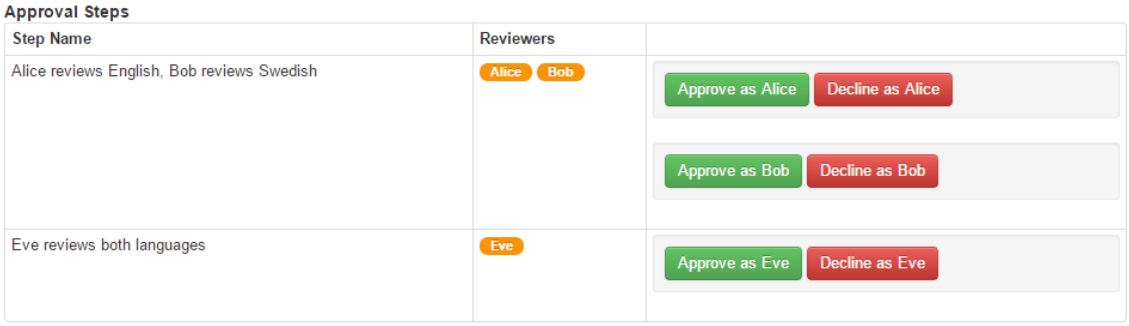
Processing an approval sequence

1. View the site as a visitor.
2. Click **CAM.**, click **Modify Start and Request Approval.**, and note the approval sequence has started, as shown in the following screenshot:

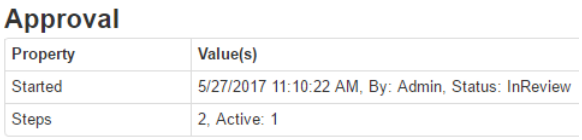
Approval

Property	Value(s)
Started	5/27/2017 11:10:22 AM, By: Admin, Status: InReview
Steps	2, Active: 0

3. Also note the **Approval Steps** have buttons for approving and declining as the reviewers, as shown in the following screenshot:

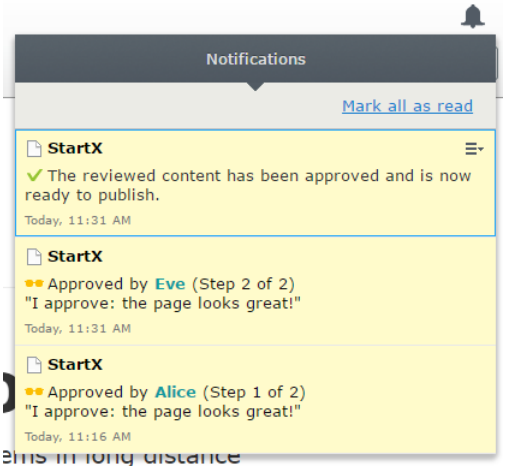


4. Click **Approve as Alice**, and note the approval moves to step 1, as shown in the following screenshot:



5. Click **Approve as Eve**, and note the approval sequence has completed with a **Status** of **Approved**, and Eve's comment is used as a completion comment, as shown in the following screenshot:

6. Log in as **Admin**.
7. Navigate to **CMS** | **Edit**, and open the **Start** page.
8. Click the bell icon for Admin's notifications, and note the three notifications messages, as shown in the following screenshot:



9. Click **Publish Changes**.

Avoid or check for exceptions

Once a content approval step is approved, it cannot be approved again, or an exception is thrown.

Your challenge is to extend the B2 exercise to either avoid the possibility of a reviewer making multiple decisions for a step, or at least handle any exceptions when they occur.

Implement the approval sequence for Swedish

When processing an approval programmatically, it is up to the developer to check for multiple languages. In the example exercises above, the code only creates and retrieves an approval workflow for the Master Language (English), so only Allice and Eve are needed to complete the workflow.

Your challenge is to extend the B2 exercise to create and modify a Swedish language branch of the Start page, and allow that approval to be processed too.

Exercise B3 – Implementing notifications

In this exercise, you will add functionality to view all notifications that have been sent or received by users, and to send notifications over a custom channel.

Prerequisites: complete Exercise 0.1.

Creating a notifications page type and template

1. In **Solution Explorer**, open **Global.cs**.
2. In the **Global** class, add a readonly string constant for **NotificationChannel**, as shown in the following code:

```
public class Global
{
    public static readonly string NotificationChannel = "epi.alloy.messages";
```

3. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **NotificationsPage.cs**.
4. Modify its statements, as shown in the following code, and note the following:
    - **NotificationsPage** is included as an allowed child on **StartPage**.
    - The class has an ignored property, so we don't need to define a custom view model with that property, but its value will be cached, so we must be careful to always clear it in the controller.

```
using EPiServer.DataAnnotations;
using EPiServer.Notification;
using System.Collections.Generic;

namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(DisplayName = "Notifications",
        GroupName = Global.GroupNames.Specialized,
        Description = "Use to manage user notifications.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class NotificationsPage : SitePageData
    {
        [Ignore]
        public virtual Dictionary<string,
            PagedUserNotificationMessageResult> Notifications { get; set; }
    }
}
```

5. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **NotificationsPageController**.
6. Modify its contents, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Notification;
using EPiServer.Shell.Security;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class NotificationsPageController : PageControllerBase<NotificationsPage>
```

```
{
    // some services that we need
    private readonly INotifier notifier;
    private readonly IUserNotificationRepository userNotificationRepository;
    private readonly QueryableNotificationUserService
        queryableNotificationUserService;
    private readonly UIUserProvider userProvider;

    public NotificationsPageController(INotifier notifier,
        IUserNotificationRepository userNotificationRepository,
        QueryableNotificationUserService queryableNotificationUserService,
        UIUserProvider userProvider)
    {
        this.notifier = notifier;
        this.userNotificationRepository = userNotificationRepository;
        this.queryableNotificationUserService = queryableNotificationUserService;
        this.userProvider = userProvider;
    }

    // Notifications API is asynchronous
    // channel parameter can be used to filter messages
    public async Task<ActionResult> Index(
        NotificationsPage currentPage, string channel)
    {
        // always reset the Ignore property to an empty dictionary
        currentPage.Notifications =
            new Dictionary<string, PagedUserNotificationMessageResult>();

        // get a list of the first 30 registered users
        IEnumerable<IUIUser> users = userProvider.GetAllUsers(
            pageIndex: 0, pageSize: 30, totalRecords: out int totalRecords);

        foreach (IUIUser user in users)
        {
            // get an object that represents notifications for each user
            INotificationUser notificationUser = await
                queryableNotificationUserService.GetAsync(user.Username);

            // build a query that includes read, unread, sent, and unsent messages
            var query = new UserNotificationsQuery
            {
                Read = null, // include read and unread
                Sent = null, // include sent and unsent
                User = notificationUser
            };

            // if a channel name is set, use it to filter the notifications
            if (!string.IsNullOrWhiteSpace(channel))
            {
                ViewData["channel"] = channel;
                query.ChannelName = channel;
            }

            // execute the query
            var result = await userNotificationRepository
                .GetUserNotificationsAsync(
                    query, startIndex: 0, maxRows: 20);

            // store the query results for the user
            currentPage.Notifications.Add(user.Username, result);
        }
```

```
            return View(PageViewModel.Create(currentPage));
        }
    }
}
```

7. In **Solution Explorer**, right-click **~/Views**, add a folder named **NotificationsPage**.
8. In **~\Views\NotificationsPage**, add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
9. Modify its contents, as shown in the following markup:

```
@using EPiServer.Notification
@using AlloyAdvanced.Models.Pages
@model PageViewModel<NotificationsPage>
<div class="row">
    <div class="span12">
        <input type="button" value="All Channels" />
        <input type="button" value="Channel:
@AlloyAdvanced.Global.NotificationChannel" />
    </div>
</div>
@foreach (KeyValuePair<string, PagedUserNotificationMessageResult> item in
Model.CurrentPage.Notifications)
{
    <div class="row">
        <div class="span12">
            <h3>@item.Key notifications</h3>
            <table class="table table-bordered table-striped table-condensed">
                <tr>
                    <th>Sender</th>
                    <th>Recipient</th>
                    <th>TypeName</th>
                    <th>Subject</th>
                    <th>Content</th>
                    <th>Posted</th>
                    <th>Send At</th>
                    <th>Sent</th>
                    <th>Read</th>
                </tr>
                @foreach (UserNotificationMessage m in item.Value.PagedResult)
                {
                    <tr>
                        <td>@m.Sender.UserName</td>
                        <td>@m.Recipient.UserName</td>
                        <td>@m.TypeName</td>
                        <td>@m.Subject</td>
                        <td>@m.Content</td>
                        <td>@m.Posted</td>
                        <td>@m.SendAt</td>
                        <td>@m.Sent</td>
                        <td>@m.Read</td>
                    </tr>
                }
            </table>
        </div>
    </div>
}
```

Creating a send notification page type and template

1. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **SendNotificationPage.cs**.

2. Modify its statements, as shown in the following code:

```
using EPiServer.DataAnnotations;

namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(DisplayName = "Send Notification",
        GroupName = Global.GroupNames.Specialized,
        Description = "Use to send a notification to another Episerver user.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class SendNotificationPage : SitePageData
    {
    }
}
```

3. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **SendNotificationPageController**.

4. Modify its contents, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Notification;
using EPiServer.Shell.Security;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class SendNotificationPageController :
        PageControllerBase<SendNotificationPage>
    {
        // some services that we need
        private readonly INotifier notifier;
        private readonly IUserNotificationRepository userNotificationRepository;
        private readonly QueryableNotificationUserService
queryableNotificationUserService;
        private readonly UIUserProvider userProvider;

        public SendNotificationPageController(INotifier notifier,
            IUserNotificationRepository userNotificationRepository,
            QueryableNotificationUserService queryableNotificationUserService,
            UIUserProvider userProvider)
        {
            this.notifier = notifier;
            this.userNotificationRepository = userNotificationRepository;
            this.queryableNotificationUserService = queryableNotificationUserService;
            this.userProvider = userProvider;
        }

        public ActionResult Index(SendNotificationPage currentPage)
        {
            // get a list of the first 30 registered users
            IEnumerable<IUIUser> users = userProvider.GetAllUsers(
                pageIndex: 0, pageSize: 30, totalRecords: out int totalRecords);

            // store the user names in ViewData so they can be used in a dropdown
listbox
            ViewData["users"] = users.OrderBy(user => user.Username)
                .Select(user => user.Username).ToArray();
```

```
            return View(PageViewModel.Create(currentPage));
        }

        [HttpPost]
        // Notifications API is asynchronous
        // parameters are for fields in the posted form
        public async Task<ActionResult> Index(SendNotificationPage currentPage,
            string from, string to, string subject, string content, string uri,
            string when, DateTime? whenDateTime, int? whenDelay)
        {
            IEnumerable<IUIUser> users = userProvider.GetAllUsers(
                pageIndex: 0, pageSize: 30, totalRecords: out int totalRecords);

            INotificationUser sender = await
                queryableNotificationUserService.GetAsync(from);

            var receivers = new List<INotificationUser>();
            string[] usernames = to.Split(';');
            foreach (var username in usernames)
            {
                INotificationUser user = await
                    queryableNotificationUserService.GetAsync(username);
                receivers.Add(user);
            }

            NotificationMessage message;

            if (whenDelay.HasValue)
            {
                message = new DelayedNotificationMessage();
                (message as DelayedNotificationMessage).Delay =
TimeSpan.FromMinutes(whenDelay.Value);
            }
            else if (whenDateTime.HasValue)
            {
                message = new ScheduledNotificationMessage();
                (message as ScheduledNotificationMessage).SendAt = whenDateTime.Value;
            }
            else
            {
                message = new NotificationMessage();
            }

            if (!string.IsNullOrWhiteSpace(uri)) message.Category = new Uri(uri);
            message.ChannelName = Global.NotificationChannel;
            message.Content = content;
            message.Subject = subject;
            message.Recipients = receivers;
            message.Sender = sender;
            message.TypeName = Global.NotificationChannel;

            await notifier.PostNotificationAsync(message);

            ViewData["users"] = users.OrderBy(user => user.Username)
                .Select(user => user.Username).ToArray();
            ViewData["messageSent"] = "Your message was sent successfully.";

            return View(PageViewModel.Create(currentPage));
        }
    }
}
```

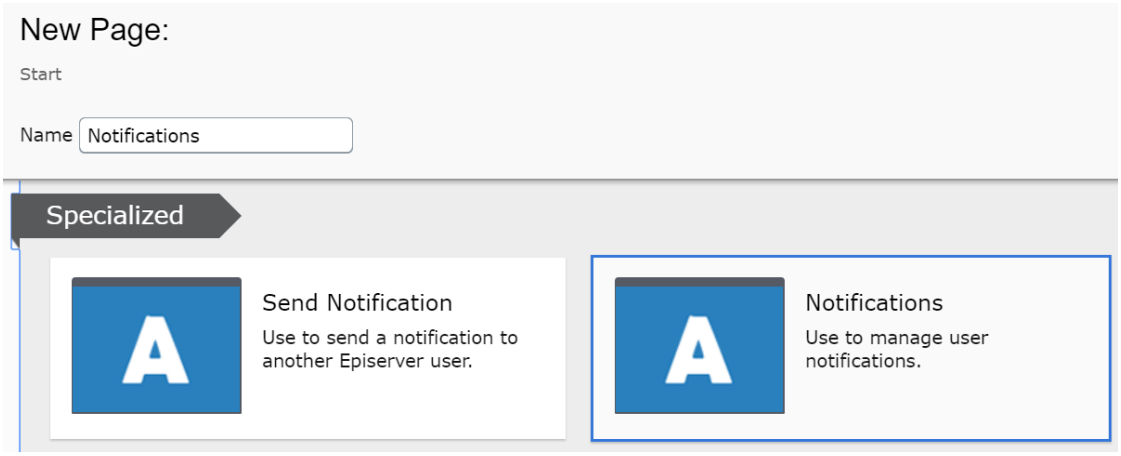
5. In **Solution Explorer**, right-click **~/Views**, add a folder named **SendNotificationPage**.
6. In **~\Views\SendNotificationPage**, add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
7. Modify its contents, as shown in the following markup:

```
@using AlloyAdvanced.Models.Pages
@model PageViewModel<SendNotificationPage>
<div class="row">
    <div class="span12">
        @if (ViewData["messageSent"] != null)
        {
            <div class="alert alert-success">
                @ViewData["messageSent"]
            </div>
        }
        <form action="@Url.ContentUrl(Model.CurrentPage.ContentLink)" method="post"
class="form-horizontal">
            <h3>Send notification on @AlloyAdvanced.Global.NotificationChannel
channel</h3>
            <div>
                From: <select name="from">
                    @foreach (string name in ViewData["users"] as string[])
                    {
                        <option value="@name">@name</option>
                    }
                </select>
            </div>
            <div>
                To: <input name="to" />
                <small>Separate user names with semi-colons:
                    @foreach (string name in ViewData["users"] as string[])
                    {
                        <span class="label label-info">@name</span>
                    }
                </small>
            </div>
            <div>Subject: <input name="subject" /></div>
            <div>Content: <textarea name="content" rows="4"
cols="30"></textarea></div>
            <div>URI: <input name="uri" /> <small>For example, /en/about-
us</small></div>
            <div><input type="radio" name="when" checked="checked" />Immediately</div>
            <div><input type="radio" name="when" />At <input name="whenDateTime"
/></div>
            <div><input type="radio" name="when" />In <input name="whenDelay" />
minutes</div>
            <div><input type="submit" value="Send" /></div>
        </form>
    </div>
</div>
```

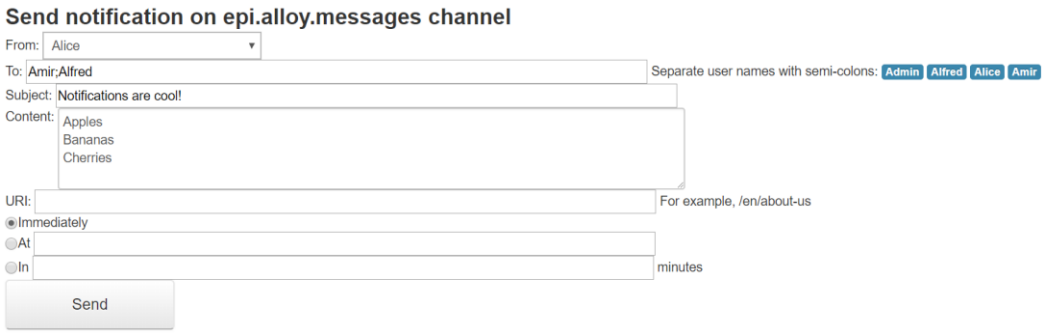
Testing the notifications functionality

1. Start the **AlloyAdvanced** site, and log in as **Admin**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Create User**, and create a few users, all members of the **WebAdmins** group, perhaps named **Alice**, **Amir**, and **Alfred**. Remember to make them all active.
3. Create a new page under **Start**.

4. Choose the **Notifications** page type, and name the new page **Notifications**, as shown in the following screenshot:



5. Publish the **Notifications** page.
6. Create a new page under **Start**.
7. Choose the **Send Notification** page type, and name the new page **Send Notification**.
8. Publish the **Send Notification** page.
9. Use the Global menu to switch to Live view.
10. Navigate to **Notifications** page, and note every user has a table of notification messages.
11. Navigate to **Send Notification** page, and complete the form to send a notification from **Alice** to **Amir** and **Alfred**, as shown in the following screenshot:



12. Set the message to be sent **Immediately**.
13. Click **Send**, and note the success message.
14. Navigate to **Notifications** page, and note that Amir and Alfred have both been sent the notification message but neither have read it, as shown in the populated **Sent** columns and empty **Read** columns in the following screenshot:

Alfred notifications

Sender	Recipient	TypeName	Subject	Content	Posted	Send At	Sent	Read
Alice	Alfred	epi.alloy.messages	Notifications are cool!	Apples Bananas Cherries	12/19/2017 5:31:29 PM		12/19/2017 5:31:29 PM	

Alice notifications

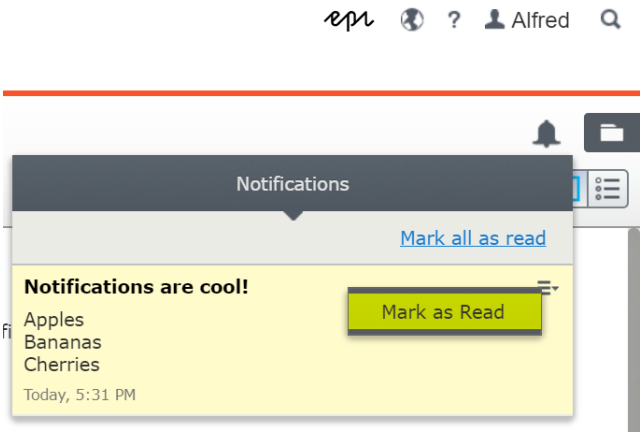
Sender	Recipient	TypeName	Subject	Content	Posted	Send At	Sent	Read

Amir notifications

Sender	Recipient	TypeName	Subject	Content	Posted	Send At	Sent	Read
Alice	Amir	epi.alloy.messages	Notifications are cool!	Apples Bananas Cherries	12/19/2017 5:31:29 PM		12/19/2017 5:31:29 PM	

15. Log out as Admin.

16. Log in as **Alfred**, navigate to Edit view, click the notification bell icon, and use the context menu to mark the message as read, as shown in the following screenshot:



17. Navigate to **Notifications** page, and note that **Alfred** has now read the notification, as shown in the **Read** column in the following screenshot:

Alfred notifications

Sender	Recipient	TypeName	Subject	Content	Posted	Send At	Sent	Read
Alice	Alfred	epi.alloy.messages	Notifications are cool!	Apples Bananas Cherries	12/19/2017 5:31:29 PM		12/19/2017 5:31:29 PM	12/19/2017 5:38:24 PM

18. Close the browser.

Exercise B4 – Implementing a commenting solution

In this exercise, you will review existing Alloy functionality, review the functional specification of a commenting feature, and then design and implement a commenting solution.

In the real world, you could implement commenting with an external system like Discus. The point of this exercise is to use Episerver functionality to implement a solution, so you learn about Episerver APIs.

Prerequisites: complete Exercise 0.1.

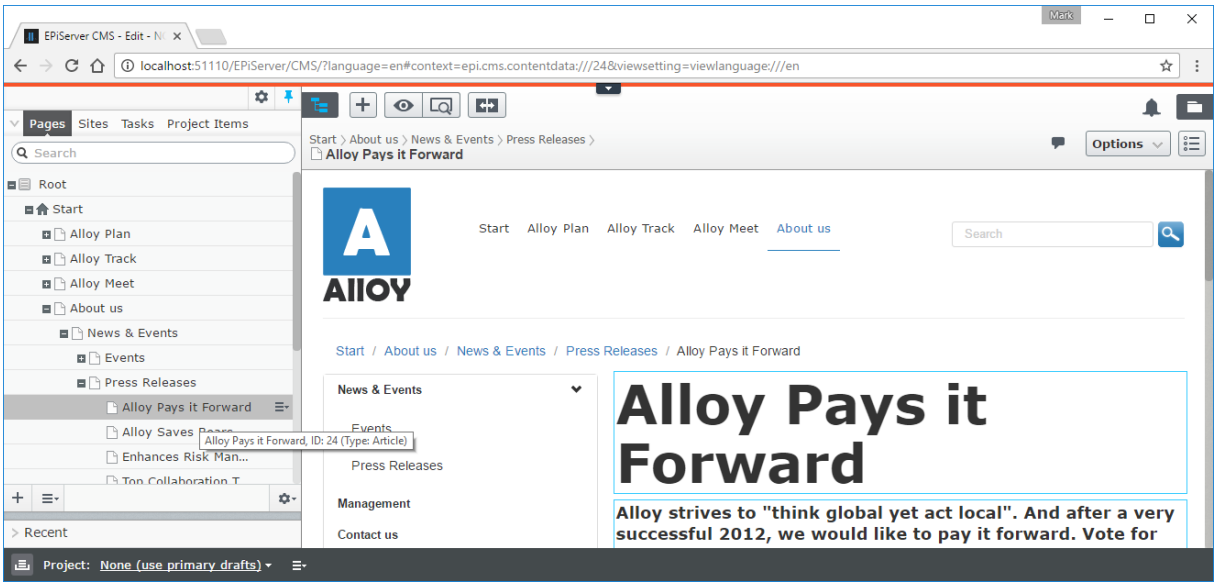
Work on your own or with the person sitting next to you to design a solution. You will design and implement a solution to allow visitors to add comments to an Alloy site page, including:

- Set up the commenting functionality.
- Add visitor comments on a page.
- Handle access rights for comments.
- Filter and sort the comments.
- Report and hide comments.

Review the existing site

First, let's look at the existing Alloy site and a typical page that a visitor might want to comment on.

1. Open the solution with the **AlloyAdvanced** project.
2. Start the site.
3. Navigate to **About us** | **News & Events** | **Press Releases** | **Alloy Pays it Forward**.
4. Log in and switch to Edit view for the article.
5. Note that hovering over the page in the **Pages** tab in the **Navigation** pane shows the page's name, ID, and type, as shown in the following screenshot. **Alloy Pays it Forward** is an example of an **ArticlePage**. All the pages under the **Press Releases** page are instances of **ArticlePage**.



6. Close the browser.

Review the view for an article page

Every page type in Alloy has its own view. Let's look at the view for an **ArticlePage**.

1. In **Solution Explorer**, open **~/Views/ArticlePage/Index.chstml**. Note that An **ArticlePage** outputs its **PageName** as an `<h1>` heading, its **MetaDescription** as a styled paragraph, its **MainBody** in a `<div>` element, and finally its **MainContentArea**.

```
@using AlloyAdvanced
@model PageViewModel<ArticlePage>

@{ Layout = "~/Views/Shared/Layouts/_LeftNavigation.cshtml"; }

<h1 @Html.EditAttributes(x => x.CurrentPage.PageName)>@Model.CurrentPage.PageName</h1>
<p class="introduction" @Html.EditAttributes(
    x => x.CurrentPage.MetaDescription)>@Model.CurrentPage.MetaDescription</p>
<div class="row">
    <div class="span8 clearfix" @Html.EditAttributes(x => x.CurrentPage.MainBody)>
        @Html.DisplayFor(m => m.CurrentPage.MainBody)
    </div>
</div>
@Html.PropertyFor(x => x.CurrentPage.MainContentArea, new { CssClass = "row",
    Tag = Global.ContentAreaTags.TwoThirdsWidth })
```

2. In **Solution Explorer**, open the **~/Controllers/DefaultPageController.cs** file and set a break point in the **Index** method. Note:
    - **DefaultPageController** is inherited by any page type that derives from **SitePageData** (for the Alloy site, this is ALL pages) so unless a page type has a more specific controller, it will use this one.
    - It uses a formatted string to forward the request to different views based on the type name. It is common to design sites with shared controllers but separate views like this.

```
[TemplateDescriptor(Inherited = true)]
public class DefaultPageController : PageControllerBase<SitePageData>
{
    public ViewResult Index(SitePageData currentPage)
    {
        var model = CreateModel(currentPage);
        return View(string.Format("~/Views/{0}/Index.cshtml",
            currentPage.GetOriginalType().Name), model);
    }
```

3. Start the site.
4. Navigate to **About us**. Note that **About us** is a **StandardPage** and it uses the **DefaultPageController**.
5. Navigate to one of the articles in the **Press Releases** section underneath **News & Events**. Note that press releases are **ArticlePage** instances and they also use the **DefaultPageController**.

Review the required functionality

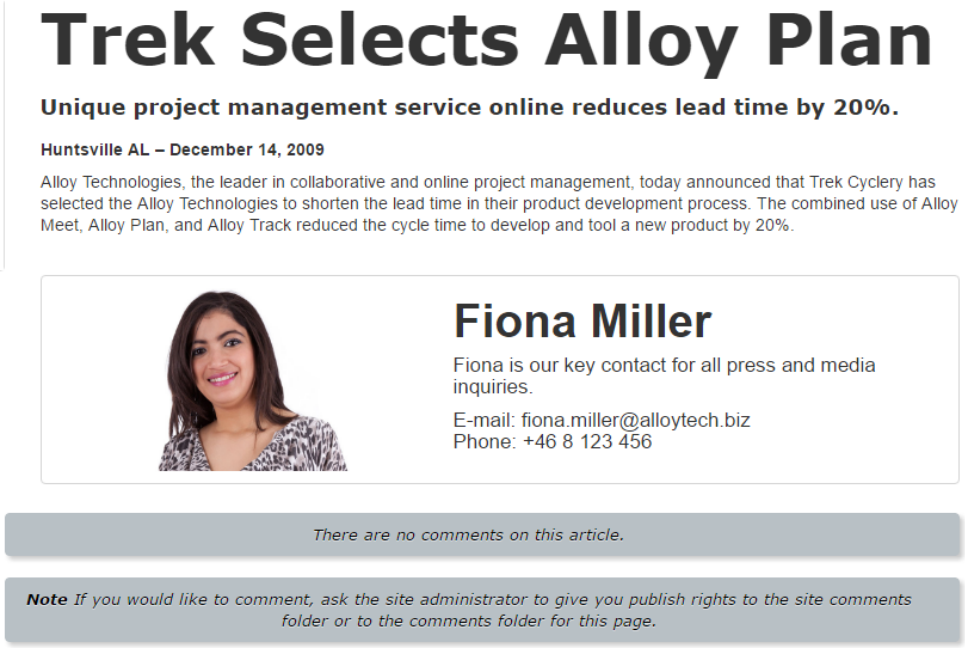
You need to allow visitors to add comments to ArticlePage instances, although it would be best to design a solution in such a way that other page types could have comments too, especially since the Alloy site already has a shared controller for most page types.

This is what the functionality should be. Note:

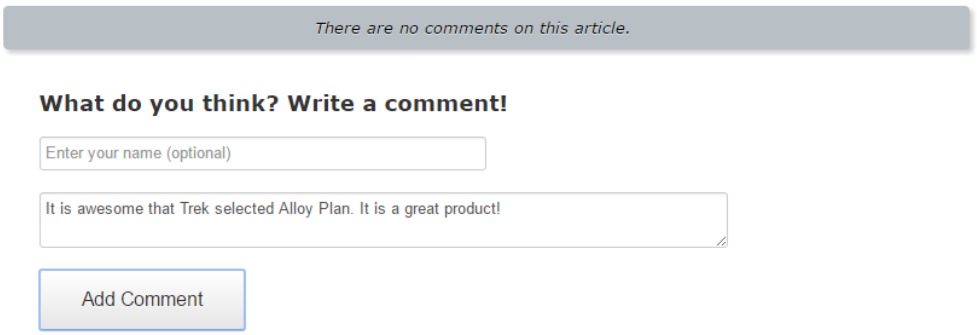
- On a page with comment functionality, but without any comments, the view will display a message saying the page has no comments, and either a form to add a comment or a message to the user explaining how they can request comment functionality.
- If an anonymous visitor goes to a page with comments, they will see the comments but they will not be able to add comments.
- Previous comments are shown above the form, including a date/time stamp, and a link to “report the comment”.
- Comments can be sorted ascending/descending on the date when the commented was added.

- Comments can be reported which immediately expires the comment so that it is hidden until an editor or administrator can review it and either republish the comment or delete it permanently.
- If a logged-in visitor goes to a page, their access rights are checked. If they have rights to either add comments every in the site or add comments to the current page, then they can see the comment form to add a comment.
- If a commenter enters an empty name, it should be set to “Anonymous”.

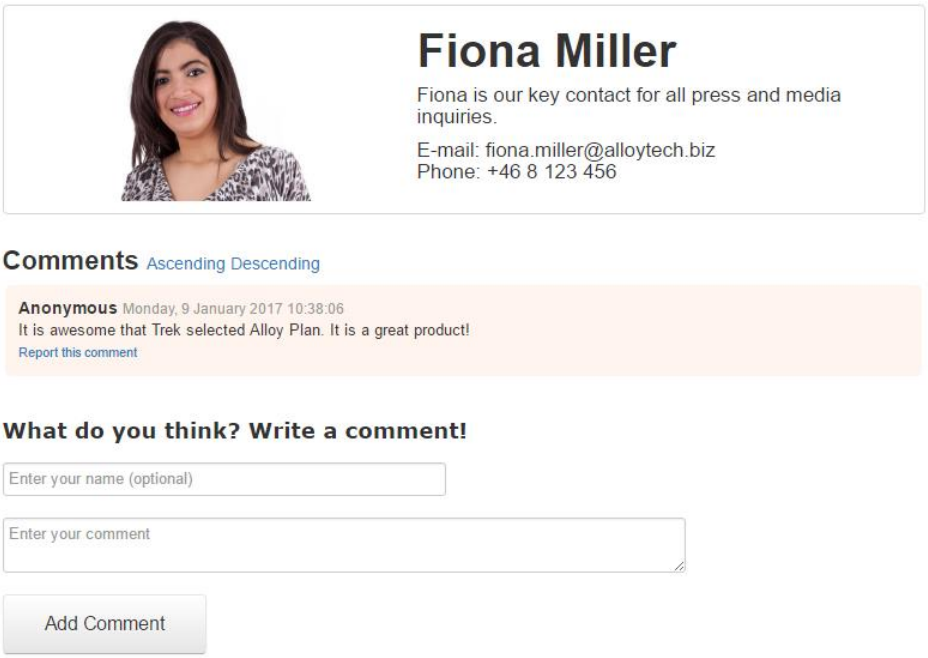
Anonymous visitor viewing a page without comments



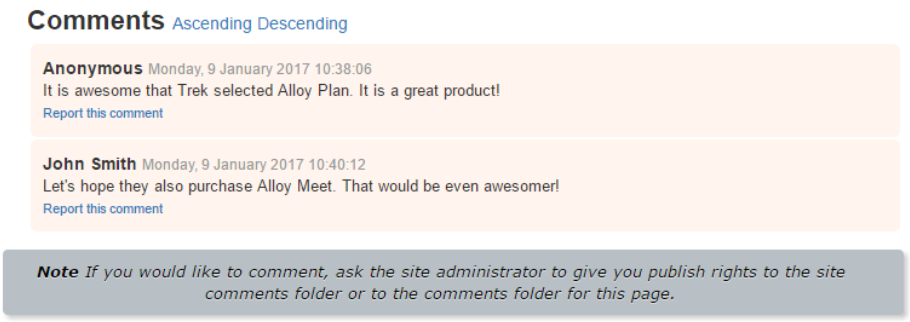
Logged-in visitor with Publish rights to start page’s Comments folder, viewing a page with no comments



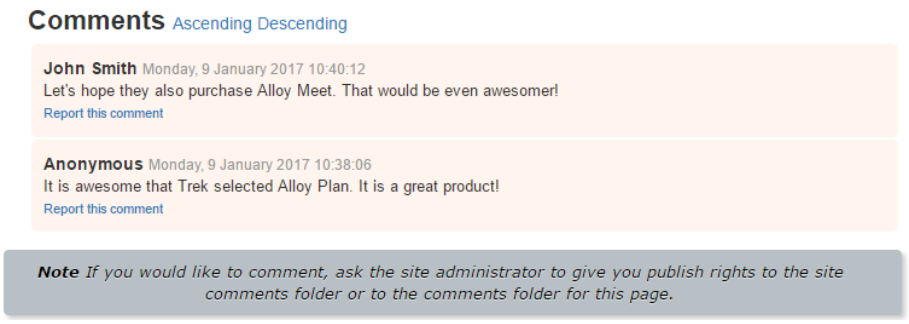
After adding the comment



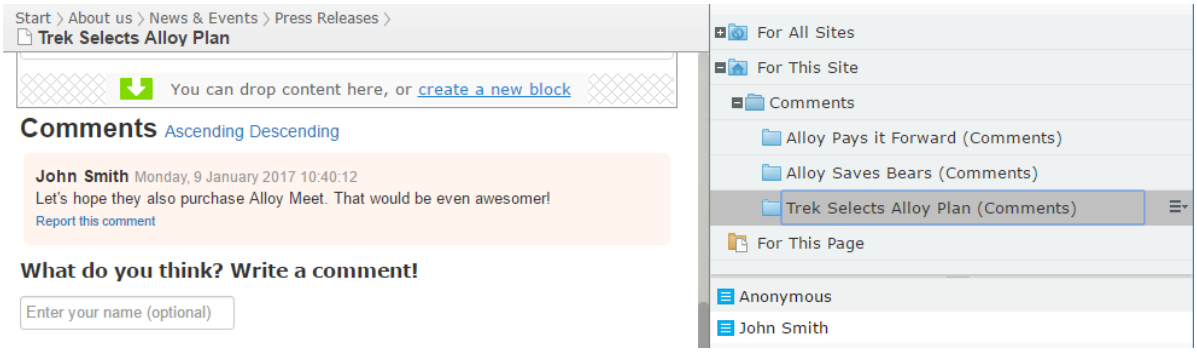
Anonymous visitor viewing comments in default (ascending) order



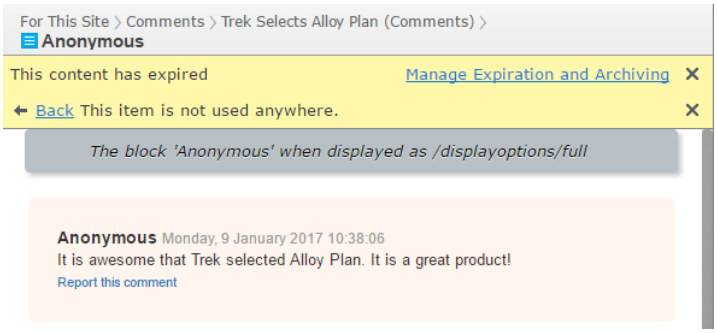
Anonymous visitor viewing comments in descending order



Editors can see expired “Report this comment” comments in the Blocks pane, like Anonymous below



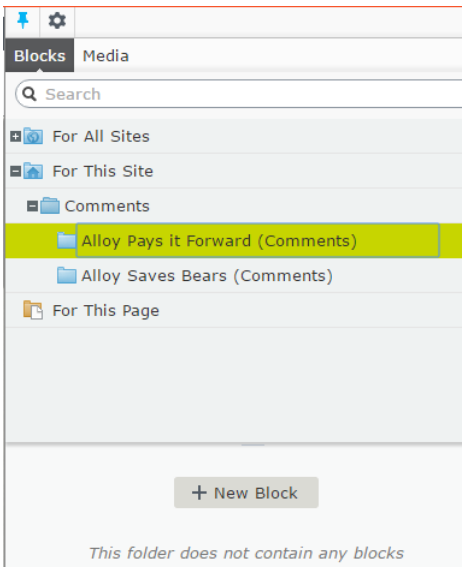
Double-clicking the comment block allows the editor to “un-expire” the comment



Reviewing a potential solution

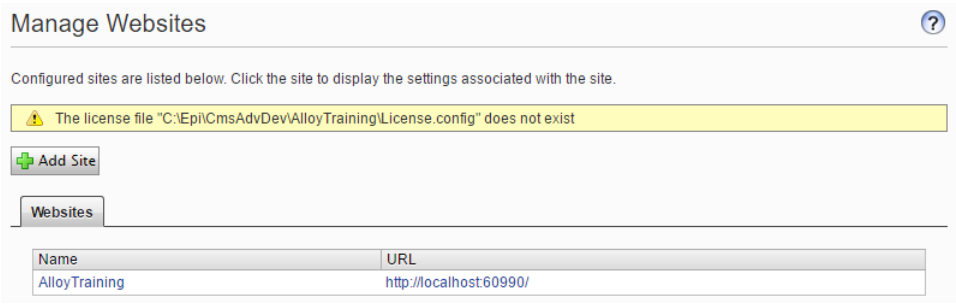
Each article could have multiple comments. This data must be stored somewhere. Since we want to be able to “expire” a comment to hide it and then allow an administrator or editor to republish it, the comments would be best stored as content in the CMS, instead of in an external database.

We could store comments as pages underneath the page they belong to. But comments will never be shown individually, as a full page, so they should be implemented as a type of block. Blocks can be stored in folders, so having one folder per page would be a good way of grouping the comments for that page. We would need a folder to group all those folders, and that “Comments” folder could also be used to assign rights add comments to the whole site, as shown in the following screenshot:

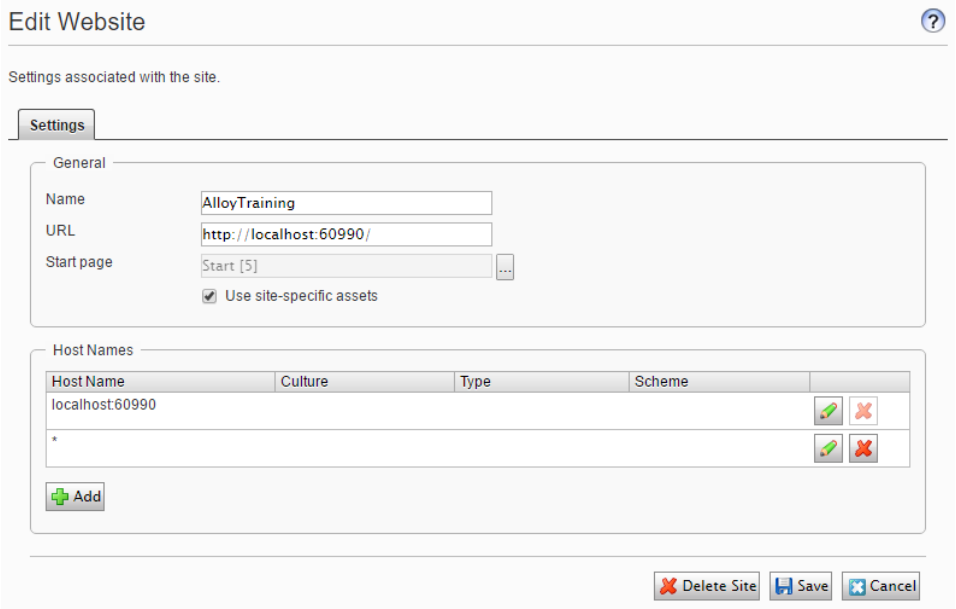


Enabling the For This Site folder

1. Navigate to **CMS** | **Admin** | **Config** | **Manage Websites**, and click on the **AlloyAdvanced** site, as shown in the following screenshot:



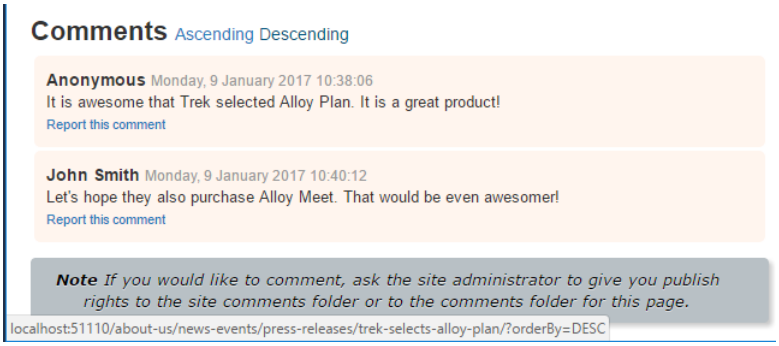
2. Check the **Use site-specific assets** box, and click **Save**, as shown in the following screenshot:



Each page will have a property to reference its comments folder

Each page needs to know where to find its comments, so we could add a property to the **SitePageData** page type that will point to its block folder for comments.

Allowing the visitor to choose sort order can be achieved by passing a query string value



Hovering over the Report this comment link shows how the id is passed to the controller

Files for the commenting solution

These are the files that need changes:

- **~/Views/ArticlePage/Index.cshtml**: the view for an article with a list of its comments and a form to add a new comment (if the user has rights to do so).
- **~/Controllers/DefaultPageController.cs**: the shared controller that adds or reports a comment in the CMS.
- **~/Models/Pages/SitePageData.cs**: the base type for pages with a new property to store a reference to the page's comments folder.
- **~/Models/Pages/ArticlePage.cs**: article page type must implement **IContentWithComment**.

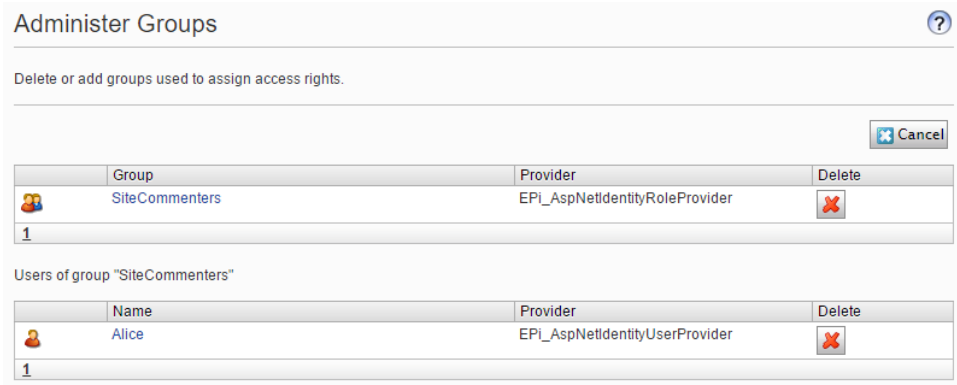
These are the files that need to be created:

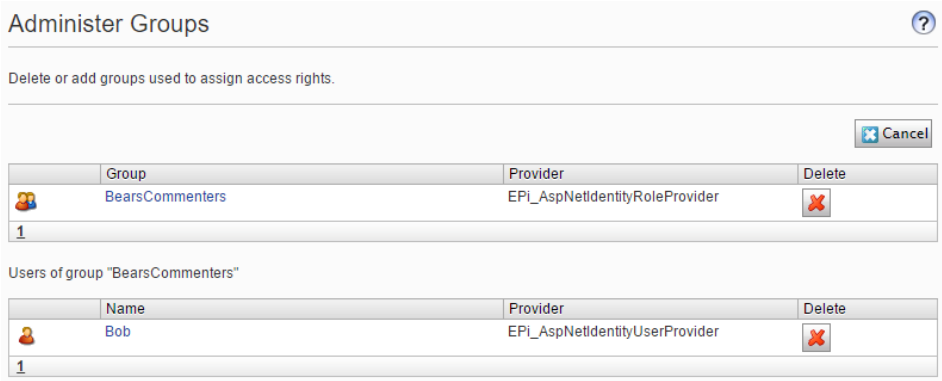
- **~/Models/Blocks/CommentBlock.cs**: the content type for storing a single comment.
- **~/Models/ViewModels/CommentsPageViewModel.cs**: the view model that contains a list of comments for any page that has them, and supporting properties such as ThisPageHasAtLeastOneComment and CurrentUserCanAddComments.
- **~/Models/IContentWithComments.cs**: the interface applied to a page type to indicate that it has commenting functionality.

Testing Access Rights

When you have implemented your solution, and are ready to test the access rights, then create two groups, each with one user in them, as shown in the screenshots below.

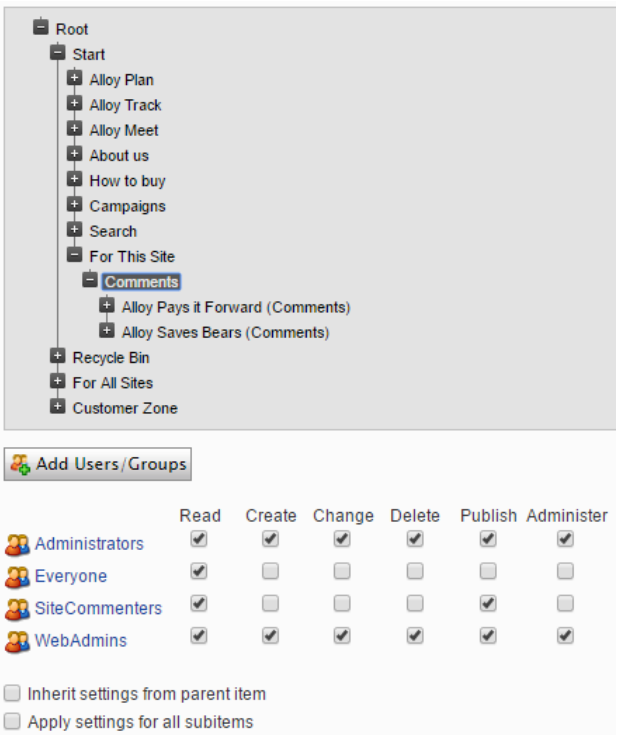
- **SiteCommenters** group contains **Alice**.
- **BearCommenters** group contains **Bob**.



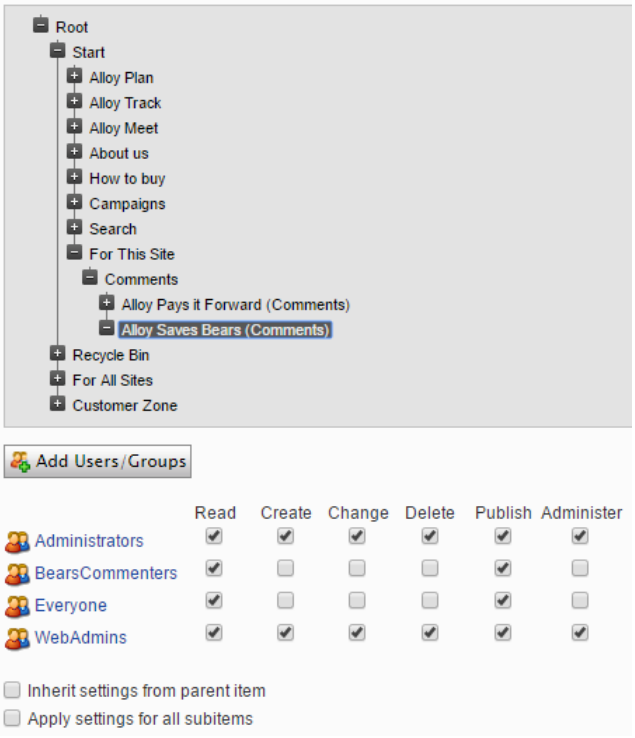


Once you have implemented your automatic set up for commenting functionality, you can run the site to create content folders for article pages.

Set access rights for the **SiteCommenters** group to **Read** and **Publish** to the **Comments** folder, as shown in the screenshots below.



Set access rights for the **BearCommenters** group to **Read** and **Publish** to the **Alloy Saves Bears** folder, as shown in the screenshots below. Note that if you want anonymous visitors to be able to add comments, do the same for the **Everyone** group.



Log in as **Alice** and then **Bob**. Alice should be able to add a comment to any press release. Bob should only be allowed to add comments to the **Alloy Saves Bears** press release.

You will find source code solutions for all the files in B2 exercise, but note that this solution has not been properly refactored or fully tested for bugs.

Module C – Integrating Data and Forms

Goal

The overall goal of the exercises in this module is learn how to use the Dynamic Data Store (DDS), how to implement a partial router to pull data from an external system and render it as if it was content, how to gather data using Episerver Forms, and how to import data using a scheduled job.

You will:

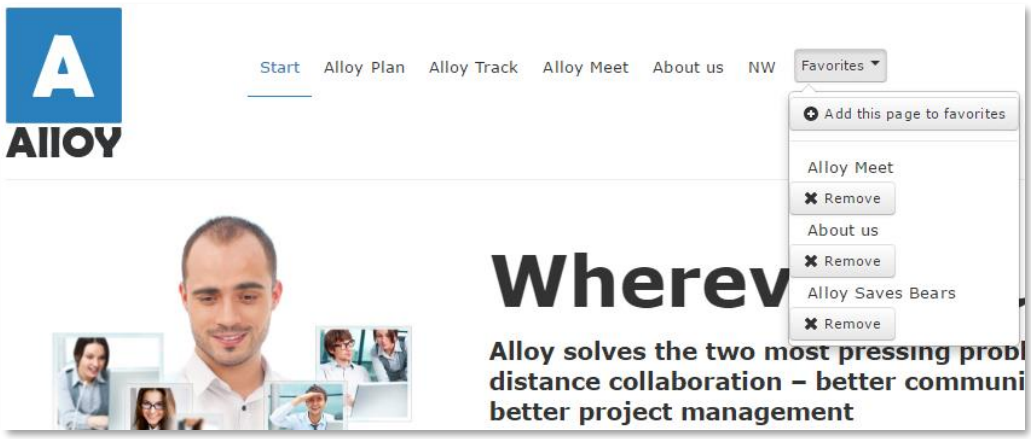
- Implement favorite page functionality by using DDS.
- Integrate an external database with a partial router.
- Gather data using Episerver Forms.
- Import external data using a scheduled job.

Exercise C1 – Implementing favorite pages with Dynamic Data Store

In this exercise, you will implement favorite page functionality.

Prerequisites: complete Exercise 0.1.

A **Favorites** menu will be added to the end of the navigation menu that allows a logged in user to add the current page to a list of favorites and to remove existing favorites, as shown in the following screenshot:



Defining an entity class and repository for DDS

To store an entity in Dynamic Data Store, you must create a class that implements `IDynamicData`.

1. Open the solution with the **AlloyAdvanced** project.
2. Inside **~/Models**, add a new folder named **DDS**.
3. Inside **~/Models/DDS**, add a new class named **Favorite** with the following code:

```
using EPiServer.Data.Dynamic;
using System;
using EPiServer.Data;
using EPiServer.Core;

namespace AlloyAdvanced.Models.DDS
{
    public class Favorite : IDynamicData
    {
        public Identity Id { get; set; }
```

```
        public string UserName { get; set; }
        public ContentReference FavoriteContentReference { get; set; }

        public Favorite()
        {
            Id = Identity.NewIdentity(Guid.NewGuid());
            UserName = string.Empty;
            FavoriteContentReference = ContentReference.EmptyReference;
        }

        public Favorite(ContentReference contentReferenceToAdd,
            string userName) : this() // calls the default constructor first
        {
            UserName = userName;
            FavoriteContentReference = contentReferenceToAdd;
        }
    }
}
```

4. Inside **~/Models/DDS**, add a new class named **FavoriteRepository** with the following code:

```
using EPiServer.Core;
using EPiServer.Data.Dynamic;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AlloyAdvanced.Models.DDS
{
    public class FavoriteRepository
    {
        private static DynamicDataStore store
        {
            get
            {
                // creates or gets reference to the store named Favorites
                return DynamicDataStoreFactory.Instance
                    .CreateStore("Favorites", typeof(Favorite));
            }
        }

        public static void Save(Favorite fav)
        {
            if (string.IsNullOrWhiteSpace(fav.UserName))
            {
                throw new NullReferenceException(
                    "Unable to add favorite without user name");
            }
            store.Save(fav);
        }

        public static List<Favorite> GetFavorites(string userName)
        {
            return store.Items<Favorite>()
                .Where(fav => fav.UserName == userName)
                .ToList();
        }

        public static Favorite GetFavorite(
            ContentReference contentReference, string userName)
        {
            return store.Items<Favorite>()
                .Where(fav => (fav.UserName == userName) &&
```

```
                    (fav.FavoriteContentReference == contentReference))
                .FirstOrDefault();
        }

        public static void Delete(Favorite fav)
        {
            store.Delete(fav.Id);
        }
    }
}
```

Defining a content template for favorites

1. Inside **~/Models/ViewModels**, add a new class named **FavoriteViewModel** with the following code:

```
using AlloyAdvanced.Models.DDS;
using EPiServer.Core;
using System.Collections.Generic;

namespace AlloyAdvanced.Models.ViewModels
{
    public class FavoriteViewModel
    {
        public List<Favorite> Favorites { get; set; }
        public ContentReference CurrentPageContentReference { get; set; }
    }
}
```

2. Inside **~/Controllers**, right-click, choose **Add** | **Controller**, and choose the scaffold template named **MVC 5 Controller – Empty** to add a controller named **FavoritesController**.

3. Modify the statements in the controller class, as shown in the following code:

```
using AlloyAdvanced.Models.DDS;
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Core;
using EPiServer.Security;
using EPiServer.Web.Routing;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class FavoritesController : Controller
    {
        private readonly UrlResolver urlResolver;

        public FavoritesController(UrlResolver urlResolver)
        {
            this.urlResolver = urlResolver;
        }

        public ActionResult Manager(SitePageData currentPage)
        {
            var model = new FavoriteViewModel();

            model.Favorites = FavoriteRepository
                .GetFavorites(PrincipalInfo.Current.Name);

            model.CurrentPageContentReference = currentPage.ContentLink;

            return PartialView("FavoritesManager", model);
        }
```

```
        public void Add(ContentReference page)
        {
            var favorite = FavoriteRepository.GetFavorite(
                page, PrincipalInfo.Current.Name);

            if (favorite == null)
            {
                var newFavorite = new Favorite(page, PrincipalInfo.Current.Name);
                FavoriteRepository.Save(newFavorite);
            }

            Response.Redirect(urlResolver.GetUrl(page));
        }

        public void Delete(ContentReference page, ContentReference fav)
        {
            var favorite = FavoriteRepository.GetFavorite(
                fav, PrincipalInfo.Current.Name);

            if (favorite != null)
            {
                FavoriteRepository.Delete(favorite);
            }

            Response.Redirect(urlResolver.GetUrl(page));
        }
    }
}
```

4. Inside **~/Business/Initialization**, add a new Episerver **Initialization Module** named **FavoritesInitializationModule.cs**.

```
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using System.Web.Mvc;
using System.Web.Routing;

namespace AlloyAdvanced.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class FavoritesInitializationModule : IInitializableModule
    {
        public void Initialize(InitializationEngine context)
        {
            RouteTable.Routes.MapRoute(
                name: "FavoritesAdd",
                url: "favs/add",
                defaults: new { controller = "Favorites", action = "Add" });

            RouteTable.Routes.MapRoute(
                name: "FavoritesDelete",
                url: "favs/del",
                defaults: new { controller = "Favorites", action = "Delete" });
        }

        public void Uninitialize(InitializationEngine context) { }
    }
}
```

5. Inside **~/Views/Shared**, add an Episerver **Page Partial View (MVC Razor)** named **FavoritesManager.cshtml**.

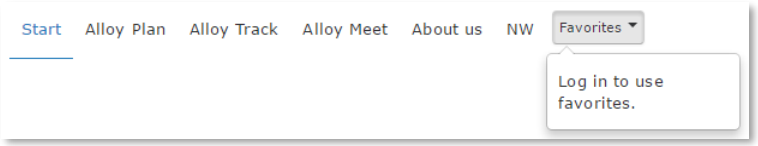
```
@using EPiServer.Security
@model AlloyAdvanced.Models.ViewModels.FavoriteViewModel
<div class="btn-group">
    <a class="btn dropdown-toggle btn-mini" data-toggle="dropdown" href="#">
        Favorites
        <span class="caret"></span>
    </a>
    <ul class="dropdown-menu">
        @if (!string.IsNullOrWhiteSpace(PrincipalInfo.Current.Name))
        {
            <li>
                <a class="btn btn-mini"
                   href="/favs/add?page=@Model.CurrentPageContentReference">
                <span class="icon-plus-sign"></span>
                Add this page to favorites</a>
            </li>
            <li class="divider"></li>
            foreach (var fav in Model.Favorites)
            {
                <li>
                    @Html.ContentLink(fav.FavoriteContentReference)
                    <a class="btn btn-mini" href=
"/favs/del?page=@(Model.CurrentPageContentReference)&fav=@(fav.FavoriteContentReferenc
e)">
                        <span class="icon-remove"></span> Remove
                    </a>
                </li>
            }
        }
        else
        {
            <li><div class="well-small">Log in to use favorites.</div></li>
        }
    </ul>
</div>
```

6. Open **~/Views/Shared/Header.cshtml**, find the `<ul class="nav">` element, as shown in the following code, and add a new **<li>** element as the last list item that calls the Action method to render the Favorites Manager at the end of the navigation menu:

```
<ul class="nav">
    <li
class="@(Model.CurrentPage.ContentLink.CompareToIgnoreWorkID(SiteDefinition.Current.St
artPage) ? "active" : null)">@Html.ContentLink(SiteDefinition.Current.StartPage)</li>
    @Html.MenuList(SiteDefinition.Current.StartPage,
    @<li class="@(item.Selected ? "active" : null)">
        @Html.PageLink(item.Page, null, new { @class = string.Join(" ",
item.Page.GetThemeCssClassNames()) })
    </li>)
    <!--Favourites menu-->
    <li>@Html.Action("Manager", "Favorites")</li>
</ul>
```

Testing the favorites functionality

1. Start the site.
2. Click the **Favorites** menu, and you will see a warning message, as shown in the following screenshot (you might need to force a refresh with *F5*):



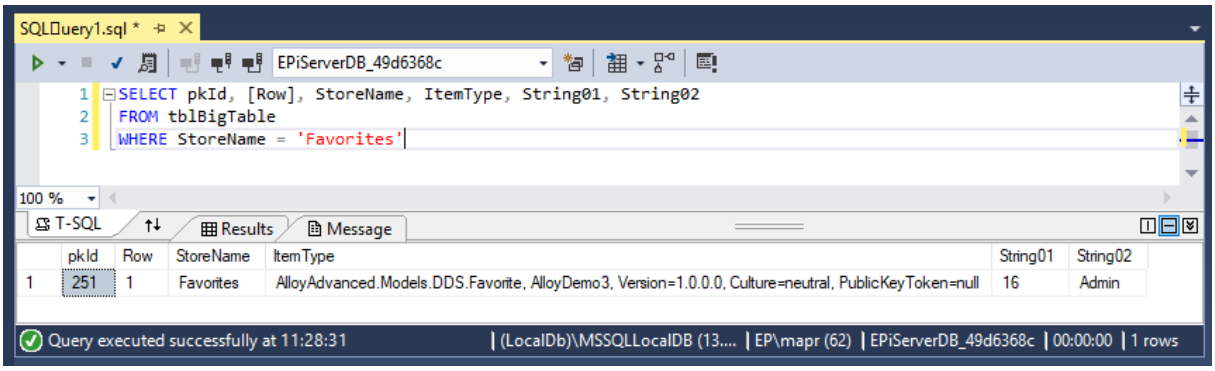
3. Log in as **Admin**.
4. Navigate around the site, and add several pages to the **Favorites** menu.
5. Use the **Favorites** menu to quickly jump to a page.
6. Remove a favorite.
7. Close the browser.

Review the entity in DDS

1. In **Solution Explorer**, in **App_Data**, double-click **EPiServerDB_{xxxxx}.mdf** to open a connection to the CMS database in **Server Explorer**.
2. Right-click **tblBigTable**, click **New Query.**, and enter the following SQL statement:

```
SELECT pkId, [Row], StoreName, ItemType, String01, String02
FROM tblBigTable
WHERE StoreName = 'Favorites'
```

3. Click **Execute** in the query pane toolbar, or press *Ctrl* + *Shift* + *E*, and note the results, as shown in the following screenshot:



In the example above, I only added a single favorite while logged in as Admin. You may have multiple favorites. String01 stores the ContentReference, String02 stores the user's name.

You have now implemented a feature that uses the Dynamic Data Store (DDS).

Warning! Currently, the **Favorites** menu is designed to manage a list of favorites by storing a **ContentReference**. This is so that even if the URL changes, perhaps because a content editor moves the page in the page tree, the content reference will still work. However, in the next topic, you will integrate with content *outside* the CMS using a partial router, and that content will not have ContentReferences, so the external content will not work with the **Favorites** menu.

Exercise C2 – Integrating external data using a partial route

In this exercise, you will create a partial route to integrate an external data source.

Prerequisites: complete Exercises 0.1 and 0.2

Creating a Northwind page type and page template

1. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **NorthwindPage**.
2. Modify its statements as shown in the following code:

Make sure that **NorthwindPage** inherits from **SitePageData**.

```
using AlloyAdvanced.Models.NorthwindEntities;
using EPiServer.DataAnnotations;
using System.Collections.Generic;

namespace AlloyAdvanced.Models.Pages
{
    // If a content editor creates an instance of this page type in the page
    // tree then it becomes the entry point for the Northwind partial router.
    [SiteContentType(DisplayName = "Northwind",
        Description = "A page that lists categories from the Northwind database.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class NorthwindPage : SitePageData
    {
        // this property is used for showing a list of categories
        // in ~/Views/NorthwindPage/Index.cshtml
        [Ignore] // not stored in CMS database
        public Dictionary<string, string> CategoryLinks { get; set; }

        // this property is used for showing details of a category
        // in ~/Views/Category/Index.cshtml
        [Ignore] // not stored in CMS database
        public Category NorthwindCategory { get; set; }
    }
}
```

When creating controllers in the Alloy site, you should inherit from the site-specific class named `PageControllerBase<T>`, instead of the usual Episerver class named `PageController<T>`.

3. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **NorthwindPageController**.
4. Modify its contents as shown in the following code:

```
using AlloyAdvanced.Models.NorthwindEntities;
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Web.Routing;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class NorthwindPageController : PageControllerBase<NorthwindPage>
    {
        private readonly UrlResolver resolver = null;

        public NorthwindPageController(UrlResolver resolver)
```

```
        {
            this.resolver = resolver;
        }

        public ActionResult Index(NorthwindPage currentPage)
        {
            var model = PageViewModel.Create(currentPage);

            // connect to the Northwind database through the domain model
            // to fetch a list of all categories and pass it to the Northwind page
            // instance to show a list of category names and links generated
            // by the partial router.

            var db = new Northwind();

            // we do not need to track changes or
            // automatically load related entities
            db.Configuration.ProxyCreationEnabled = false;
            db.Configuration.LazyLoadingEnabled = false;

            foreach (Category category in db.Categories)
            {
                string name = category.CategoryName;

                string url = resolver.GetVirtualPathForNonContent(
                    partialRoutedObject: category, language: null,
                    virtualPathArguments: null).GetUrl();

                model.CurrentPage.CategoryLinks.Add(name, url);
            }
            return View(model);
        }
    }
}
```

The **UrlResolver** will call our partial router to get the virtual path for each category stored in the external database (non-content).

5. In **Solution Explorer**, right-click **~/Views**, add a folder named **NorthwindPage**.
6. In **~\Views\NorthwindPage**, add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
7. Modify its contents as shown in the following markup:

```
@model PageViewModel<AlloyAdvanced.Models.Pages.NorthwindPage>
<h2>Northwind Categories</h2>
<ul>
    @foreach (KeyValuePair<string, string> category in
Model.CurrentPage.CategoryLinks)
    {
        <li><a href="@category.Value">@category.Key</a></li>
    }
</ul>
```

Creating and registering a partial router

The core of a partial router is a class that can:

- Convert an entity into a partial URL path, and
- A partial URL path into an entity.

1. In **Solution Explorer**, right-click **~/Business**, add a folder named **PartialRouters**, and add a class named **NorthwindToCategoryPartialRouter**.

2. Modify its code as shown below:

Make sure that you import the **System.Data.Entity** namespace to enable the Include() extension method that supports lambda expressions as well as strings.

```
using AlloyAdvanced.Models.NorthwindEntities;
using AlloyAdvanced.Models.Pages;
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using EPiServer.Web.Routing;
using EPiServer.Web.Routing.Segments;
using System.Linq;
using System.Web.Routing;
using System.Data.Entity;

namespace AlloyAdvanced.Business.PartialRouters
{
    public class NorthwindToCategoryPartialRouter :
        IPartialRouter<NorthwindPage, Category>
    {
        // this method must convert a Category entity into a partial URL path
        // e.g. the category named "Meat/Poultry"
        // into the URL "/Northwind/Meat_Poultry"
        public PartialRouteData GetPartialVirtualPath(Category content,
            string language, RouteValueDictionary routeValues,
            RequestContext requestContext)
        {
            var northwindPages = ServiceLocator.Current.GetInstance<IContentLoader>()
                .GetChildren<NorthwindPage>(ContentReference.StartPage);

            // the base of the URL will be the URL for the NorthwindPage instance
            var basePath = ContentReference.EmptyReference;
            if (northwindPages.Count() > 0)
            {
                basePath = northwindPages.First().ContentLink;
            }

            var partialRouteData = new PartialRouteData
            {
                BasePathRoot = basePath,
                PartialVirtualPath = content.CategoryName.Replace('/','_') + "/"
            };

            return partialRouteData;
        }

        // this method must convert a partial URL path into a Category entity
        // e.g. the URL "/Northwind/Meat_Poultry"
        // into the category named "Meat/Poultry"
        public object RoutePartial(NorthwindPage content,
            SegmentContext segmentContext)
        {
            // get the next part from the URL, i.e. what comes after "/Northwind/"
            var categorySegment = segmentContext.GetNextValue(
                segmentContext.RemainingPath);
            var categoryName = categorySegment.Next;

            // find the matching Category entity
            var db = new Northwind();
            db.Configuration.ProxyCreationEnabled = false;
            db.Configuration.LazyLoadingEnabled = false;
```

```
            var alternativeCategory = categoryName.Replace('_', '/');
            var category = db.Categories
                .Include(c => c.Products)
                .SingleOrDefault(c =>
                    (c.CategoryName == categoryName) ||
                    (c.CategoryName == alternativeCategory));

            if (category != null)
            {
                // store the found Category in the route data
                // so it can be passed into a view
                segmentContext.SetCustomRouteData("category", category);

                // store the remaining path so it could be processed
                // by another partial router, perhaps for Products
                segmentContext.RemainingPath = categorySegment.Remaining;
            }

            return category;
        }
    }
}
```

Partial routers must be registered during the Episerver initialization process.

3. In **Solution Explorer**, right-click **~/Business/Initialization**, and add an Episerver **Initialization Module** named **NorthwindToCategoryPartialRouterRegistration**.
4. Modify its code as shown below:

```
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using System.Web.Routing;
using EPiServer.Web.Routing;
using AlloyAdvanced.Business.PartialRouters;

namespace AlloyAdvanced.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class NorthwindToCategoryPartialRouterRegistration : IInitializableModule
    {
        public void Initialize(InitializationEngine context)
        {
            RouteTable.Routes.RegisterPartialRouter(
                new NorthwindToCategoryPartialRouter());
        }

        public void Uninitialize(InitializationEngine context) { }
    }
}
```

Creating a Category page template

We already have a class that represents a Category and a view model for passing it to a view. But we need to create the equivalent of a page template to render a Category: a combination of a controller and a view.

1. In **Solution Explorer**, right-click **~/Controllers**, and add class named **CategoryController**.
2. Modify its code as shown below:

```
using AlloyAdvanced.Models.NorthwindEntities;
```

```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using EPiServer.Web;
using EPiServer.Web.Routing;
using System.Linq;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    // by implementing IRenderTemplate<Category>, this becomes
    // the "page template" for a Category entity
    public class CategoryController : Controller, IRenderTemplate<Category>
    {
        public ActionResult Index()
        {
            // Note: the GetRoutedData extension method uses the partial router to
            // convert a URL segment into a Category instance.
            var category = Request.RequestContext.GetRoutedData<Category>();

            var northwindPages = ServiceLocator.Current.GetInstance<IContentLoader>()
                .GetChildren<NorthwindPage>(ContentReference.StartPage);

            NorthwindPage currentPage = null;
            if (northwindPages.Count() > 0)
            {
                currentPage = northwindPages.First();
            }

            var model = PageViewModel.Create(currentPage);
            model.CurrentPage.NorthwindCategory = category;

            return View(model);
        }
    }
}
```

3. In **Solution Explorer**, right-click **~/Views**, add a folder named **Category**, and add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.

4. Modify its code as shown below:

```
@using AlloyAdvanced.Models.NorthwindEntities
@model PageViewModel<AlloyAdvanced.Models.Pages.NorthwindPage>
<h2>@Model.CurrentPage.NorthwindCategory.CategoryName</h2>
<div>
    @Model.CurrentPage.NorthwindCategory.Description
</div>
<table class="table table-striped">
    <tr><th>Product Name</th><th>Unit Price</th></tr>
    @foreach (Product p in Model.CurrentPage.NorthwindCategory.Products)
    {
        <tr><td>@p.ProductName</td><td>@p.UnitPrice.Value.ToString("$0.00")</td></tr>
    }
</table>
```

Prevent the Favorites menu from appearing for the partial route

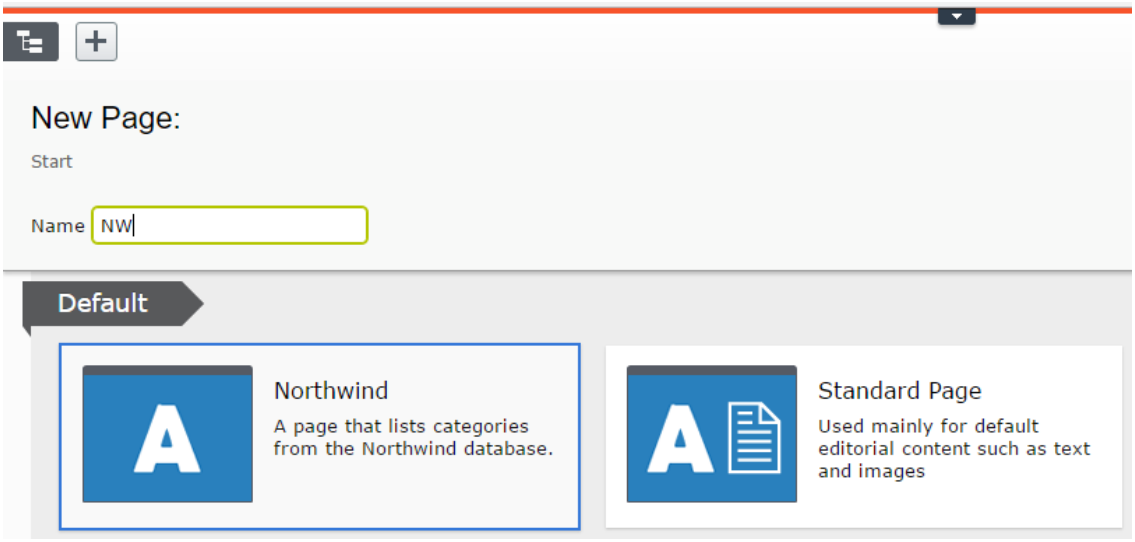
If you completed Exercise C1, then you will need to disable the Favorites menu for any page that was generated by this partial route.

1. Open **~\Views\Shared\Header.cshtml**.
2. Modify the view to only output the **Favorites** menu if the model is NOT a **NorthwindPageViewModel**, as shown in the following markup:

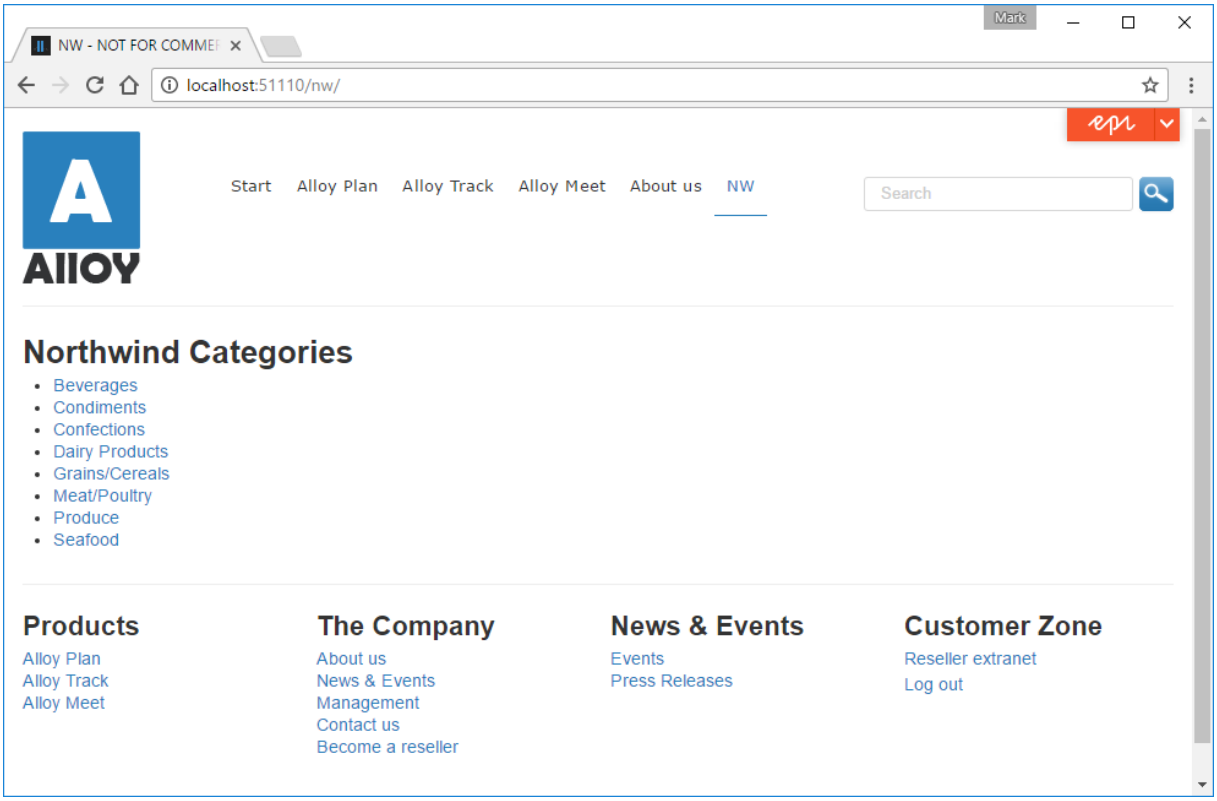
```
<!--Favourites menu-->
@if (!(Model is NorthwindPageViewModel))
{
<li>@Html.Action("Manager", "Favorites")</li>
}
```

Testing the partial router

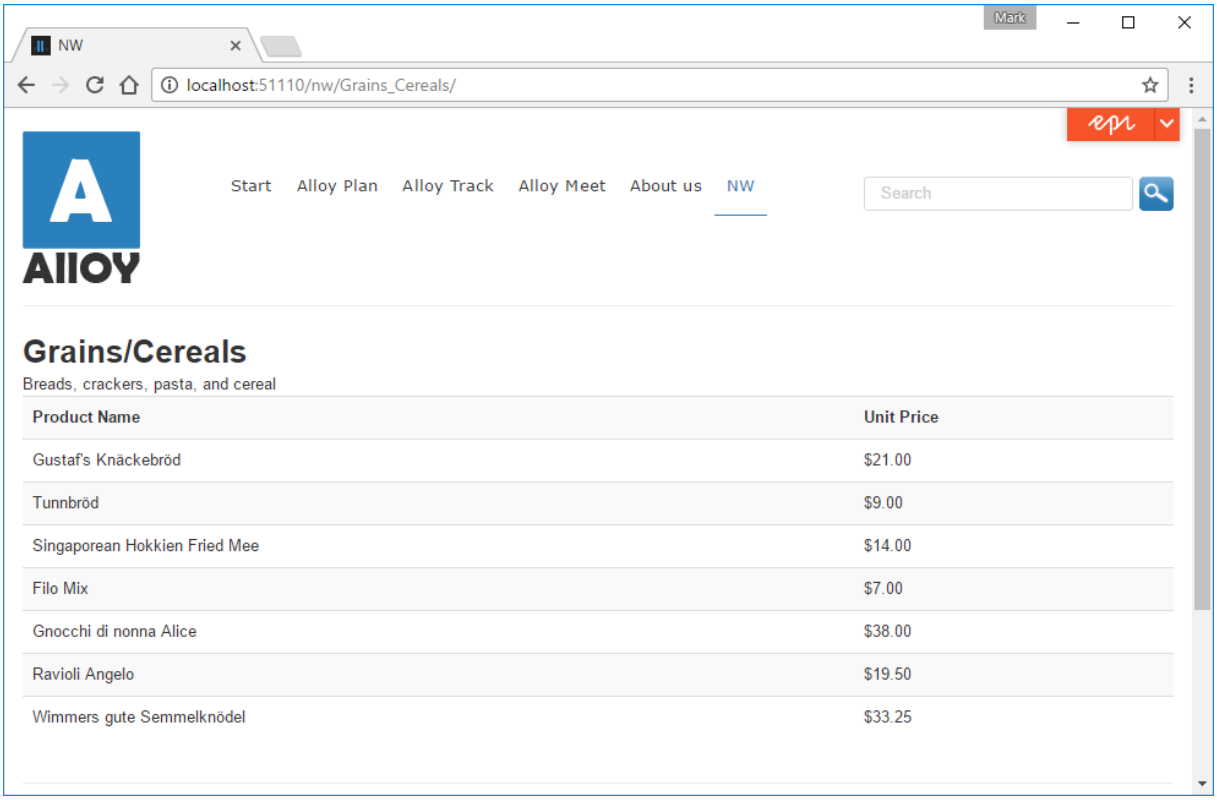
1. Start the site, and log in as **Admin**.
2. Switch to **Edit View**, and add a **Northwind** page named **NW** underneath the **Start** page.



3. **Publish** the page and switch to visitor view.



4. Click **Grains/Cereals** and note the URL path.



Exercise C3 – Gathering data using Episerver Forms

In this exercise, you will create an Episerver Forms form and an ASP.NET Web API service to act as a Web Hook.

Prerequisites: complete Exercise 0.1.

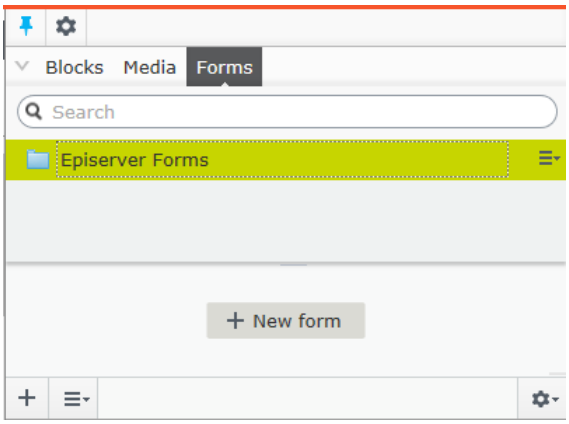
Installing the Episerver Forms add-on

1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
3. Set **Package source** to **All**, and enter the following command:

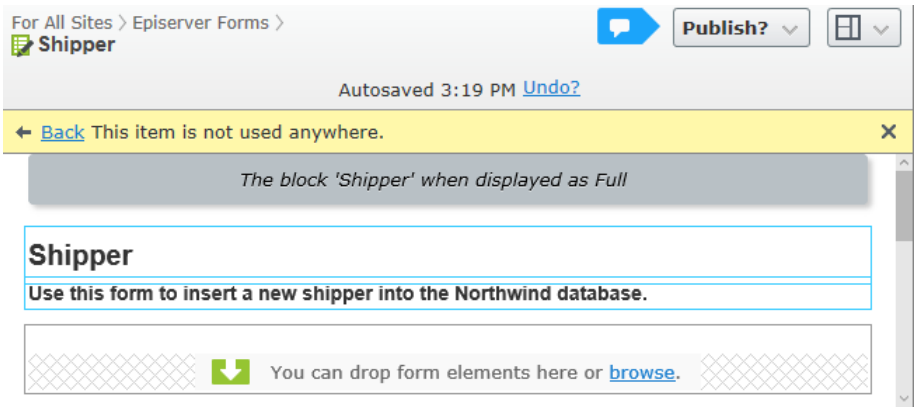
```
Install-Package -ProjectName AlloyAdvanced EPiServer.Forms
```

Creating an Episerver Forms form

1. Start the site and log in as **Admin**.
2. Edit the **Start** page.
3. In the **Assets** pane, click **Forms**, and then click **+ New form**, as shown in the following screenshot:

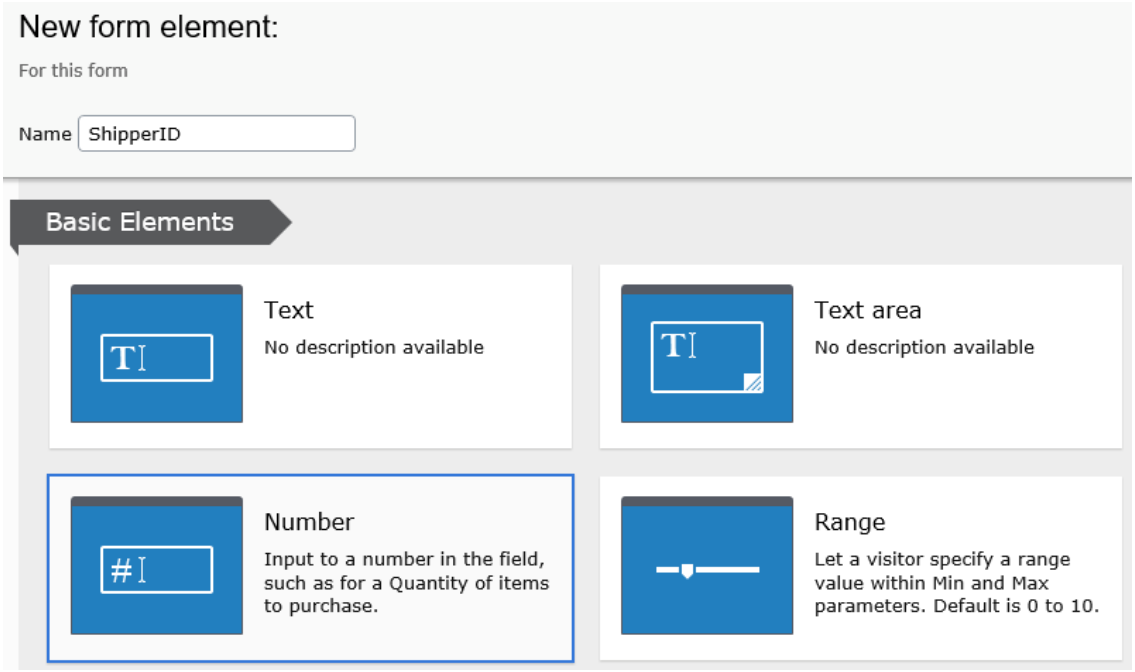


4. Name the form **Shipper**, and click **OK**.
5. Set the title to Shipper and description to Use this form to insert a new shipper into the Northwind database, as shown in the following screenshot:

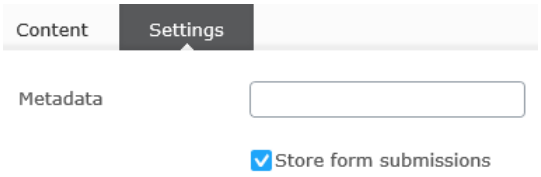


6. To create the first input element, click **browse**.

7. Enter the name **ShipperID**, and click **Number**, as shown in the following screenshot:

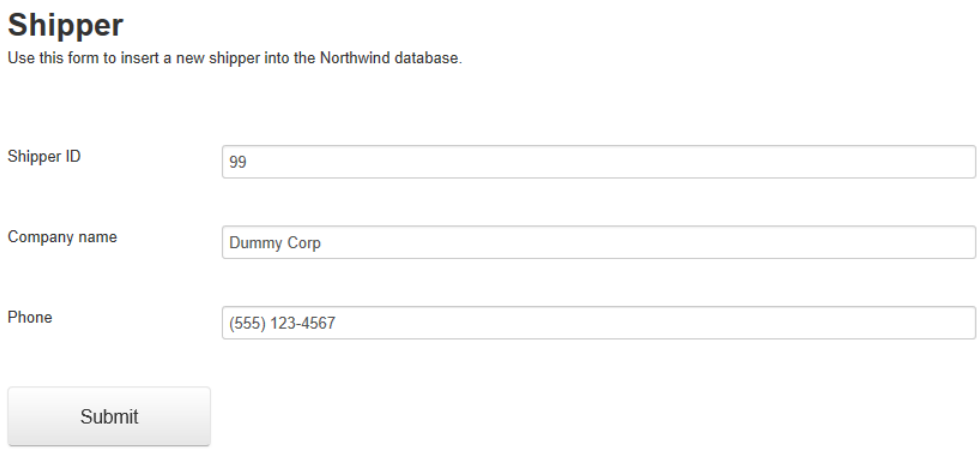


8. Click the context menu, and click **Edit**.
9. Change the **Label text** to **Shipper ID**, check the **Required** box, and then **Publish** the changes.
10. To create the second input element, in the **Assets** pane, expand the **Form Elements** section, and drag and drop a **Text** element onto the form.
11. Click the context menu, and click **Edit**.
12. Change the Name to **CompanyName**, the Label text to **Company name**, check **Required**, and **Publish** the changes.
13. Add another **Text** element named **Phone**.
14. Add a **Submit** element.
15. Switch to **All Properties** view for the form.
16. On the **Content** tab, scroll down, and set the **Allow multiple submissions from the same IP/cookie** checkbox.
17. Click **Settings**, and note the check box to **Store form submissions**. This means submitted forms will be stored in Dynamic Data Store, as shown in the following screenshot:

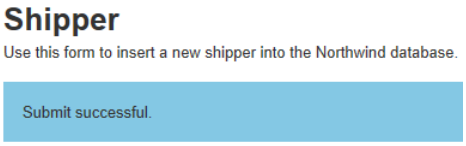


18. Publish the **Shipper** form.
19. Edit the **Start** page.
20. Drag and drop the **Shipper** form into the **Large content area**.
21. **Publish** changes to the Start page.
22. View the site as a visitor.

23. Complete the form with some sample data, as shown in the following screenshot, and click **Submit**:

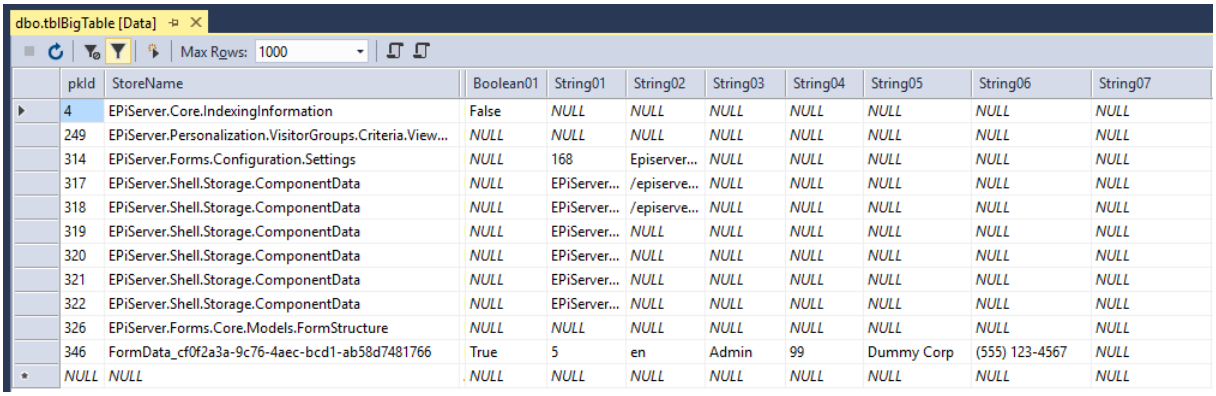


24. When you see the success message, as shown in the following screenshot, close the browser:



View the stored data

1. In Visual Studio, expand **App_Data**, and double-click the **EPiServerDB_xxxxx.mdf**.
2. In Server Explorer, expand **Tables**, right-click **tblBigTable**, and choose **Show Table Data**.
3. Note the last row. The **StoreName** will begin with **FormData_** and then a GUID that represents the form definition in the CMS content database. Boolean01 indicates if the submission is finalized. String01 to String03 are used to record language and username, then String04 to String06 are used for the three inputs in our form, as shown in the following screenshot:

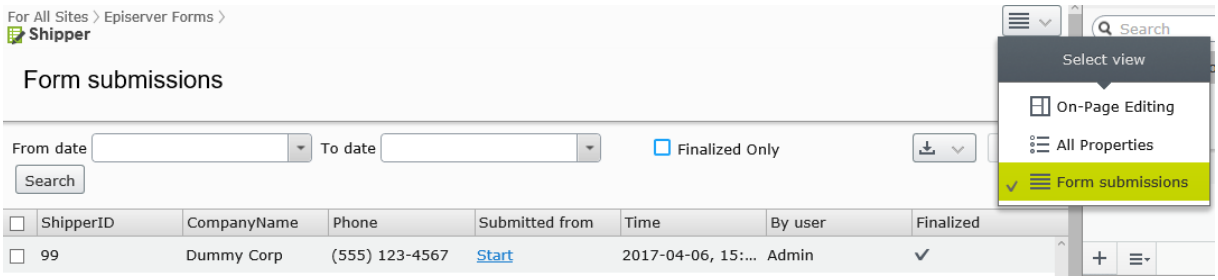

dbo.tblBigTable [Data]

Max Rows: 1000

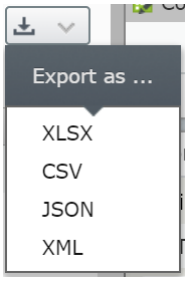
pkId	StoreName	Boolean01	String01	String02	String03	String04	String05	String06	String07
4	EPiServer.Core.IndexingInformation	False	NULL	NULL	NULL	NULL	NULL	NULL	NULL
249	EPiServer.Personalization.VisitorGroups.Criteria.View...	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
314	EPiServer.Forms.Configuration.Settings	NULL	168	Episerver...	NULL	NULL	NULL	NULL	NULL
317	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	/episerve...	NULL	NULL	NULL	NULL	NULL
318	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	/episerve...	NULL	NULL	NULL	NULL	NULL
319	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	NULL	NULL	NULL	NULL	NULL	NULL
320	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	NULL	NULL	NULL	NULL	NULL	NULL
321	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	NULL	NULL	NULL	NULL	NULL	NULL
322	EPiServer.Shell.Storage.ComponentData	NULL	EPiServer...	NULL	NULL	NULL	NULL	NULL	NULL
326	EPiServer.Forms.Core.Models.FormStructure	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
346	FormData_cf0f2a3a-9c76-4aec-bcd1-ab58d7481766	True	5	en	Admin	99	Dummy Corp	(555) 123-4567	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Start the site and log in as **Admin**.
5. Edit the **Start** page.
6. In **Assets** pane, double-click **Shipper** form.

7. Switch to **Form submissions** view, as shown in the following screenshot, and note this view shows the same information stored in **tblBigTable**:

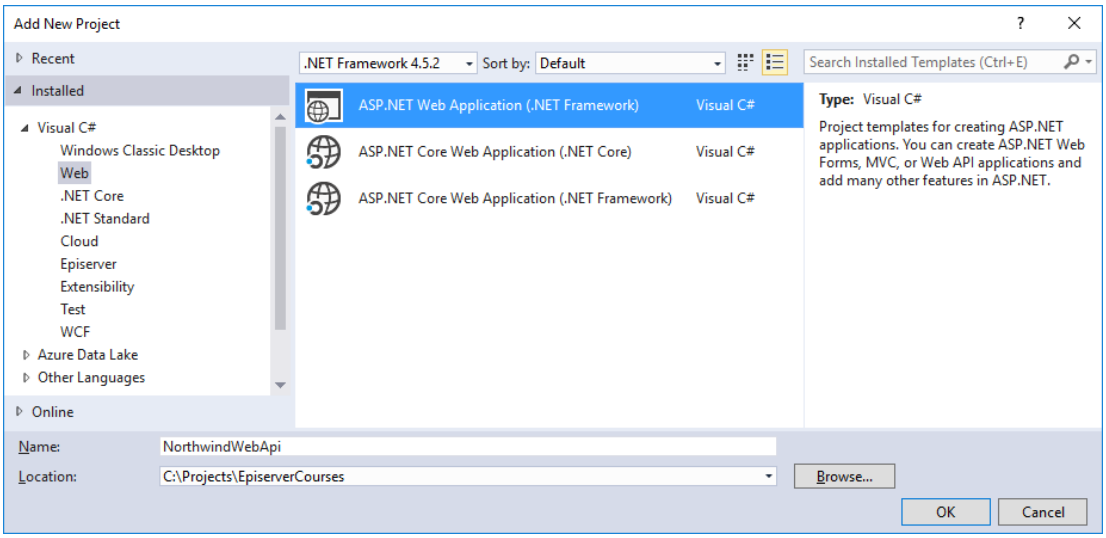


8. Although you can export this data in common formats, as shown in the following screenshot, you will now create an ASP.NET Web API service that will be called whenever this form is submitted, to perform any custom processing needed:

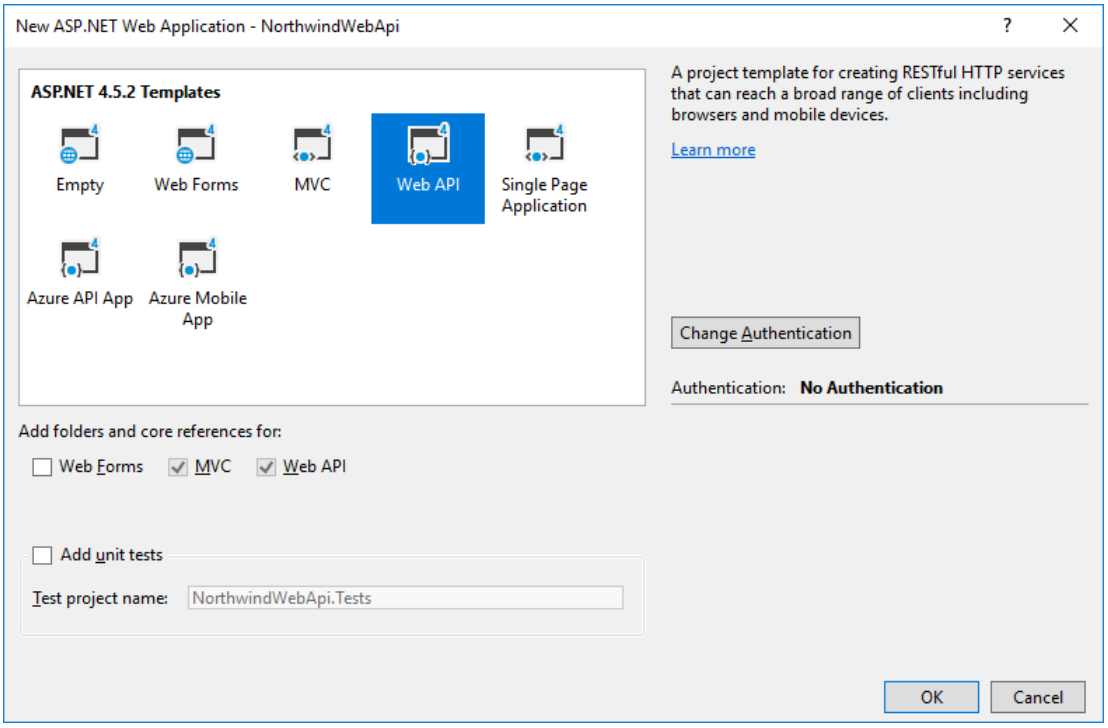


Creating a Web Hook for an Episerver Forms form

1. In Visual Studio, add a new project, choose **ASP.NET Web Application**, name it **NorthwindWebApi**, and click **OK**, as shown in the following screenshot:



2. Choose **Web API**, and click **OK**, as shown in the following screenshot:



3. Right-click **Models**, and add a class named **Shipper**.
4. Modify the contents, as shown in the following code:

```
namespace NorthwindWebApi.Models
{
    public class Shipper
    {
        public int ShipperID { get; set; }
        public string CompanyName { get; set; }
        public string Phone { get; set; }
    }
}
```

The model class must have public propreties that match submitted data, i.e., the name of the elements in the Episerver Forms form.

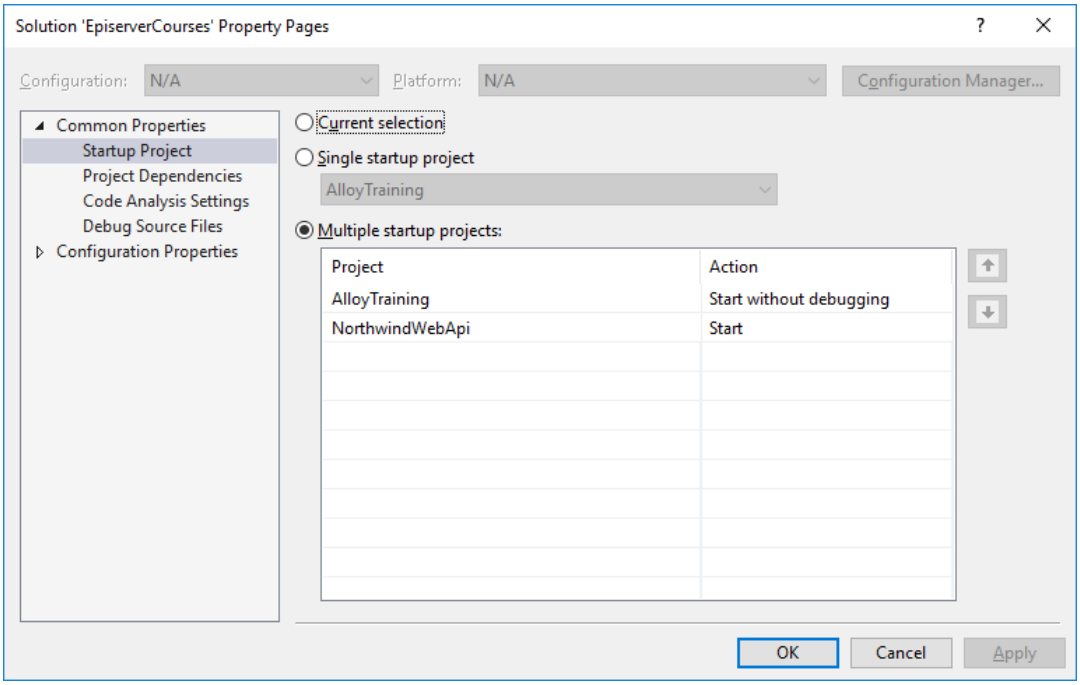
5. Right-click **Controllers**, and choose **Add** | **Controller**...
6. Choose a **Web API 2 Controller – Empty**, and click **Add**.
7. Enter the name **ShipperFormController**.
8. Modify the contents, as shown in the following code:

```
using NorthwindWebApi.Models;
using System;
using System.IO;
using System.Web.Http;

namespace NorthwindWebApi.Controllers
{
    public class ShipperFormController : ApiController
    {
        public string Get()
        {
            return "ShipperFormController says, 'Hi!'";
        }
```

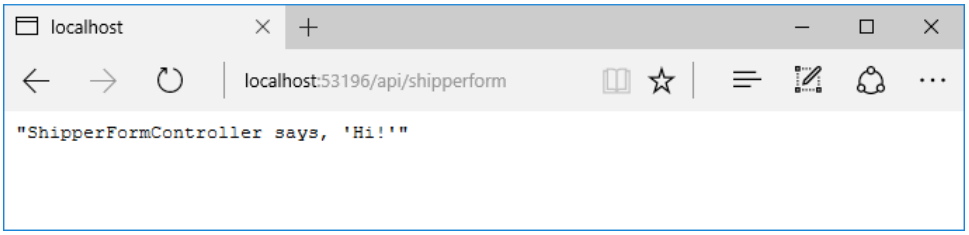
```
        public void Post(Shipper shipper)
        {
            string add_data = System.Web.HttpContext.Current
                .Server.MapPath(@"~\App_Data");

            string path = Path.Combine(add_data,
                $"shipperform_{DateTime.Now.ToString("yyyy-MM-dd_hh-mm-ss")}");

            File.WriteAllText(path, $"{shipper.ShipperID}: {shipper.CompanyName}");
        }
    }
}
```

ASP.NET Web API has a model binder that will automatically instantiate the Shipper class and populate its properties with the submitted data values.

9. In **Solution Explorer**, right-click the solution and choose **Set StartUp Projects**.
10. Set **AlloyAdvanced** to **Start without debugging**, and **NorthwindWebApi** to **Start**, as shown in the following screenshot, and click **OK**:

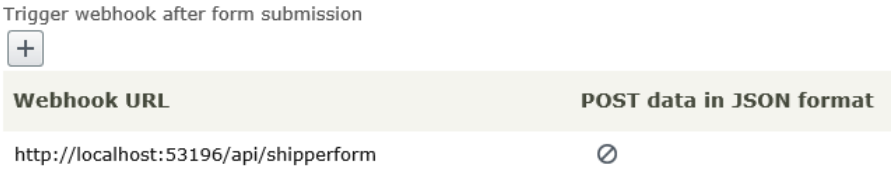


11. Start the solution.
12. Enter the following at the end of the address bar: `api/shipperform`, and press Enter, to check that the service responds, as shown in the following screenshot:

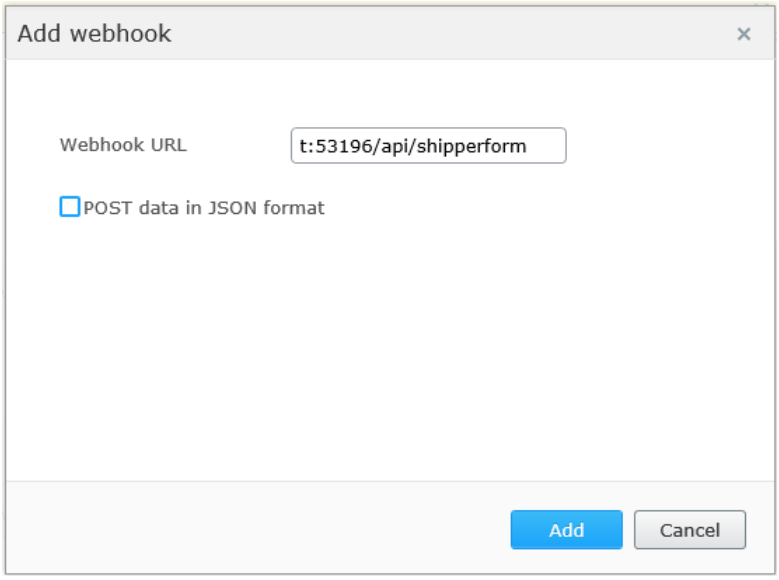


13. Start the **AlloyAdvanced** site, and log in as **Admin**.
14. Edit the **Start** page.
15. In **Assets**, double-click the **Shipper** form.

16. Switch to **All Properties** view.
17. Click **Settings**.
18. Click the + button to create a web hook, as shown in the following screenshot:

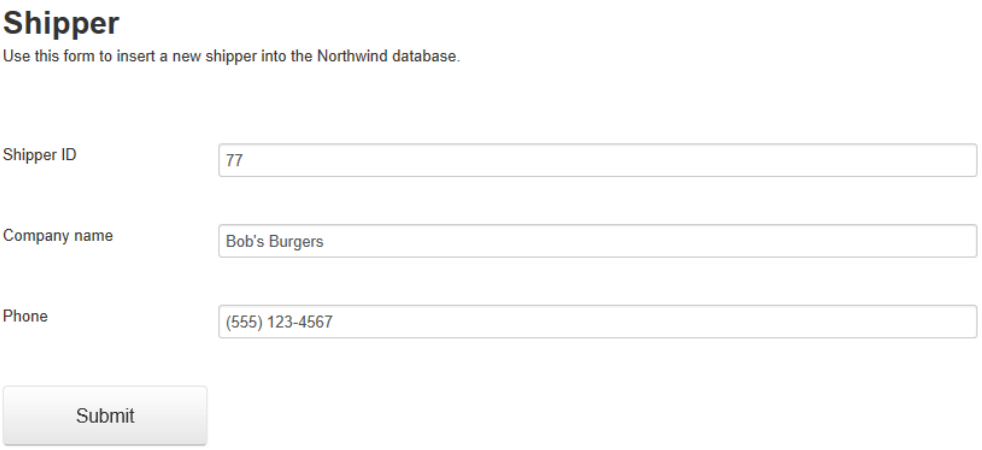


19. Enter the URL, `http://localhost:xxxxx/api/shipperform`, as shown in the following screenshot, and click **Add**:

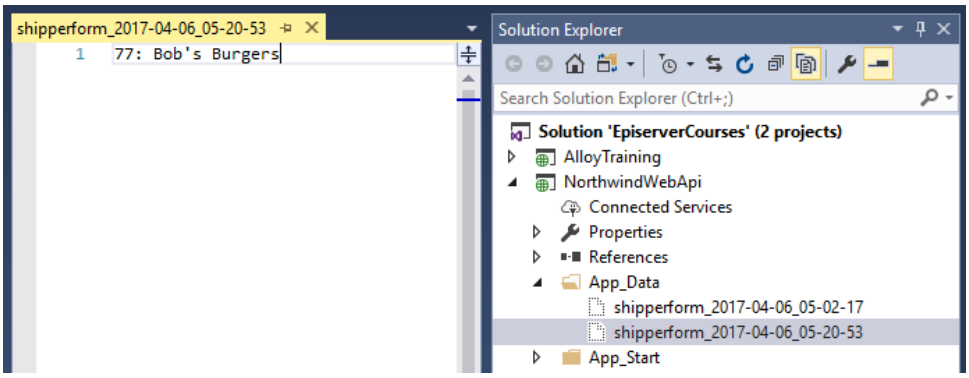


Your port number will be different!

20. Publish changes to the form.
21. View the site as a visitor.
22. Complete the Shipper form with some sample data, as shown in the following screenshot, and click **Submit**.



23. In **NorthwindWebApi** project, expand **~\App_Data**, and **Show All Files**, and you will see a new file has been created:



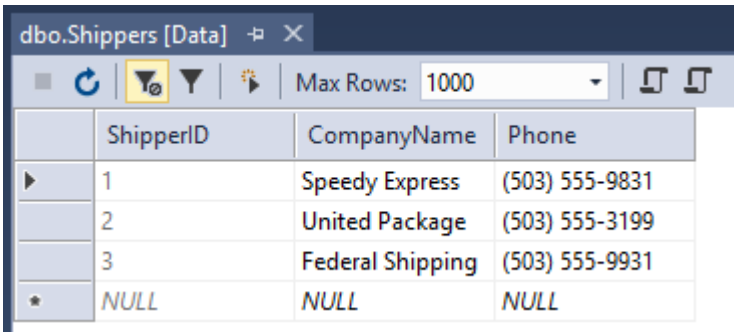
In this exercise, the Web API service wrote the posted data to a plain text file, but of course, in the real world it could do anything you like!

Exercise C4 – Importing data using a scheduled job

In this exercise, you will create a scheduled job to connect to an external data source and regularly import new records.

Prerequisites: complete Exercises 0.1 and 0.2.

The **Northwind** database has a table named **Shippers**, as shown in the following screenshot:



ShipperID	CompanyName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931
NULL	NULL	NULL

We need to import the data in this table and store it as content in the CMS, and then enrich it with additional data, for example, shipping costs, and allow content editors to mark one as the default to use for shipping. If new records are added in the future, those records must also be imported.

We will create a scheduled job to do this work, but first, we must define some content types to store the data inside the CMS, and render it to the visitors.

Creating Shippers and Shipper page types, and a page template for Shippers

ShippersPage will show a list of shippers that have been imported into instances of **ShipperPage** and that are stored as children of the **ShippersPage**. **ShippersPage** can be added under a **StartPage** and can only have **ShipperPage** instances as its children.

1. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **ShippersPage**.
2. Modify its statements as shown in the following code:

```
using EPiServer.DataAnnotations;

namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(DisplayName = "Shippers",
        Description = "Displays a list of imported shippers.")]
    [SiteImageUrl]
    [AvailableContentTypes(
        Availability = EPiServer.DataAbstraction.Availability.Specific,
        Include = new[] { typeof(ShipperPage) },
        IncludeOn = new[] { typeof(StartPage) })]
    public class ShippersPage : SitePageData
    {
    }
}
```

3. In **Solution Explorer**, right-click **~/Models/Pages**, and add an Episerver **Page Type** named **ShipperPage**.
4. Modify its statements as shown in the following code:

```
using EPiServer.Core;
using EPiServer.DataAnnotations;
using System.ComponentModel.DataAnnotations;

namespace AlloyAdvanced.Models.Pages
```

```
{
    [SiteContentType(DisplayName = "Shipper",
        AvailableInEditMode = false,
        Description = "A templateless leaf page node to store shipper data.")]
    [SiteImageUrl]
    public class ShipperPage : PageData
    {
        public virtual int ShipperID { get; set; }

        [StringLength(40)]
        public virtual string CompanyName { get; set; }

        [StringLength(24)]
        public virtual string Phone { get; set; }

        // properties to enrich the imported data

        public virtual string CostPerUnit { get; set; }
    }
}
```

5. In **Solution Explorer**, in **~/Models/Pages**, open the class named **StartPage**.
6. Modify its contents to add a property named **Shippers** that can only reference an instance of **ShippersPage**:

```
public class StartPage : SitePageData
{
    [AllowedTypes(typeof(ShippersPage))]
    public virtual ContentReference Shippers { get; set; }
```

7. In **Solution Explorer**, right-click **~/Models/ViewModels**, and add a class named **ShippersPageViewModel**.
8. Modify its contents as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using EPiServer.Core;
using System.Collections.Generic;

namespace AlloyAdvanced.Models.ViewModels
{
    public class ShippersPageViewModel : IPageViewModel<ShippersPage>
    {
        public ShippersPageViewModel(ShippersPage currentPage)
        {
            CurrentPage = currentPage;
        }
        public ShippersPage CurrentPage { get; set; }
        public LayoutModel Layout { get; set; }
        public IContent Section { get; set; }
        public IEnumerable<ShipperPage> Shippers { get; set; }
    }
}
```

When creating controllers in the Alloy site, you should inherit from the site-specific class named `PageControllerBase<T>`, instead of the usual Episerver class named `PageController<T>`.

9. In **Solution Explorer**, right-click **~/Controllers**, and add an Episerver **Page Controller (MVC)** named **ShippersPageController**.
10. Modify its contents as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
```

```
using AlloyAdvanced.Models.ViewModels;
using EPiServer;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    public class ShippersPageController : PageControllerBase<ShippersPage>
    {
        private readonly IContentLoader loader;

        public ShippersPageController(IContentLoader loader)
        {
            this.loader = loader;
        }

        public ActionResult Index(ShippersPage currentPage)
        {
            var model = new ShippersPageViewModel(currentPage);
            model.Shippers = loader.GetChildren<ShipperPage>(
                currentPage.ContentLink);
            return View(model);
        }
    }
}
```

11. In **Solution Explorer**, right-click **~/Views**, add a folder named **ShippersPage**, and add an Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**.
12. Modify its code as shown below:

```
@model AlloyAdvanced.Models.ViewModels.ShippersPageViewModel
<table class="table table-condensed table-bordered table-condensed">
    <tr>
        <th>
            ID
        </th>
        <th>
            Company Name
        </th>
        <th>
            Phone
        </th>
        <th>
            Cost Per Unit
        </th>
        <th>
            Last changed
        </th>
    </tr>
    @foreach (ShipperPage item in Model.Shippers)
    {
        <tr>
            <td>
                @item.ShipperID
            </td>
            <td>
                @item.CompanyName
            </td>
            <td>
                @item.Phone
            </td>
            <td>
                @item.CostPerUnit
```

```
                </td>
                <td>
                    @item.Changed
                </td>
            </tr>
        }
    </table>
```

Create a scheduled job to import shippers

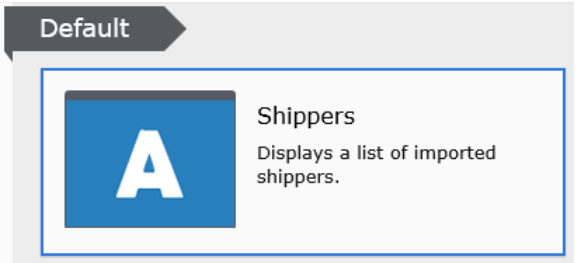
1. In **Solution Explorer**, right-click **~/Business**, add a folder named **ScheduledJobs**, and add an Episerver **Scheduled Job** named **ImportShippersScheduledJob**.
2. Modify its code as shown below:

```
using System.Linq;
using EPiServer.Core;
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using AlloyAdvanced.Models.NorthwindEntities;
using EPiServer;
using AlloyAdvanced.Models.Pages;

namespace AlloyAdvanced.Business.ScheduledJobs
{
    [ScheduledPlugIn(DisplayName = "Import Shippers")]
    public class ImportShippersScheduledJob : ScheduledJobBase
    {
        private bool _stopSignaled;

        public ImportShippersScheduledJob()
        {
            IsStoppable = true;
        }

        private readonly IContentRepository repo;

        public ImportShippersScheduledJob(IContentRepository repo) : this()
        {
            this.repo = repo;
        }

        public override void Stop()
        {
            _stopSignaled = true;
        }

        public override string Execute()
        {
            int shippersImported = 0;

            OnStatusChanged(
                "Starting execution of 'Import Shippers' job.");

            var startPage = repo.Get<StartPage>(ContentReference.StartPage);

            var existingShippers = repo.GetChildren<ShipperPage>(startPage.Shippers);

            var existingIDs = existingShippers.Select(s => s.ShipperID).ToArray();

            var db = new Northwind();

            var shippers = db.Shippers
```

```
                .Where(s => !existingIDs.Contains(s.ShipperID));

            foreach (Shipper item in shippers)
            {
                var newshipper = repo.GetDefault<ShipperPage>(startPage.Shippers);

                newshipper.Name = item.CompanyName;
                newshipper.ShipperID = item.ShipperID;
                newshipper.CompanyName = item.CompanyName;
                newshipper.Phone = item.Phone;

                repo.Save(newshipper,
                    EPiServer.DataAccess.SaveAction.Publish,
                    EPiServer.Security.AccessLevel.NoAccess);

                shippersImported++;

                if (_stopSignaled)
                {
                    return "'Import Shippers' job was stopped.";
                }
            }

            if (shippersImported == 0)
            {
                return "No new shippers to import.";
            }
            else
            {
                return string.Format(
                    "Successfully imported {0} shippers.",
                    shippersImported);
            }
        }
    }
}
```

Test the scheduled job

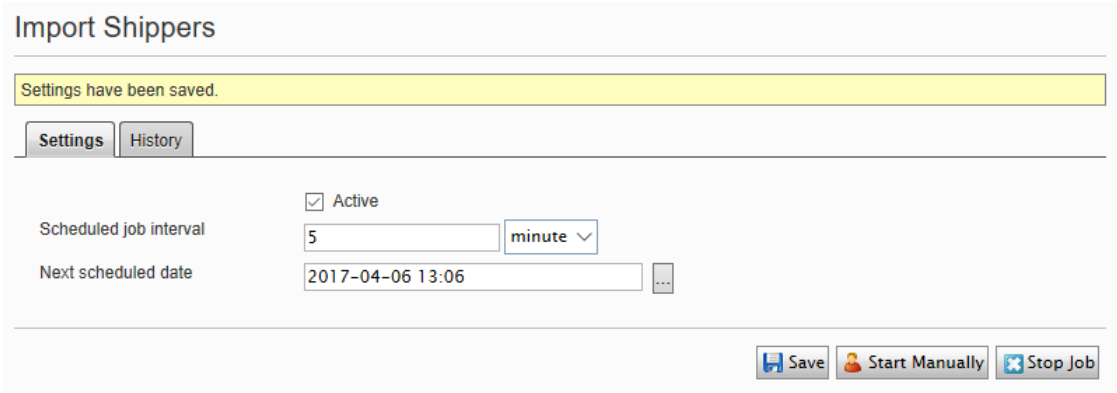
1. Start the site and log in as **Admin**.
2. Create a **Shippers** page underneath the **Start** page.



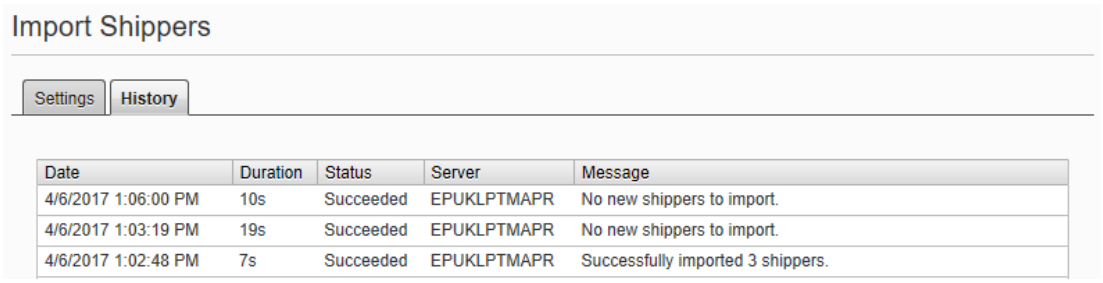
3. Publish the **Shippers** page.

4. Edit the **Start** page, and set the **Shippers** property to reference the **Shippers** page, as shown in the following screenshot:

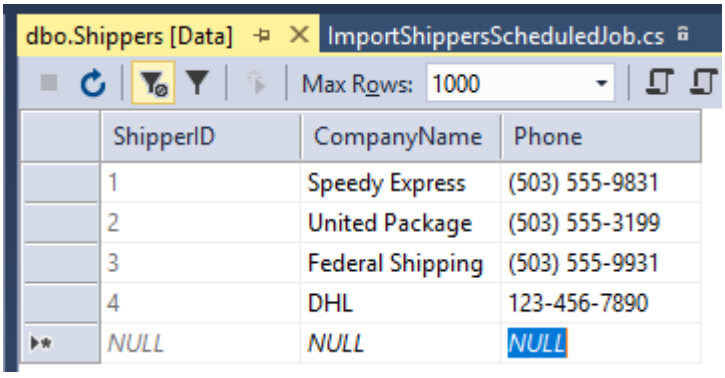
5. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Import Shippers**.
6. Set the job to be active, execute every 5 minutes, and save the changes, as shown in the following screenshot:



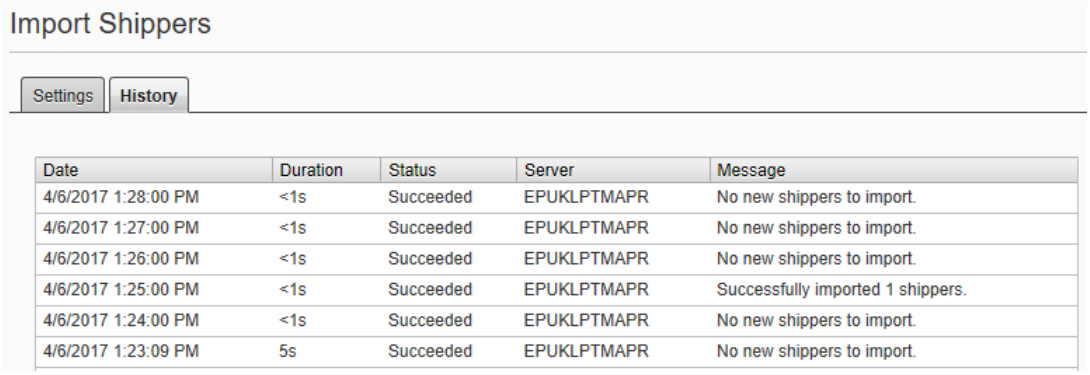
7. Wait for a few minutes and check the **History**, as shown in the following screenshot:



8. In Visual Studio, use **Server Explorer** to open the Northwind database, and add a new shipper, as shown in the following screenshot:



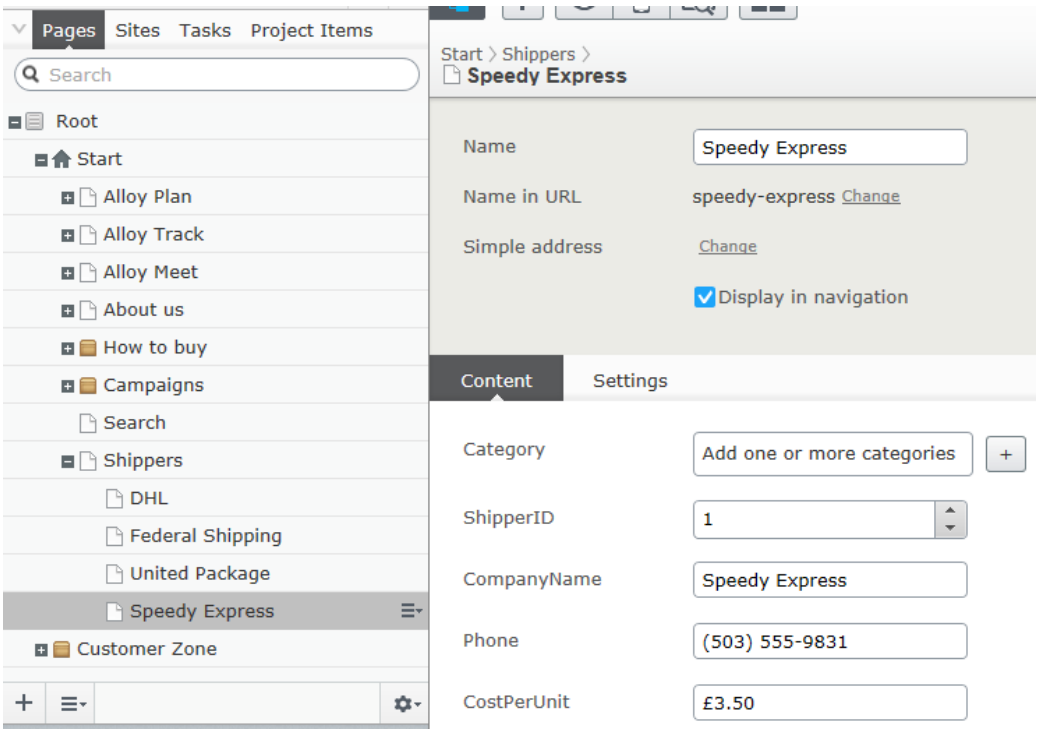
9. Back in the **Import Shippers** scheduled job, view the **History**, as shown in the following screenshot:



Add a default shipper option

Each shipper might have a cost per unit shipped. These values will enrich the data imported. A content editor may also want to choose a default shipper and store that choice in the CMS.

1. Navigate to **CMS** | **Edit**, and enrich the shipper pages by adding values for cost per unit for each shipper, as shown in the following screenshot:



2. Open **~\Models\Pages\ShippersPage**.
3. Add a property named **DefaultShipper** that will store an int shipper ID value.

```
public virtual int DefaultShipper { get; set; }
```

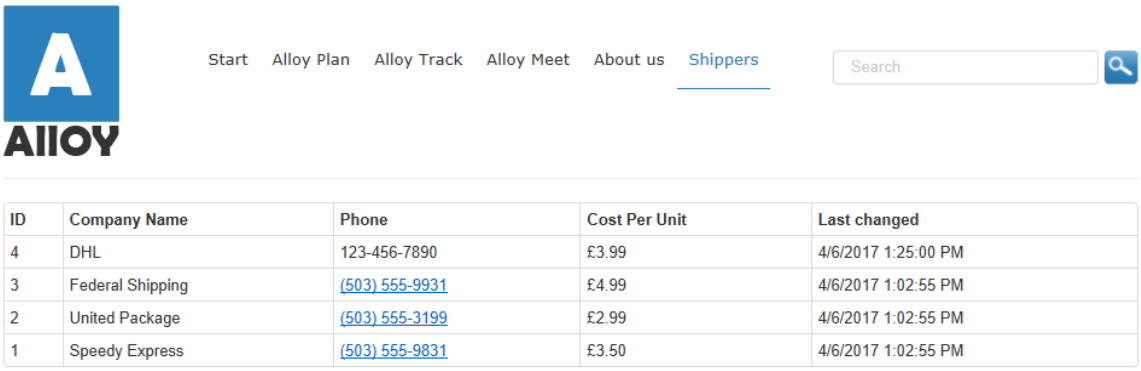
In this exercise, the editor must know the integer value for the shipper that will they want to be the default. In the real world, you would use a custom SelectionFactory to allow the editor to pick from a drop-down listbox of shippers. You will learn how to do this in Module D exercises.

4. Open **~\Views\ShippersPage\Index.cshtml**.
5. Above the table, add a rendering of the DefaultShipper property but only if the page is being edited, as shown in the following code:

```
@if (EPiServer.Editor.PageEditing.PageIsInEditMode)
{
    <div>
```

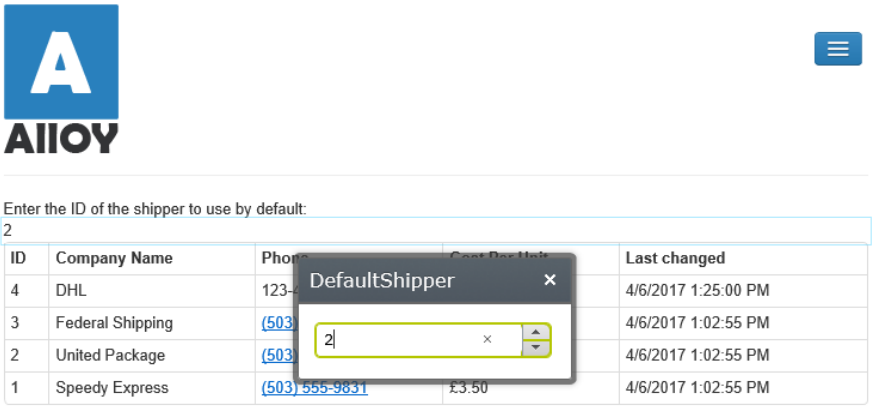
```
            Enter the ID of the shipper to use by default:
            @Html.PropertyFor(m => m.CurrentPage.DefaultShipper)
        </div>
    }
```

6. View the Shippers page as a visitor, as shown in the following screenshot:



ID	Company Name	Phone	Cost Per Unit	Last changed
4	DHL	123-456-7890	£3.99	4/6/2017 1:25:00 PM
3	Federal Shipping	(503) 555-9931	£4.99	4/6/2017 1:02:55 PM
2	United Package	(503) 555-3199	£2.99	4/6/2017 1:02:55 PM
1	Speedy Express	(503) 555-9831	£3.50	4/6/2017 1:02:55 PM

7. Edit the Shippers page, and set the default to the lowest cost shipper, as shown in the following screenshot:



Next steps

How would you handle a requirement to perform updates to already imported shippers?

Module D – Customizing Properties

Goal

The overall goal of the exercises in this module is to see how you can customize the user experience for content editors with properties. You will:

- Provide choice selection for property values.
- Localize selection factories used by editors to set property values.
- Apply customizations at design time with UI hints.
- Apply customizations at runtime with editor descriptors.
- Review a custom property type and its use.
- Create a custom editing experience using Dojo.

Exercise D1 – Applying CSS to a property editor

In this exercise, you will apply a CSS style to the default property editor for the string type so that the text box is wider.

Prerequisites: complete Exercise 0.1.

Reviewing existing contact pages

1. Open the solution with the **AlloyAdvanced** project.
2. Open **~/Global.cs**.
3. Modify the **SiteUIHints** class to add a string const named **Embiggen**, as shown in the following code:

```
public static class SiteUIHints
{
    public const string Embiggen = "Embiggen";
    public const string Contact = "contact";
    public const string Strings = "StringList";
}
```

4. In **~/Business/EditorDescriptors** folder, add a new class named **EmbiggenEditorDescriptor**, as shown in the following code:

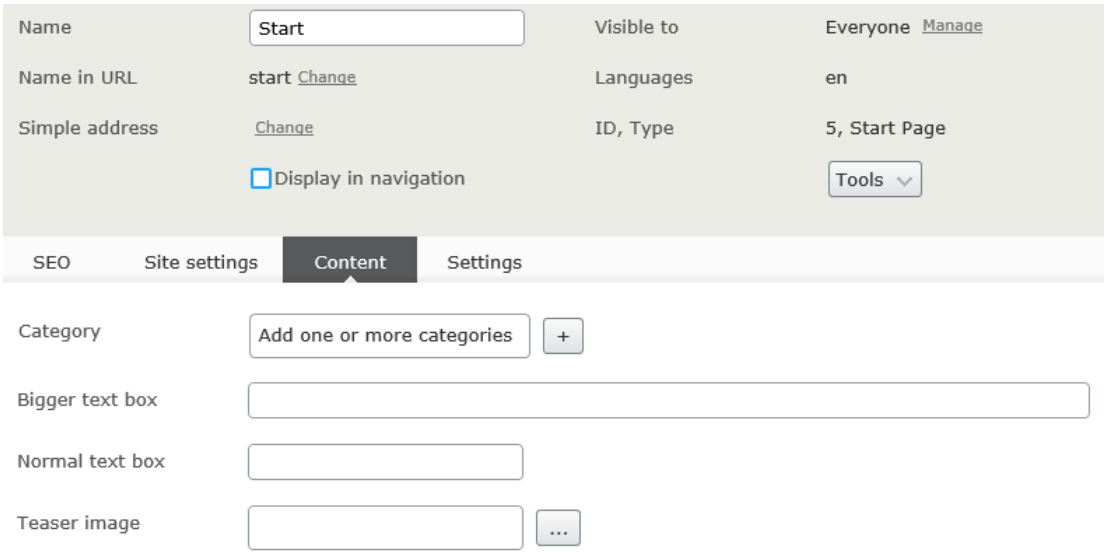
```
using EPiServer.Shell.ObjectEditing;
using EPiServer.Shell.ObjectEditing.EditorDescriptors;
using System;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    [EditorDescriptorRegistration(
        TargetType = typeof(string),
        UIHint = Global.SiteUIHints.Embiggen)]
    public class EmbiggenEditorDescriptor : EditorDescriptor
    {
        public override void ModifyMetadata(
            ExtendedMetadata metadata, IEnumerable<Attribute> attributes)
        {
            base.ModifyMetadata(metadata, attributes);

            metadata.EditorConfiguration.Add("style", "width: 600px");
        }
```

```
    }
}
```

5. Open **~/Models/Pages/StartPage.cs**.
6. Add two properties to the **StartPage** class, as shown in the following code:

```
[UIHint(Global.SiteUIHints.Embiggen)]
[Display(Name = "Bigger text box")]
public virtual string BiggerTextBox { get; set; }

[Display(Name = "Normal text box")]
public virtual string NormalTextBox { get; set; }
```

7. Start the site, and log in as **Admin**.
8. Switch to **Edit** view.
9. For the **Start** page, switch to **All Properties** view, and click on the **Content** tab. Note the two text boxes, one larger than the other, as shown in the following screenshot:



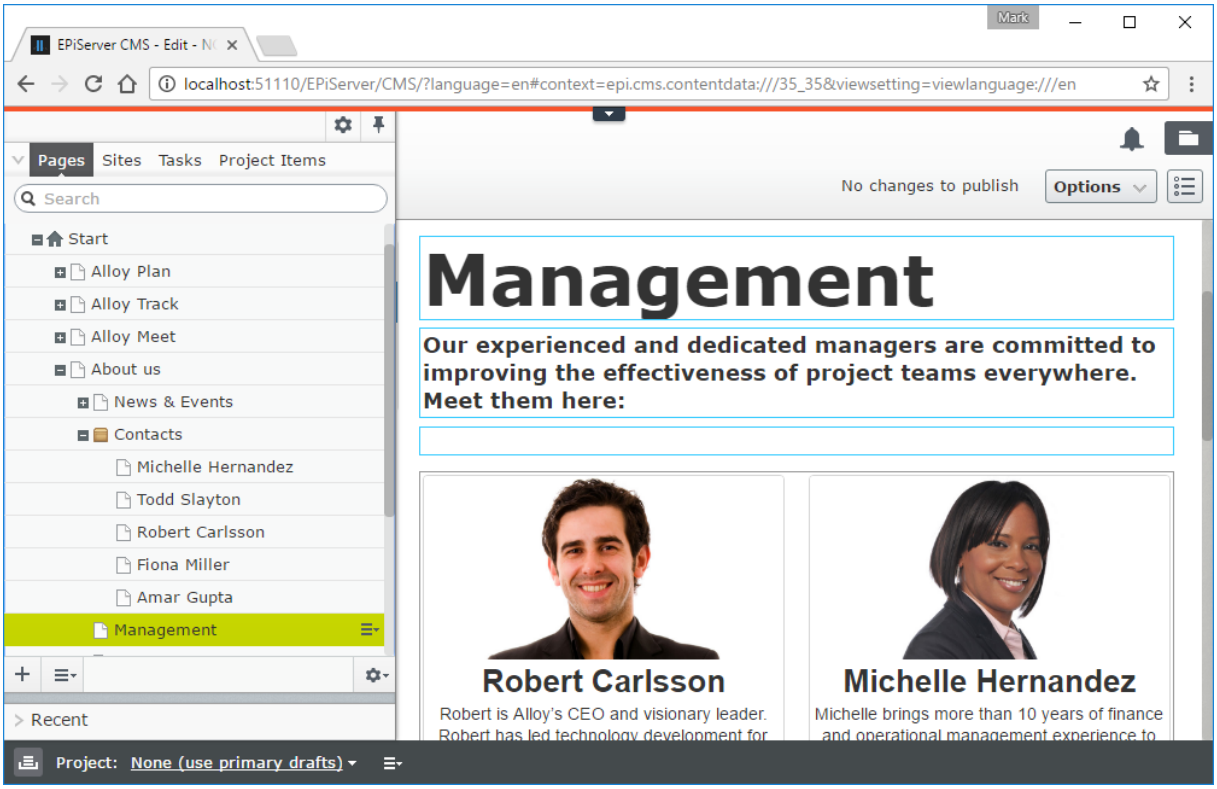
Exercise D2 – Selecting choices for property values

In this exercise, you will allow an editor to select choices for enums and strings by using either a drop-down list box or multiple check boxes.

Prerequisites: complete Exercise 0.1.

Reviewing existing contact pages

1. Open the solution with the **AlloyAdvanced** project.
2. Start the site, and log in as **Admin**.
3. Switch to **Edit** view.
4. Show the **Navigation** pane, and the **Pages** tree.
5. The Alloy site page tree has a container page named **Contacts** that contains five **ContactPage** instances. Each contact page has the following properties: an email address, an image, and a telephone number. They also inherit properties from **SitePageData** like **TeaserText**. The **Management** page under **About us** has a content area into which **ContactPage** instances have been added, as shown in the following screenshot:



6. Open **~/Views/Shared/PartialPages/ContactPage.cshtml** and **ContactPageWide.cshtml** to see how these pages are currently rendered.

```
@model ContactPage
<div class="border">
    <img src="@Url.ContentUrl(Model.PageImage)" />
    <h2>@Model.PageName</h2>
    <p>@Model.TeaserText</p>
    <p>
        @Html.Translate("/contact/email"): @Html.DisplayFor(x => x.Email)<br />
        @Html.Translate("/contact/phone"): @Model.Phone
    </p>
</div>
```

We might want contacts to have:

- a property to show their region,
- a favorite YouTube video from a limited list, and
- a home city, and other cities they frequently travel to.

A contact's region could be stored as an **enum** value, and cities could be stored as **string** values. To make life easier for editors, we want these choices to be set using a drop-down list or list of checkboxes of valid choices.

Defining an enum for regions and a list for Episerver YouTube videos

1. In **~/Models**, add a class named **Region**. Modify its code as follows.

```
namespace AlloyAdvanced.Models
{
    public enum Region
    {
        Unknown,     // 0
        Europe,      // 1
        UK,          // 2
        USA,         // 3
        RestOfWorld // 4
    }
}
```

2. In **~/Models**, add a class named **EpiserverYouTube**.
3. Modify its code as follows.

```
using EPiServer.Shell.ObjectEditing;
using System.Collections.Generic;

namespace AlloyAdvanced.Models
{
    public static class EpiserverYouTube
    {
        public static List<SelectItem> Videos = new List<SelectItem>();

        static EpiserverYouTube()
        {
            Videos.Add(new SelectItem
            {
                Value = "xbNRxExM-sY",
                Text = "Episerver Advanced CMS Authoring"
            });
            Videos.Add(new SelectItem
            {
                Value = "v8tWqYVArVY",
                Text = "Episerver Ascend Las Vegas 2017"
            });
            Videos.Add(new SelectItem
            {
                Value = "ErHS21Js0Do",
                Text = "Episerver Find"
            });
            Videos.Add(new SelectItem
            {
                Value = "8CJgklPCAiA",
                Text = "Episerver Forms"
            });
            Videos.Add(new SelectItem
            {
                Value = "vaGZGpQB394",
```

```
                Text = "Episerver PowerSlice"
            });
            Videos.Add(new SelectItem
            {
                Value = "Bf0eoyJv8xE",
                Text = "Episerver Projects"
            });
        }
    }
}
```

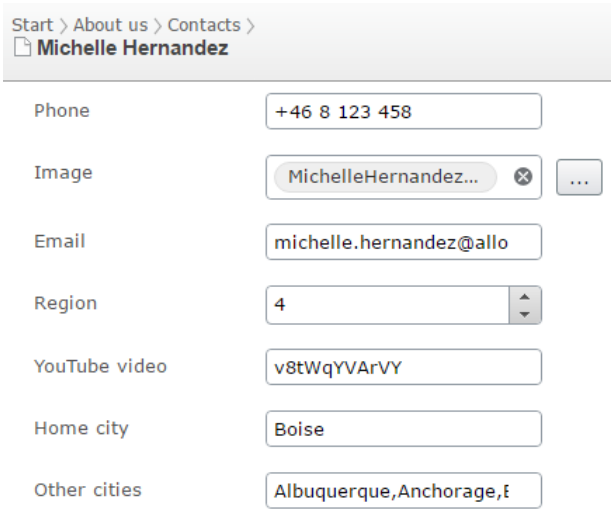
In the real world, you would load a list of videos from an XML or JSON file, or from a database, or call a web service. For this example, we hard-coded the list.

Adding properties for contacts

1. Open **~/Models/Pages/ContactPage.cs**.
2. Add new properties as shown in the following code:

```
[Display(
    Name = "Region",
    GroupName = Global.GroupNames.Contact,
    Order = 10)]
public virtual Region Region { get; set; }

[Display(
    Name = "YouTube video",
    GroupName = Global.GroupNames.Contact,
    Order = 20)]
public virtual string YouTubeVideo { get; set; }

 [Display(
    Name = "Home city",
    GroupName = Global.GroupNames.Contact,
    Order = 30)]
public virtual string HomeCity { get; set; }

[Display(
    Name = "Other cities",
    GroupName = Global.GroupNames.Contact,
    Order = 40)]
public virtual string OtherCities { get; set; }
```

3. Start the site, and log in as **Admin**.
4. Switch to Edit view.
5. In the **Navigation** pane's **Pages** tree, navigate to **Start** | **About us** | **Contacts** | **Michelle Hernadez**.
6. Note the editor would need to manually enter city names and a video identifier into a text box and the **Region** is treated as an integer (because that's how enum values are stored internally), as shown in the following screenshot:



This part of the exercise is to show default behavior for properties that use **string** and **enum** (**int**) data types. Next, you will improve the editor's experience by creating some custom selectors.

Creating a selector for enums

Let's start by changing the editor experience for the **Region** to provide a drop-down list box.

First, we must create an implementation of **ISelectionFactory** that returns the list of choices based on an enum.

For maximum reuse, this selection factory will be written to work with any enum, not just **Region**.

1. In **~/Business**, add a new folder named **Selectors**.
2. In **~/Business/Selectors**, add a new class named **EnumSelectionFactory.cs**.
3. Modify its code as shown in the following code:

Make sure that you use `EPiServer.Shell.ObjectEditing.ISelectionFactory`, not `EPiServer.Personalization.VisitorGroups.ISelectionFactory`

```
using System;
using System.Collections.Generic;
using EPiServer.Shell.ObjectEditing;

namespace AlloyAdvanced.Business.Selectors
{
    public class EnumSelectionFactory<TEnum> : ISelectionFactory
    {
        public IEnumerable<ISelectItem> GetSelections(
            ExtendedMetadata metadata)
        {
            var values = Enum.GetValues(typeof(TEnum));
            foreach (var value in values)
            {
```

```
                    yield return new SelectItem
                    {
                        Text = Enum.GetName(typeof(TEnum), value),
                        Value = value
                    };
                }
            }
        }
    }
```

Second, we will create an attribute that can be applied to a property to activate the list box for that property.

4. In **~/Business/Selectors**, add a new class named **SelectOneEnumAttribute.cs**.
5. Modify its code like this.

```
using System;
using System.Web.Mvc;
using EPiServer.Shell.ObjectEditing;

namespace AlloyAdvanced.Business.Selectors
{
    public class SelectOneEnumAttribute : SelectOneAttribute, IMetadataAware
    {
        public SelectOneEnumAttribute(Type enumType)
        {
            if (!enumType.IsEnum)
            {
                throw new ArgumentException("Type must be an enum.");
            }
            this.EnumType = enumType;
        }

        public Type EnumType { get; protected set; }

        public new void OnMetadataCreated(ModelMetadata metadata)
        {
            this.SelectionFactoryType = typeof(EnumSelectionFactory<>)
                .MakeGenericType(this.EnumType);
            base.OnMetadataCreated(metadata);
        }
    }
}
```

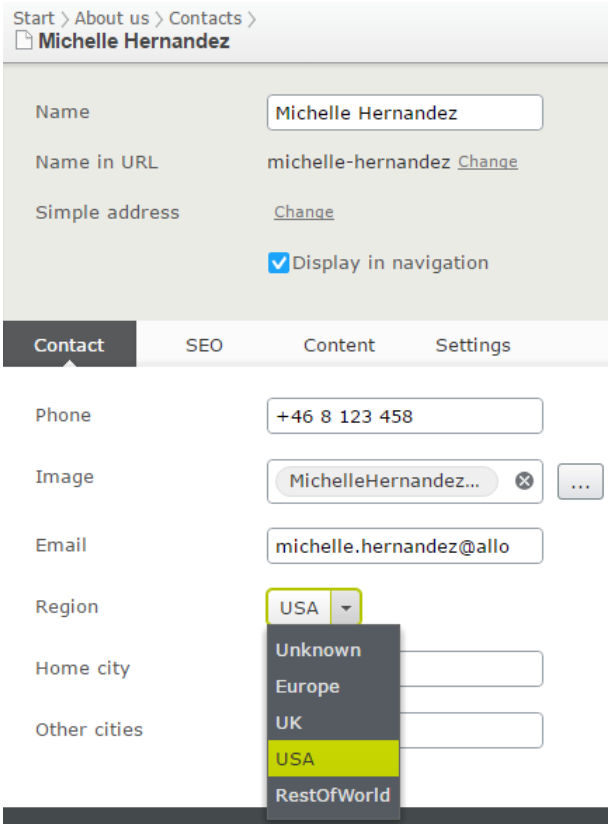
In the real world, you would want extensions like these to be stored in a class library or add-on to enable easier reuse. You will learn how to create add-ons in a later module.

Testing the enum selector

1. Open **~/Models/Pages/ContactPage.cs** and import the **AlloyAdvanced.Business.Selectors** namespace.
2. Apply your new attribute to the **Region** property like this.

```
[Display(
    Name = "Region",
    GroupName = Global.GroupNames.Contact,
    Order = 10)]
[SelectOneEnum(typeof(Region))]
public virtual Region Region { get; set; }
```

3. Start the site, and log in as **Admin**.
4. Edit one of the contacts. Now the editor can select from a drop-down list box as shown in the following screenshot.



Optional task: Allowing multiple regions

Create a `[SelectManyEnum]` attribute that allows the editor to select multiple regions.

You will need to modify the **Region** enum values so that their bits do not "overlap" and apply the `[Flags]` attribute, as shown in the following code:

```
namespace AlloyAdvanced.Models
{
    [System.Flags]
    public enum Region
    {
        Unknown = 0,
        Europe = 1,
        UK = 2,
        USA = 4,
        RestOfWorld = 8
    }
}
```

How to handle the editor's experience of the zero (0) **Unknown** choice is an interesting question. For a single selection, the drop-down list should show the **Unknown** choice. But for a multiple selection, it should not, because that choice would be indicated by *none* of the check boxes being selected.

Creating a selector for videos

1. In **Solution Explorer**, right-click **~/Business/Selectors**, and add a new class named **YouTubeSelectionFactory.cs**. Modify its code like this.

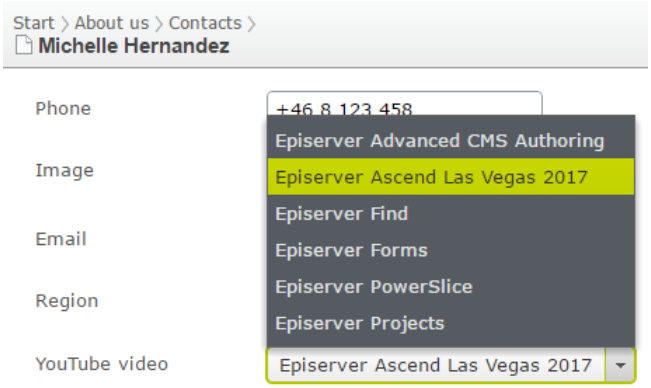
   ```
   using AlloyAdvanced.Models;
   ```

```
using EPiServer.Shell.ObjectEditing;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.Selectors
{
    public class YouTubeSelectionFactory : ISelectionFactory
    {
        public IEnumerable<ISelectItem> GetSelections(
            ExtendedMetadata metadata)
        {
            return EpiserverYouTube.Videos;
        }
    }
}
```

2. Open **~/Models/Pages/ContactPage.cs**. Apply the **[SelectOne]** attribute to the **YouTubeVideo** property using your YouTube selection factory, like this.

```
[Display(
    Name = "YouTube video",
    GroupName = Global.GroupNames.Contact,
    Order = 20)]
[SelectOne(SelectionFactoryType = typeof(YouTubeSelectionFactory))]
public virtual string YouTubeVideo { get; set; }
```

3. Start the site, and log in as **Admin**.
4. Edit one of the contacts. Now the editor can select the title of a YouTube video from a drop-down list box as shown in the following screenshot, and it will store the identifier of the video in the CMS database.



Creating selectors for cities

Prerequisites: complete Exercise 0.2.

1. In **Solution Explorer**, right-click **~/Business/Selectors**, and add a new class named **CitySelectionFactory.cs**.
2. Modify its contents as shown in the following code:

```
using AlloyAdvanced.Models.NorthwindEntities;
using EPiServer.Logging;
using EPiServer.Shell.ObjectEditing;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AlloyAdvanced.Business.Selectors
{
    public class CitySelectionFactory : ISelectionFactory
```

```
    {
        private static readonly ILogger logger =
            LogManager.GetLogger(typeof(CitySelectionFactory));

        private static List<SelectItem> list = null;

        public IEnumerable<ISelectItem> GetSelections(
            ExtendedMetadata metadata)
        {
            if (list == null)
            {
                list = new List<SelectItem>();
            }
            if (list.Count == 0)
            {
                var db = new Northwind();

                try
                {
                    list = db.Customers
                        .Select(customer => customer.City)
                        .Distinct()
                        .OrderBy(c => c)
                        .Select(city => new SelectItem
                        {
                            Text = city,
                            Value = city
                        })
                        .ToList();
                }
                catch (Exception ex)
                {
                    logger.Error($"{ex.GetType()}: {ex.Message}");
                }
            }
            return list;
        }
    }
}
```

Testing the city selector

1. Open **~/Models/Pages/ContactPage.cs**.
2. Apply the **[SelectOne]** attribute to the **HomeCity** property and the **[SelectMany]** attribute to the **OtherCities** property, both using your city selection factory, like this.

```
[Display(
    Name = "Home city",
    GroupName = Global.GroupNames.Contact,
    Order = 30)]
[SelectOne(SelectionFactoryType = typeof(CitySelectionFactory))]
public virtual string HomeCity { get; set; }

[Display(
    Name = "Other cities",
    GroupName = Global.GroupNames.Contact,
    Order = 40)]
[SelectMany(SelectionFactoryType = typeof(CitySelectionFactory))]
public virtual string OtherCities { get; set; }
```

3. Open **~/Views/Shared/PartialPages/ContactPage.cshtml**.

4. Modify its content to output the extra properties, as shown in the following code:

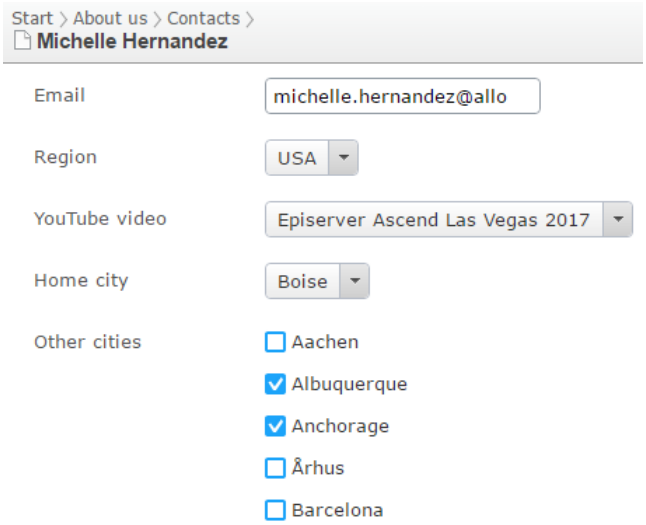
```
@model ContactPage
<div class="border">
    <img src="@Url.ContentUrl(Model.PageImage)" />
    <h2>@Model.PageName</h2>
    <p>@Model.TeaserText</p>
    <p>
        @Html.Translate("/contact/email"): @Html.DisplayFor(x => x.Email)<br />
        @Html.Translate("/contact/phone"): @Model.Phone<br />
        Region: @Html.DisplayFor(m => m.Region)<br />
        Video: @Html.DisplayFor(m => m.YouTubeVideo)<br />
        Home city: @Html.DisplayFor(m => m.HomeCity)<br />
        Other cities: @Html.DisplayFor(m => m.OtherCities)
    </p>
</div>
```

It would be better practice to localize the view using the **Html.Translate()** method, but for now you will hard-code the labels for the new propertes.

5. Do a similar modification to **ContactPageWide.cshtml**.

Testing the contact page properties

1. Start the site, and log in as **Admin**.
2. Edit Michelle Hernadez's contact page, as shown in the following screenshot:



3. Publish the changes and view the **About us** | **Management** page as a visitor, as shown in the following screenshot:

You will learn how to customize the rendering for the video and other cities in the exercises for Module E.

Although the Email property is a string, it has the **[Business.EmailAddress]** attribute that inherits from a Microsoft attribute named **DataTypeAttribute** with an enum value **DataType.EmailAddress**. When output in a view using `@Html.DisplayFor()`, this will render as a clickable `<a href="mailto:value">value</a>` element. When edited, it validates the string against a regular expression to ensure it's a valid email.

Creating a UIHint for city selection

1. Open **~/Global.cs**.
2. Add a string constant named **City** to the **SiteUIHints** class, as shown in the following code:

```
public static class SiteUIHints
{
    public const string City = "City";
    public const string Contact = "contact";
    public const string Strings = "StringList";
}
```

3. In **Solution Explorer**, in **~/Business/EditorDescriptors**, add a new class named **CityEditorDescriptor.cs**. Modify its code like this.

```
using AlloyAdvanced.Business.Selectors;
using EPiServer.Shell.ObjectEditing;
using EPiServer.Shell.ObjectEditing.EditorDescriptors;
using System;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    [EditorDescriptorRegistration(TargetType = typeof(string),
        UIHint = Global.SiteUIHints.City)]
    public class CityEditorDescriptor : EditorDescriptor
    {
        public override void ModifyMetadata(ExtendedMetadata metadata,
```

```
            IEnumerable<Attribute> attributes)
        {
            SelectionFactoryType = typeof(CitySelectionFactory);
            ClientEditingClass = "epi-cms/contentediting/editors/SelectionEditor";
            base.ModifyMetadata(metadata, attributes);
        }
    }
}
```

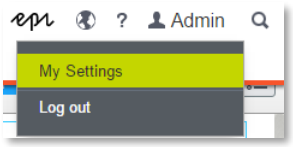
4. Open **~/Models/Pages/ContactPage.cs**. Comment out the **[SelectOne]** attribute for the **HomeCity** property. Apply the **[UIHint]** attribute instead, like this.

```
[Display(
    Name = "Home city",
    GroupName = Global.GroupNames.Contact,
    Order = 30)]
//[SelectOne(SelectionFactoryType = typeof(CitySelectionFactory))]
[UIHint(Global.SiteUIHints.City)]
public virtual string HomeCity { get; set; }
```

5. Start the site, and note that the same behavior applies.

Switching to Swedish

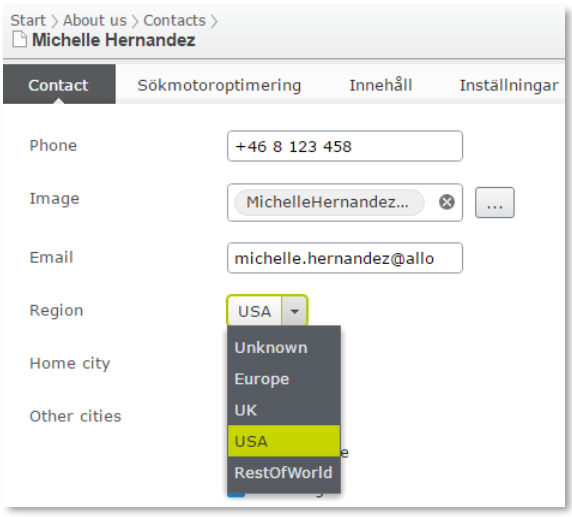
1. Open the solution with the **AlloyAdvanced** project.
2. In the **Global** menu, click **Admin**, and then **My Settings**, as shown:



3. Click the **Display Options** tab. The Admin user has their **Personal Language** set to **Use system language**, which on my laptop, is English. Change the language to **Svenska** and click **Save**.

Reviewing existing non-localization of the enum selection factory

1. Edit one of the contact pages and switch to **All Properties** view.
2. You will see that the **Region** property's drop-down list shows English words. It needs to be localized into Swedish. Even if you don't need to localize, it would be nicer if the choices had spaces in the names instead of showing the enums internal string values, like **RestOfWorld**, as shown in the following screensot:



Implementing localization for the enum selection factory

1. Open **~/Business/Selectors/EnumSelectionFactory.cs** and add the following imported namespace to the top of the file.

```
using EPiServer.Framework.Localization;
```

2. Inside the class, add a method named **GetLocalizedName**, as shown in the following code. Note that it uses the Episerver **LocalizationService** to try to load the localized string from a language file (if it exists), otherwise it fails back to the code text string.

```
// this method exists so that we can localize the string values
private string GetLocalizedName(object value)
{
    var staticName = Enum.GetName(typeof(TEnum), value);

    string localizationPath = string.Format(
        "/enum/{0}/{1}",
        typeof(TEnum).Name.ToLowerInvariant(),
        staticName.ToLowerInvariant());

    string localizedName;
    if (LocalizationService.Current.TryGetString(
        localizationPath,
        out localizedName))
    {
        return localizedName;
    }

    return staticName;
}
```

3. Inside the **GetSelections**() method, comment out the statement that converts an enum's value into a string representation, and replace it with a statement to call the new method to localize the value, as shown:

```
//Text = Enum.GetName(typeof(TEnum), value),
Text = GetLocalizedName(value),
```

Defining the localization text values for enums

1. In **~/Resources/LanguageFiles**, add a new XML file named **Enums.xml**. Modify it as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <enum>
      <region>
        <unknown>Unknown</unknown>
        <europe>Europe</europe>
        <uk>United Kingdom</uk>
        <usa>United States of America</usa>
        <restofworld>Rest of the World</restofworld>
      </region>
    </enum>
  </language>
</languages>
```

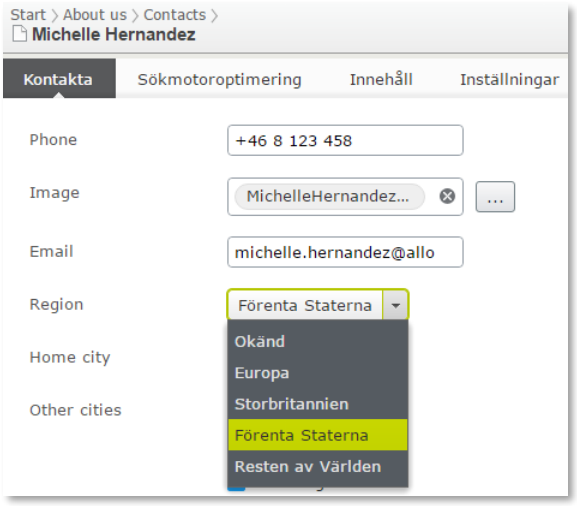
2. Copy and paste the file, and modify it as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="Svenska" id="sv">
    <enum>
```

```
        <region>
          <unknown>Okänd</unknown>
          <europe>Europa</europe>
          <uk>Storbritannien</uk>
          <usa>Förenta Staterna</usa>
          <restofworld>Resten av Världen</restofworld>
        </region>
      </enum>
    </language>
  </languages>
```

Testing the localization of the enum selection factory

1. Start the site, and log in as **Admin**.
2. Edit one of the contact pages and switch to **All Properties** view. You will see that the **Region** property's drop-down list shows Swedish words.



Exercise D3 – Using UIHints to select an editor

In this exercise, you will review a custom property editor that uses a selection factory for properties of type PageReference that allows the property to be set to an instance of a ContactPage using a drop-down list.

Prerequisites: complete Exercise 0.1.

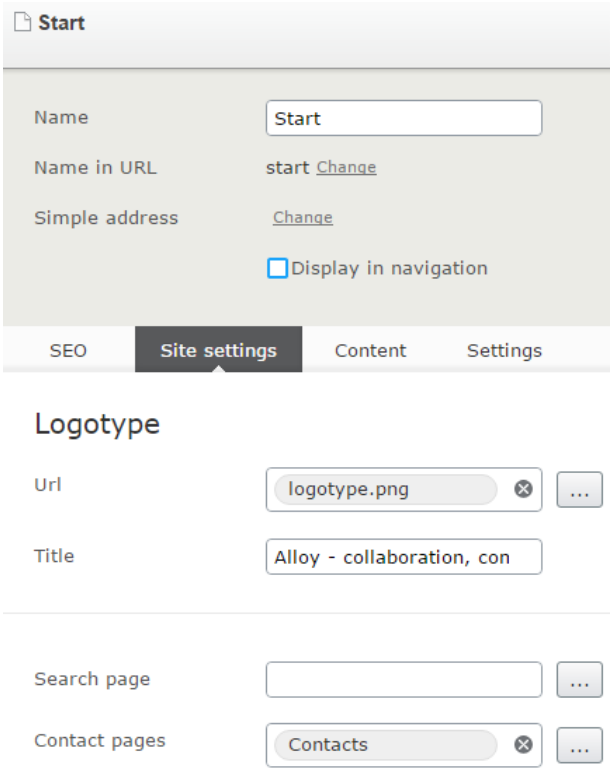
Now that you have seen how to implement the **ISelectionFactory** interface, let's review a more complex example that has already been implemented in the Alloy sample site.

Reviewing the existing functionality of the contact picker

You've already seen that the Alloy site has a container page with five instances of **ContactPage** within it. A property of the **StartPage** named **ContactsPageLink** references the container page named **Contacts**.

```
[Display(GroupName = Global.GroupNames.SiteSettings)]
public virtual PageReference ContactsPageLink { get; set; }
```

When editing the **Start** page, the property looks like the following screenshot:



Alloy has a selection factory that can be used to provide a drop-down list box of any of the **ContactPage** instances inside that container page.

1. Open **~/Business/EditorDescriptors/ContactPageSelectionFactory.cs**.

```
using System.Collections.Generic;
using System.Linq;
using EPiServer.ServiceLocation;
using EPiServer.Shell.ObjectEditing;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    public class ContactPageSelectionFactory : ISelectionFactory
    {
        private Injected<ContentLocator> ContentLocator { get; set; }

        public IEnumerable<ISelectItem> GetSelections(ExtendedMetadata metadata)
```

```
        {
            var contactPages = ContentLocator.Service.GetContactPages();

            return new List<SelectItem>(contactPages.Select(
                c => new SelectItem {Value = c.PageLink, Text = c.Name}));
        }
    }
}
```

The list for the drop-down is provided by the **ContactLocator** class.

2. Open **~/Business/ContentLocator.cs**. Its **GetContactPages** method looks like this.

```
public IEnumerable<ContactPage> GetContactPages()
{
    var contactsRootPageLink = _contentLoader.Get<StartPage>(
        SiteDefinition.Current.StartPage).ContactsPageLink;

    if (ContentReference.IsNullOrEmpty(contactsRootPageLink))
    {
        throw new ConfigurationErrorsException("No contact page root specified in site
settings, unable to retrieve contact pages");
    }

    return _contentLoader.GetChildren<ContactPage>(
        contactsRootPageLink).OrderBy(p => p.PageName);
}
```

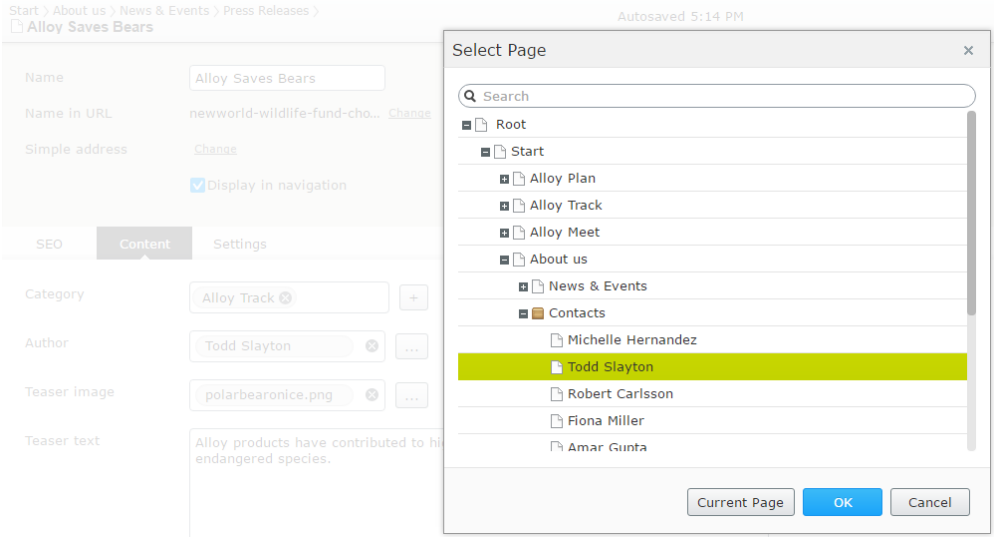
Applying the contact picker to a property

Let's try applying this selection factory to a property that needs to be set to a contact. We might want to assign each ArticlePage to a contact as the author of the article.

1. Open **~/Models/Pages/ArticlePage.cs**.
2. Add a new property named **Author**, as shown below.

```
public virtual PageReference Author { get; set; }
```

3. Start the site, and log in as **Admin**.
4. In **Pages**, navigate to **Start** | **About us** | **News & Events** | **Press Releases**.
5. Edit one of the articles, switch to **All Properties** view, click the **Content** tab, and note the **Author** property. Although an editor can use the built-in page selector to pick a contact page for the **Author**, as shown in the screenshot below, the experience could be improved:

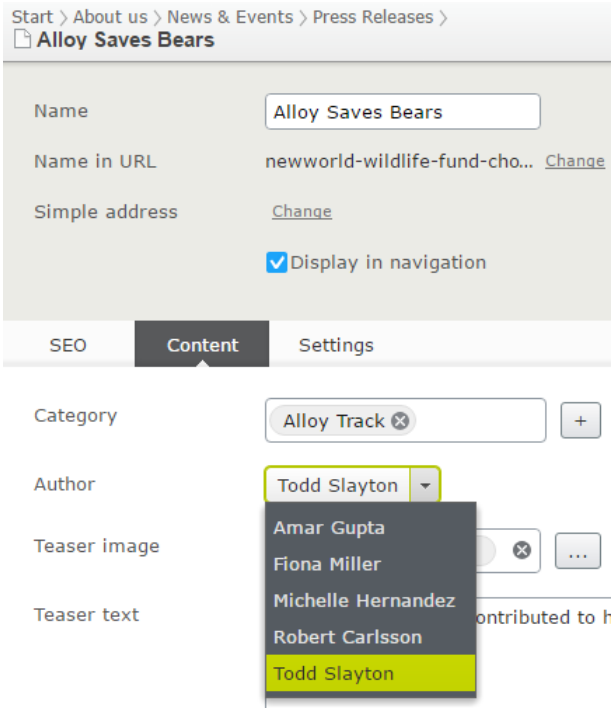


Modify the ArticlePage type

1. Open **~/Models/Pages/ArticlePage.cs**.
2. Apply the **[SelectOne]** attribute with the contact selection factory, to the property named **Author**, as shown below. You will also need to import the namespaces, as shown. The **IContentWithComments** interface was created and applied to the **ArticlePage** class in Module B exercises. If you did not complete that exercise, don't include that interface!

```
using AlloyAdvanced.Business.EditorDescriptors;
using EPiServer.Core;
using EPiServer.Shell.ObjectEditing;

namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(
        GroupName = Global.GroupNames.News,
        GUID = "AEECADF2-3E89-4117-ADEB-F8D43565D2F4")]
    [SiteImageUrl(Global.StaticGraphicsFolderPath +
        "page-type-thumbnail-article.png")]
    public class ArticlePage : StandardPage, IContentWithComments
    {
        [SelectOne(SelectionFactoryType = typeof(ContactPageSelectionFactory))]
        public virtual PageReference Author { get; set; }
    }
}
```

3. Start the site, and log in as **Admin**.
4. View one of the articles, switch to **Edit** view, switch to **All Properties** view, and click the **Content** tab.



Applying UI hints

An alternative to applying the **[SelectOne]** attribute with a selection factory is to apply the **[UIHint]** attribute and associate it with an editor descriptor.

1. Open **~/Models/Pages/ArticlePage.cs**.
2. Comment out the **[SelectOne]** attribute.

3. Add the **[UIHint]** attribute to the property named **Author**.
4. You will also need to import some different namespaces, as shown in the following code:

```
using EPiServer.Core;
using EPiServer.Web;
using System.ComponentModel.DataAnnotations;

namespace AlloyAdvanced.Models.Pages
{
    [SiteContentType(
        GroupName = Global.GroupNames.News,
        GUID = "AEECADF2-3E89-4117-ADEB-F8D43565D2F4")]
    [SiteImageUrl(Global.StaticGraphicsFolderPath
        + "page-type-thumbnail-article.png")]
    public class ArticlePage : StandardPage, IContentWithComments
    {
        //[SelectOne(SelectionFactoryType = typeof(ContactPageSelectionFactory))]
        [UIHint(Global.SiteUIHints.Contact)]
        public virtual PageReference Author { get; set; }
    }
}
```

If you start the site and edit an article, you will discover that the same behavior is used. How does this work? The **[UIHint]** attribute simply applies a "magic string" to a model's property.

5. Open **~/Global.cs**. It has a static **SiteUIHints** class with some string constants defined.

```
public static class SiteUIHints
{
    public const string Contact = "contact";
    public const string Strings = "StringList";
}
```

6. Open **~/Business/EditorDescriptors/ContactPageSelector.cs**, and note:

- The registration of this editor descriptor associates the "magic string" and restricts its application to properties of type PageReference.
- The ModifyMetadata method sets the selection factory.

```
using System;
using System.Collections.Generic;
using EPiServer.Core;
using EPiServer.Shell.ObjectEditing;
using EPiServer.Shell.ObjectEditing.EditorDescriptors;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    [EditorDescriptorRegistration(TargetType = typeof(PageReference),
        UIHint = Global.SiteUIHints.Contact)]
    public class ContactPageSelector : EditorDescriptor
    {
        public override void ModifyMetadata(
            ExtendedMetadata metadata, IEnumerable<Attribute> attributes)
        {
            SelectionFactoryType = typeof(ContactPageSelectionFactory);

            ClientEditingClass = "epi-cms/contentediting/editors/SelectionEditor";

            base.ModifyMetadata(metadata, attributes);
        }
    }
}
```

Exercise D4 – Customize any property at runtime using EditorDescriptors

In this exercise, you will hide unused properties such as Category using an EditorDescriptor.

Prerequisites: complete Exercise 0.1.

First, let's explore how properties interact with editor descriptors by creating one that outputs information about every property of a certain type.

Creating a property logger

1. Open the solution with the **AlloyAdvanced** project.
2. In **~/Business/EditorDescriptors**, add a new class named **PropertyLoggerEditorDescriptor**.
3. Modify it like this.

```
using EPiServer.Shell.ObjectEditing.EditorDescriptors;
using System;
using System.Collections.Generic;
using EPiServer.Shell.ObjectEditing;
using System.Diagnostics;
using EPiServer.Core;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    [EditorDescriptorRegistration(TargetType = typeof(int?))]
    [EditorDescriptorRegistration(TargetType = typeof(string))]
    [EditorDescriptorRegistration(TargetType = typeof(PageReference))]
    [EditorDescriptorRegistration(TargetType = typeof(CategoryList))]
    public class PropertyLoggerEditorDescriptor : EditorDescriptor
    {
        public override void ModifyMetadata(ExtendedMetadata metadata,
            IEnumerable<Attribute> attributes)
        {
            try
            {
                Debug.WriteLine(
                    $"Property: {metadata.PropertyName,-30} " +
                    $"Display: {metadata.ShowForDisplay,-5} " +
                    $"Edit: {metadata.ShowForEdit,-5} " +
                    $"Tab: {metadata.GroupName,-14} " +
                    $"Order: {metadata.Order,3} " +
                    $"Parent: {(metadata.Parent.Model as IContent).Name,-20} " +
                    $"Type: {metadata.ModelType.Name,-15}");
            }
            catch { }
            base.ModifyMetadata(metadata, attributes);
        }
    }
}
```

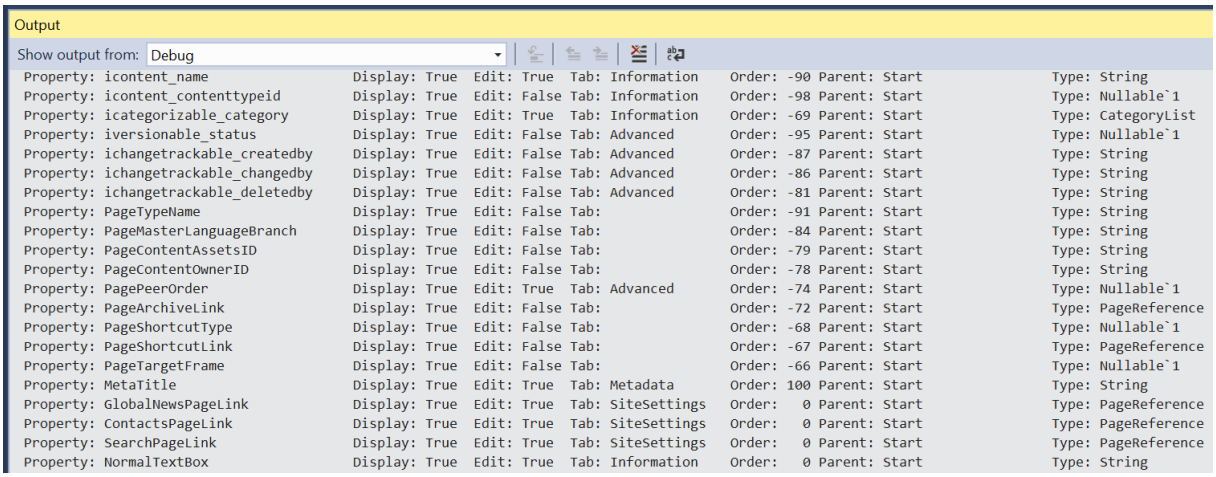
Note:

- This editor descriptor is registered to execute for any property that uses the following types: string, nullable int, CategoryList, or PageReference.
- For each property, the following is output: name, type, tab, order, content owner, if it should be shown for display and for editing.

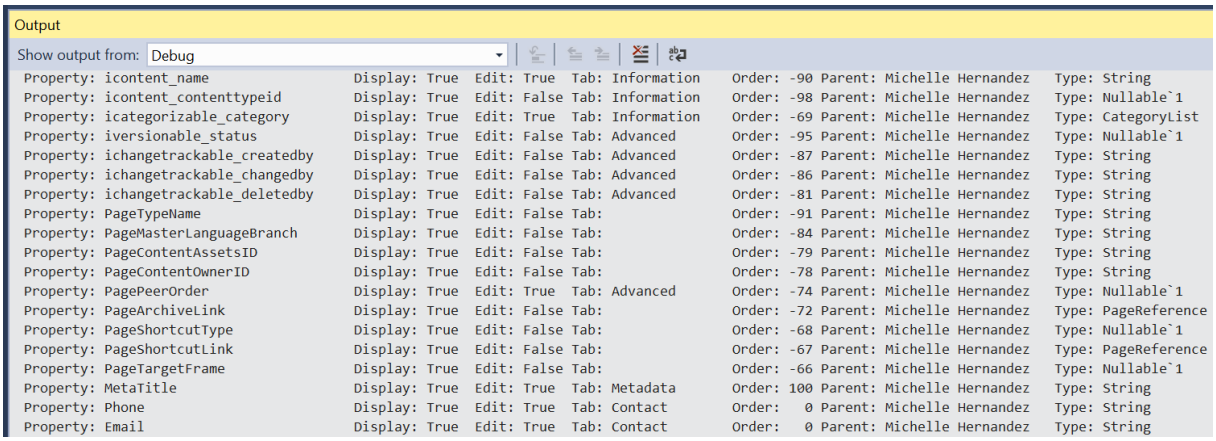
Viewing the logged output

1. Start the site by navigating to **Debug** | **Start Debugging**, or press *F5*.

2. Log in as **Admin**, and edit the **Start** page.
3. In Visual Studio's **Output**, you will see output like that shown in the following screenshot:



4. Edit one of the contact pages.
5. In Visual Studio's **Output**, you will see output like that shown in the following screenshot:



Note:

- The property named **icategorizable_category** has a type of **CategoryList** and currently has its **ShowForEdit** set to True.
- The property named **PagePeerOrder** has a type of **int?** and is currently on the **Advanced** aka **Settings** tab. Note: The **Content** tab used to be named **Information**, and the **Settings** tab used to be named **Advanced**. Internally the "magic strings" are the old values!
- Any of these properties could be programmatically modified at runtime to change their behavior.

Hiding the Category property

All pages and blocks inherit the **Category** property and appears at the top of the **Content** tab in **All Properties** view. If your site doesn't use this feature, you now know how to hide it.

1. Right-click **~/Business/EditorDescriptors** and add a new class named **HideCategoryEditorDescriptor**. Modify it like this.

```
using EPiServer.Core;
using EPiServer.Shell.ObjectEditing;
using EPiServer.Shell.ObjectEditing.EditorDescriptors;
using System;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.EditorDescriptors
```

```
{
    [EditorDescriptorRegistration(TargetType = typeof(CategoryList))]
    public class HideCategoryEditorDescriptor : EditorDescriptor
    {
        public override void ModifyMetadata(ExtendedMetadata metadata,
            IEnumerable<Attribute> attributes)
        {
            if (metadata.PropertyName == "icategorizable_category")
            {
                metadata.ShowForEdit = false;
            }
            base.ModifyMetadata(metadata, attributes);
        }
    }

}
```

2. It is bad practice to use a "magic string" in code, so instead of hard-coding the **icategorizable_category** property name, you could add a new set of string constants inside the **Global** class like this:

```
public static class SystemPropertyNames
{
    [Display(Name = "Category", Order = 1)]
    public const string Category = "icategorizable_category";

    [Display(Name = "SortIndex", Order = 2)]
    public const string SortIndex = "PagePeerOrder";
}
```

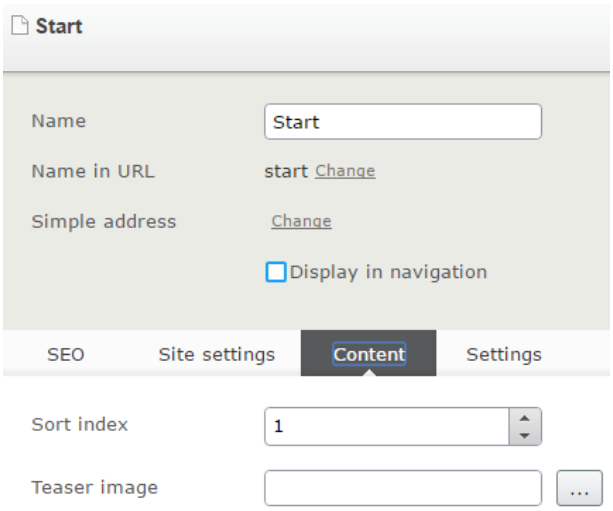
3. Then, you can use the following if-statement in the editor descriptor:

```
if (metadata.PropertyName == Global.SystemPropertyNames.Category)
```

4. Start the site, and log in as **Admin**.
5. Edit any page or block, switch to **All Properties** view, and click the **Content** tab. The Category property has gone!

Optional task: Moving the Sort index

Create a similar edit descriptor to move the **Sort index** property (aka PagePeerOrder) from the **Settings** tab to the **Content** tab, as shown in the screenshot below.



Exercise D5 – Create a custom editing experience for date-only pickers using Dojo

In this exercise, you will swap a built-in editing experience for a more appropriate one.

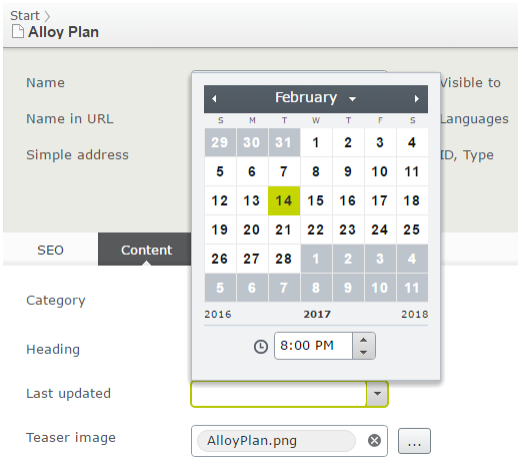
Prerequisites: complete Exercise 0.1.

Reviewing the date and time picker

1. Open the solution with the **AlloyAdvanced** project.
2. In **~\Models\Pages**, open **ProductPage.cs**.
3. Add a property to store when the product was last updated, as shown in the following code:

```
[Display(
    Name = "Last updated",
    GroupName = SystemTabNames.Content,
    Order = 20)]
public virtual System.DateTime LastUpdated { get; set; }
```

4. Start the site, and log in as **Admin**.
5. Edit the **Alloy Plan** page, switch to **All Properties** view, and click the **Content** tab.
6. When you change the **Last updated** property, a Dojo date and time picker is used, as shown in the following screenshot:



Most of the default **dijits** (Dojo "form widgets") can be swapped in as replacements for compatible Episerver properties. All you need to do is to set the proper **ClientEditingClass**. Read about all the available "form widgets": http://dojotoolkit.org/reference-guide/1.10/dijit/form.html#dijit-form

Creating a date-only picker

1. Open **~/Global.cs**.
2. Add a string constant named **DateOnly** to the **SiteUIHints** class, as shown in the following code:

```
public static class SiteUIHints
{
    public const string City = "City";
    public const string Contact = "contact";
    public const string DateOnly = "dateonly";
    public const string Strings = "StringList";
}
```

3. In **~\Business\EditorDescriptors**, add a class named **DateOnlyEditorDescriptor.cs**.

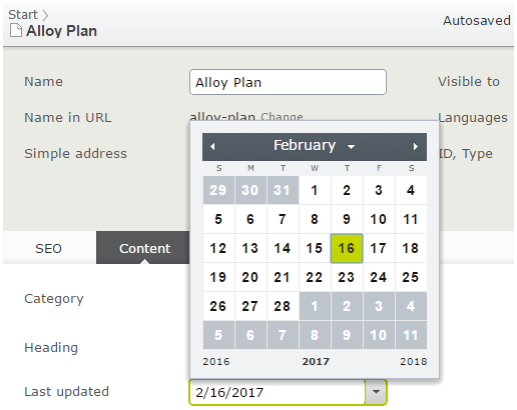
4. Modify the statements as shown in the following code:

```
using EPiServer.Shell.ObjectEditing;
using EPiServer.Shell.ObjectEditing.EditorDescriptors;
using System;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.EditorDescriptors
{
    [EditorDescriptorRegistration(TargetType = typeof(DateTime),
        UIHint = Global.SiteUIHints.DateOnly)]
    [EditorDescriptorRegistration(TargetType = typeof(DateTime?),
        UIHint = Global.SiteUIHints.DateOnly)]
    public class DateOnlyEditorDescriptor : EditorDescriptor
    {
        public override void ModifyMetadata(ExtendedMetadata metadata,
            IEnumerable<Attribute> attributes)
        {
            ClientEditingClass = "dijit/form/DateTextBox";
            base.ModifyMetadata(metadata, attributes);
        }
    }
}
```

5. Open **~\Models\Pages\ProductPage.cs**.
6. Apply `[UIHint]` to the **LastUpdated** property to activate the editor descriptor that you just created, as shown in the following code:

```
[Display(
    Name = "Last updated",
    GroupName = SystemTabNames.Content,
    Order = 20)]
[UIHint(Global.SiteUIHints.DateOnly)]
public virtual System.DateTime LastUpdated { get; set; }
```

7. Start the site, and log in as **Admin**.
8. Edit the **Alloy Plan** page.
9. Modify the **Last updated** property, and note that the time part is now hidden, as shown in the following screenshot:



Read more about Dojo

http://marisks.net/2014/04/11/episerver-writing-dojo-widget/

https://robertlinde.se/episerver%207.5/plugin/custom%20property/dojo/2014/05/10/custom-episerver-properties-with-dojo.html

https://andersnordby.wordpress.com/2014/10/24/creating-a-custom-property-with-a-dojo-widget/

https://www.david-tec.com/2014/08/EPiServer-Dojo-Useful-links/

http://azanganeh.com/blog/2016/10/30/episerver-custom-property-in-simple-steps-by-step-example/

Module E – Rendering, Personalizing, and Indexing Content

Goal

The overall goal of the exercises in this module is to customize how content is rendered to the visitor in the HTTP response using a display channel, personalizing responses using a custom visitor group criterion, and customizing indexed search.

You will:

- Create display templates that will be used when rendering if a UI hint is applied to a property.
- Create a display channel to render content in Adobe Acrobat PDF format.
- Create a visitor group criteria for cookie detection.
- Include additional fields to be indexed by Episerver Search.

Exercise E1 – Using UIHints to select display templates

In this exercise, you will define a UIHint to apply a display template.

Prerequisites: complete Exercise 0.1.

In Exercise D5, you saw how the **[UIHint]** attribute can be applied to associate an editor descriptor with a property. This applies at **edit** time to improve the experience for content editors.

UIHints are also read when rendering a property for the visitor, i.e. they also apply at **display** time to improve the experience for visitors.

Turn back to page 111 to see how the **About us** | **Management** page looks to visitors before you make the following changes.

Creating the global constants for UI hints

1. Open the solution with the **AlloyAdvanced** project.
2. Open **~/Global.cs**.
3. Add string constants named **Cities**, **Email**, and **YouTube** to the **SiteUIHints** class, as shown in the following code:

```
public static class SiteUIHints
{
    public const string City = "city";
    public const string Contact = "contact";
    public const string DateOnly = "dateonly";
    public const string Strings = "StringList";
    public const string Cities = "Cities";
    public const string Email = "Email";
    public const string YouTube = "YouTube";
}
```

Creating the display templates

1. In **Solution Explorer**, right-click **~/Views/Shared/DisplayTemplates**, and add a new **Block Partial View (MVC Razor)** named **Email.cshtml**. Modify its code like this.

```
@model string
@if (!string.IsNullOrWhiteSpace(Model))
{
    <a class="label label-info" style="color: white;" href="@Model">@Model</a>
```

```
}
```

2. In **Solution Explorer**, right-click **~/Views/Shared/DisplayTemplates**, and add a new **Block Partial View (MVC Razor)** named **YouTube.cshtml**. Modify its code like this.

```
@model string
@if (!string.IsNullOrWhiteSpace(Model))
{
    <iframe width="420" height="315"
            src="//www.youtube.com/embed/@Model?rel=0"
            frameborder="0" allowfullscreen></iframe>
}
```

3. In **Solution Explorer**, right-click **~/Views/Shared/DisplayTemplates**, and add a new **Block Partial View (MVC Razor)** named **Cities.cshtml**. Modify its code like this.

```
@model string
@if (!string.IsNullOrWhiteSpace(Model))
{
    string[] cities = Model.Split(',');
    foreach (string item in cities)
    {
        <span class="label">@item</span>
    }
}
```

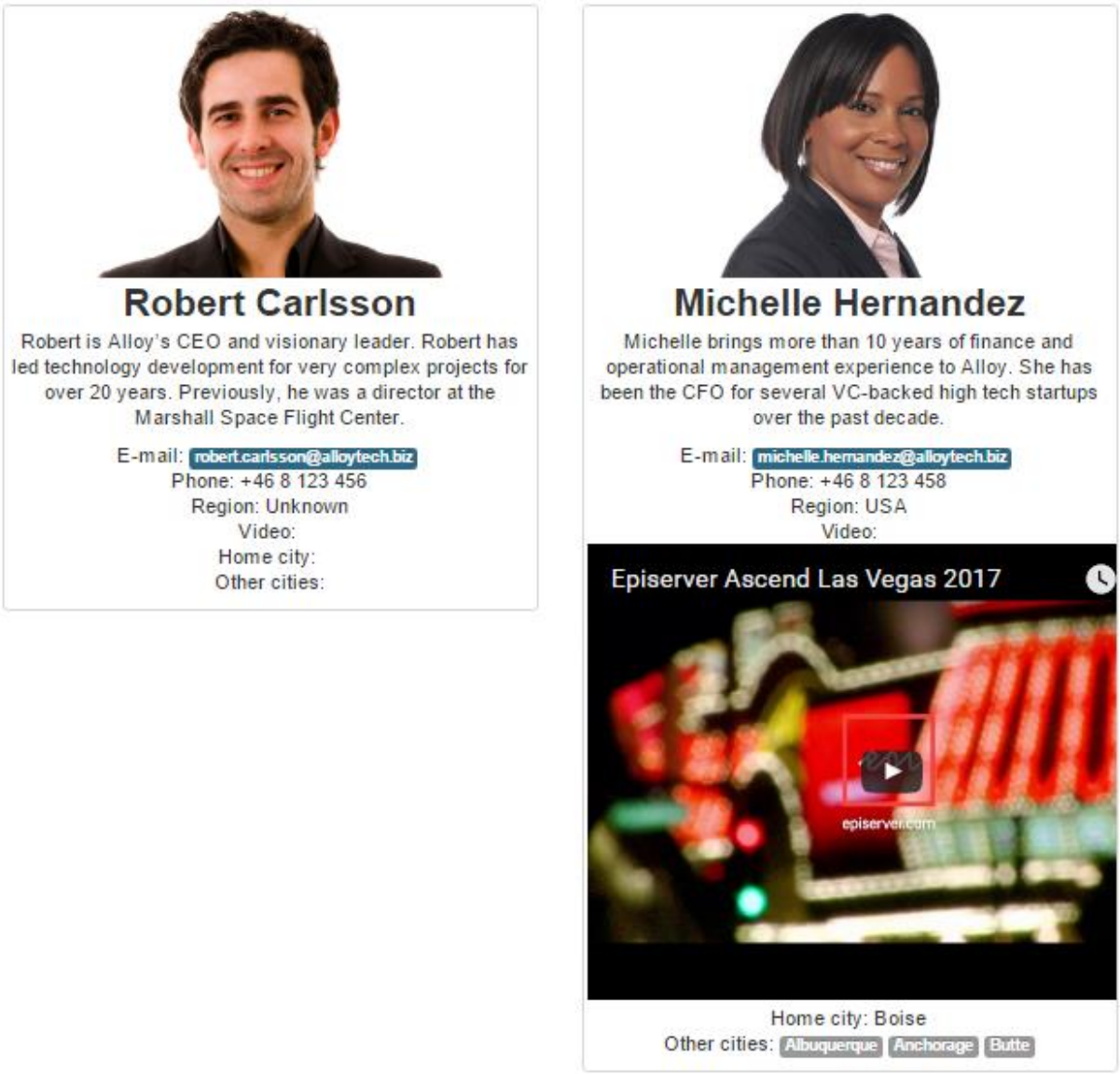
Applying the UI hints to content properties

1. Open **~/Models/Pages/ContactPage.cs**.
2. Apply the **[UIHint]** attribute to the **YouTubeVideo** property, like this.

```
[Display(GroupName = Global.GroupNames.Contact)]
[Business.EmailAddress]
[UIHint(Global.SiteUIHints.Email)]
public virtual string Email { get; set; }


[Display(
    Name = "YouTube video",
    GroupName = Global.GroupNames.Contact,
    Order = 20)]
[SelectOne(SelectionFactoryType = typeof(YouTubeSelectionFactory))]
[UIHint(Global.SiteUIHints.YouTube)]
public virtual string YouTubeVideo { get; set; }


[Display(
    Name = "Other cities",
    GroupName = Global.GroupNames.Contact,
    Order = 40)]
[SelectMany(SelectionFactoryType = typeof(CitySelectionFactory))]
[UIHint(Global.SiteUIHints.Cities)]
public virtual string OtherCities { get; set; }
```

Testing the UI hints

1. Start the site.
2. As a visitor, navigate to **About us** | **Management**, as shown in the following screenshot, and note the difference in the email links, the rendering of the videos, and the rendering of the list of other cities:



Exercise E2 – Creating a PDF display channel

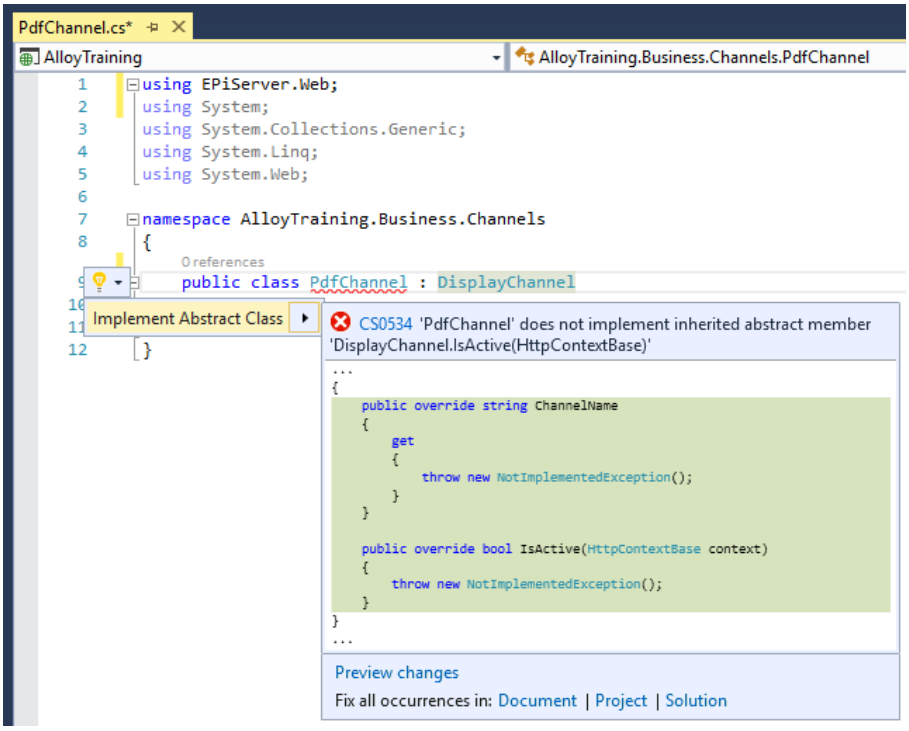
In this exercise, you will learn how to add a channel and how to work with DisplayChannelService to see if a channel is active.

Prerequisites: complete Exercise 0.1.

The scenario is to add a PDF channel that will be used for the product pages. In the Product Page template a link, **View as PDF**, will be added. When a visitor clicks the link, the display channel will be activated and create the PDF file "on the fly".

Creating the display channel

1. Open the solution with the **AlloyAdvanced** project.
2. Add a new class to the **~/Business/Channels** folder named **PdfChannel**.
3. Import the **EPiServer.Web** namespace.
4. Make **PdfChannel** inherit from **DisplayChannel**.
5. Use Visual Studio to implement the abstract class, as shown in the following screenshot:



6. Implement the **ChannelName** property by returning **"PDF"**.
7. Implement the **IsActive** property by returning **true** if a query string parameter named **pdf** exists, or return **false** otherwise.

The completed class should look like this:

```
using EPiServer.Web;
using System.Web;

namespace AlloyAdvanced.Business.Channels
{
    public class PdfChannel : DisplayChannel
    {
        public override string ChannelName
        {
            get
            {
```

```
                return "PDF";
            }
        }

        public override bool IsActive(HttpContextBase context)
        {
            if (context == null) return false;

            return (context.Request.QueryString["pdf"] != null);
        }
    }
}
```

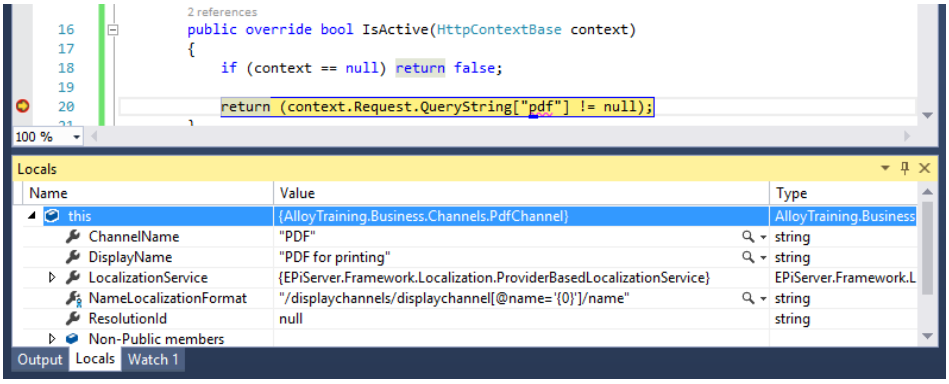
Localizing the display channel

You will learn more details about localizing for content editors in Module F. For now, just follow the steps.

1. Add an XML file to the **~/Resources/LanguageFiles** folder named **DisplayChannels.xml**.

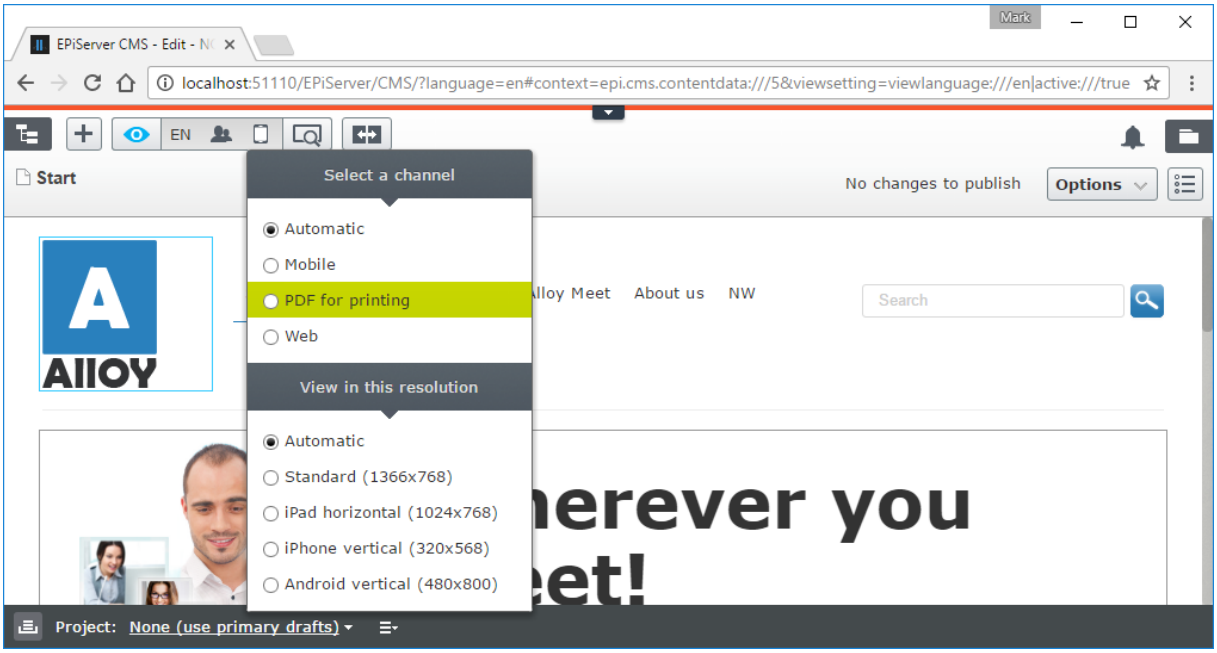
```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <displaychannels>
      <displaychannel name="PDF">
        <name>PDF for printing</name>
      </displaychannel>
    </displaychannels >
  </language>
  <language name="Svenska" id="sv">
    <displaychannels>
      <displaychannel name="PDF">
        <name>PDF för utskrift</name>
      </displaychannel>
    </displaychannels >
  </language>
</languages>
```

2. Set a break point in the **IsActive** property of the **PdfChannel** class.
3. Start the site.
4. When the breakpoint is hit, you see why the XML file must be structured in the way it is. Display channels have a property named **NameLocalizationFormat** that is an XPath expression that looks for the display name to use, as you can see in the following screenshot:



5. Start the site, and log in as **Admin**.
6. Edit any page.

7. In the toolbar, click **Toggle view settings** (it looks like an eye).
8. Click the **View channel and resolution options** button (it looks like a mobile phone), and note that your display channel is registered and reading its label from the localization XML file, as shown in the screenshot below:



9. Close the browser.

Installing the iTextSharp package

1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. Set **Package Source** to **All**, and enter the following command:

```
Install-Package -ProjectName AlloyAdvanced iTextSharp
```

Creating a link for PDFs

1. Open **~/Views/ProductPage/Index.cshtml**.
2. At the bottom of the file, but above the rendering of the **RelatedContentArea** property, add a `<div>` element containing a `<span>` for a print icon, and a **View as PDF** link, as shown in the following code:

```
    <div class="well-large" style="background-color: peachpuff">
        <span class="icon-print"></span>
        @Html.ActionLink("View as PDF", "Index", new { pdf = "" })
    </div>

    @Html.PropertyFor(x => x.CurrentPage.RelatedContentArea,
        new { CssClass = "row", Tag = Global.ContentAreaTags.OneThirdWidth })
}
```

Creating a controller for PDFs

1. In the **~/Business/Channels** folder, add an existing file from the solutions ZIP: **~/Solutions/E2/Business/Channels/PDFChannelHelper.cs**.
2. In the **~/Controllers** folder, add an **Episerver** | **Page Controller (MVC)** named **ProductPageAsPdfController**.

When creating controllers in the Alloy site, you should inherit from the site-specific class named `PageControllerBase<T>`, instead of the usual Episerver class named `PageController<T>`.

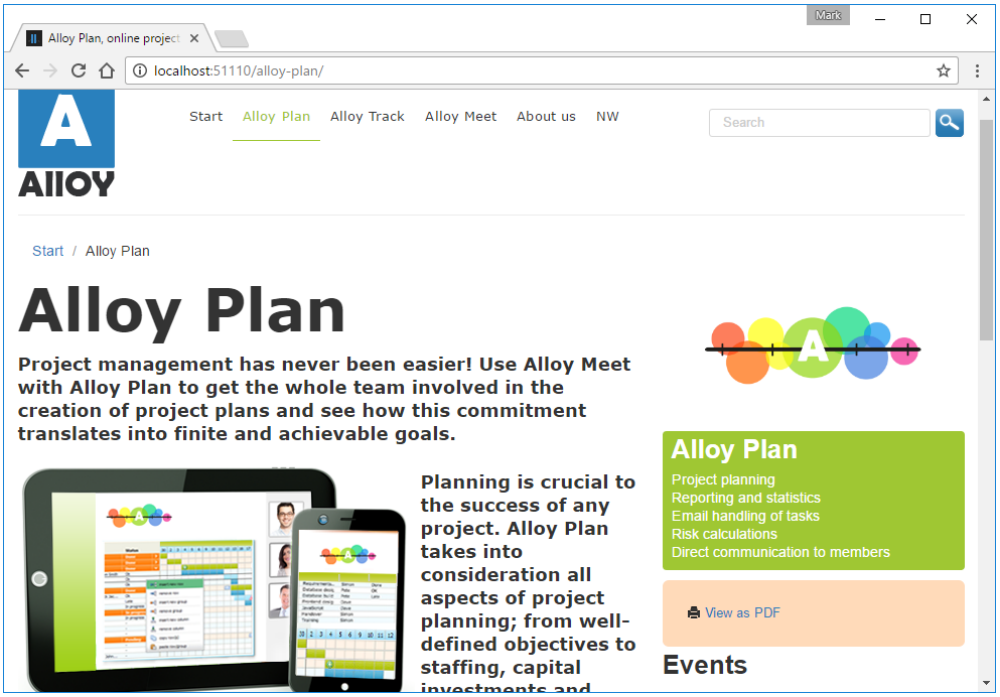
3. Modify the controller, as shown in the following code:

```
using System.Web.Mvc;
using EPiServer.Framework.DataAnnotations;
using EPiServer.Web.Mvc;
using AlloyAdvanced.Models.Pages;
using System.Text;
using AlloyAdvanced.Models.ViewModels;
using AlloyAdvanced.Business.Channels;

namespace AlloyAdvanced.Controllers
{
    // the Tag should match the ChannelName of the DisplayChannel
    [TemplateDescriptor(Inherited = true, Tags = new[] { "PDF" })]
    public class ProductPageAsPdfController : PageControllerBase<ProductPage>
    {
        public ActionResult Index(ProductPage currentPage)
        {
            // create HTML to send to PDF
            var sb = new StringBuilder();
            sb.Append($"<h1>{currentPage.Name}</h1>");
            sb.Append($"<h3>{currentPage.MetaDescription}</h3>");
            sb.Append(currentPage.MainBody.ToString());

            // generate the PDF and send directly to the response stream
            PDFChannelHelper.GeneratePDF(sb.ToString(), currentPage.Name);

            var model = PageViewModel.Create(currentPage);

            return View(model);
        }
    }
}
```

Viewing the PDFs

1. Start the site.

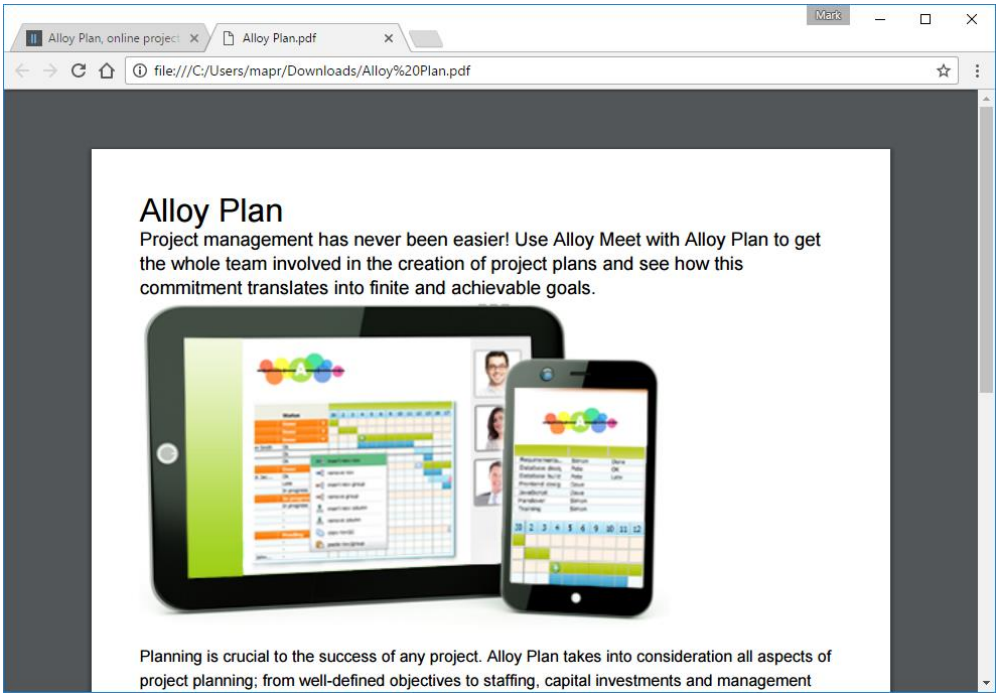
2. View the **Alloy Plan** product page as a visitor, as shown in the following screenshot:



3. Click **View as PDF**.

4. In Chrome, you would see the following at the bottom of the browser window:

5. Click the PDF file to open it, as shown in the following screenshot:



Exercise E3 – Detecting visitor groups with cookies

In this exercise, you will create an add-on that defines a visitor group criteria to detect cookies.

Prerequisites: complete Exercise 0.1.

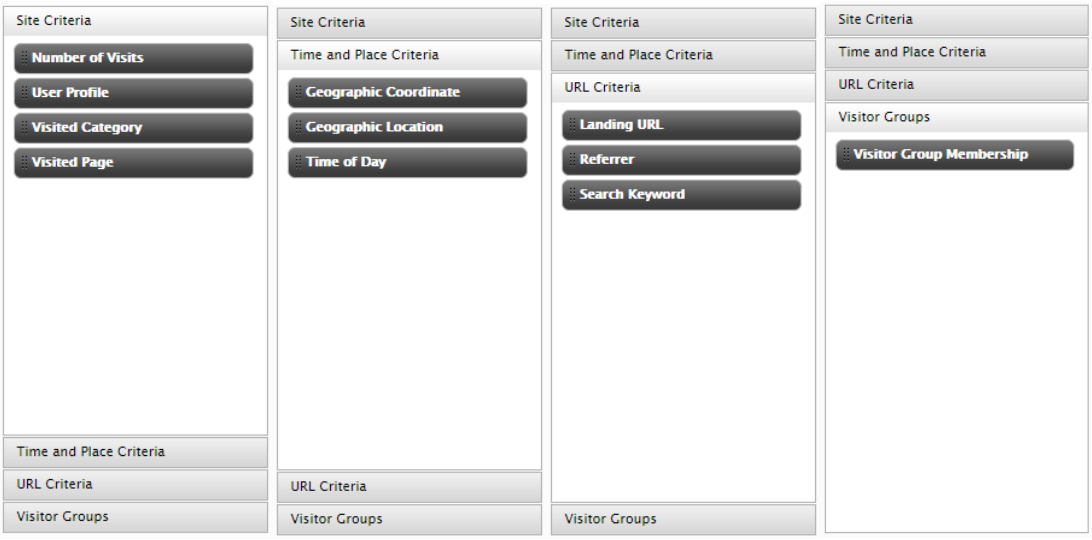
Exploring the Visitor Groups Criteria Pack

Before creating your own visitor group criterion, you will review an existing set.

1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
3. Enter the following command:

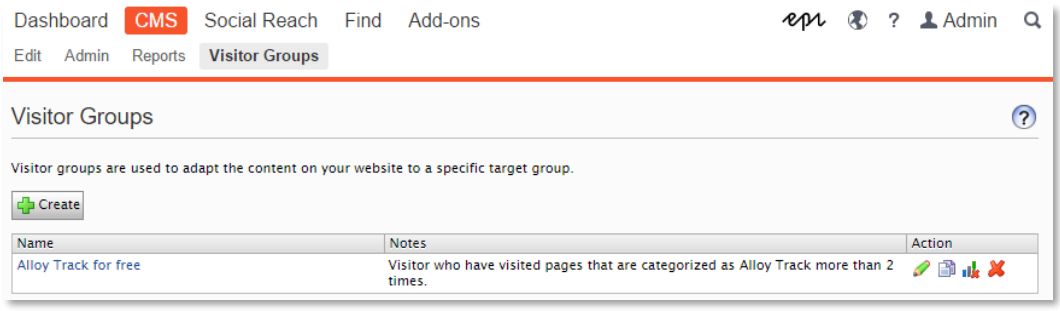
```
Install-Package -ProjectName AlloyAdvanced
    EPiServer.VisitorGroupsCriteriaPack
```

The default 11 visitor groups criteria are shown in the following screenshots:

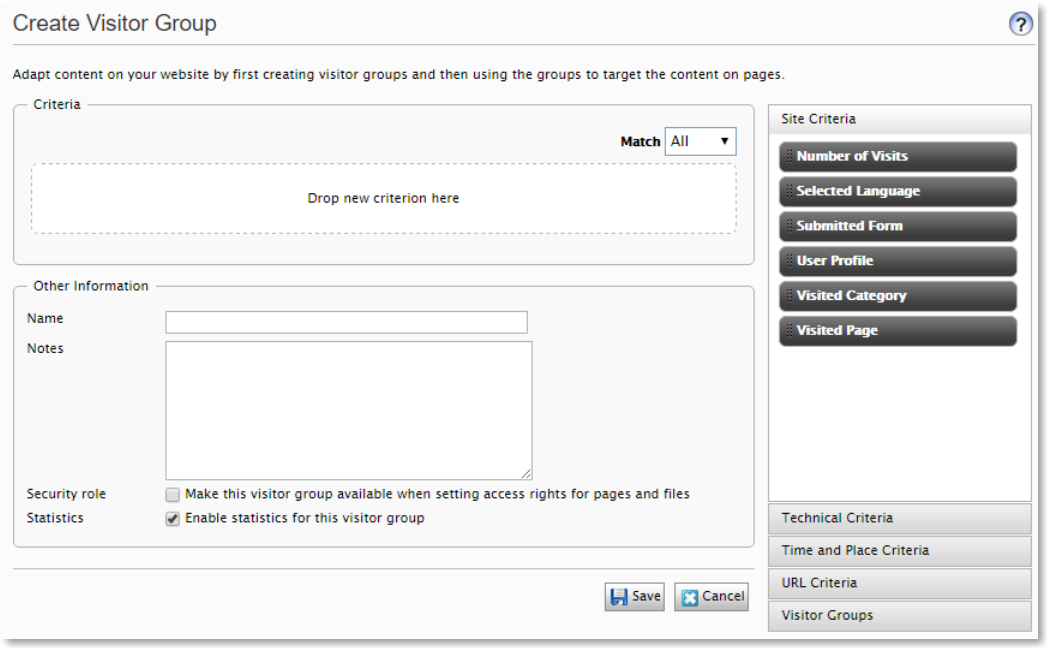


After installing the *Visitor Groups Criteria Pack* add-on, you will have 11 more.

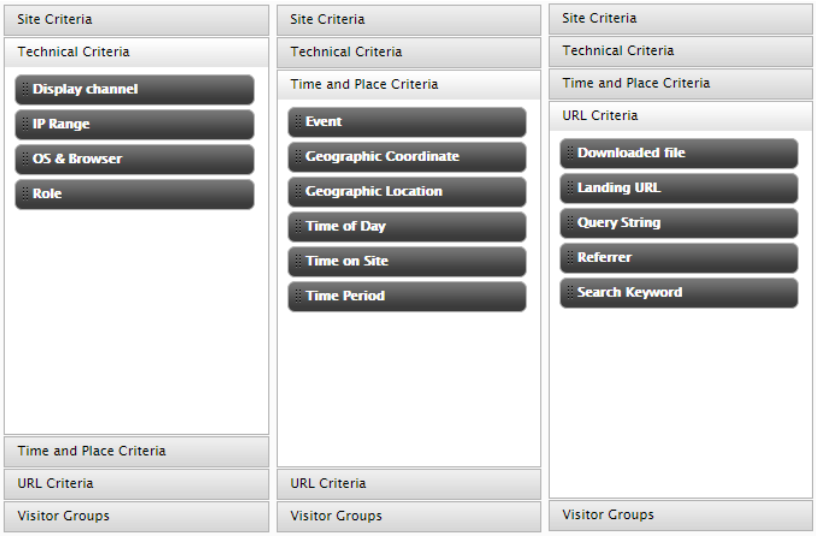
1. Navigate to **CMS** | **Visitor Groups**.



2. Click **Create**. Note the two extra **Site Criteria**, named **Selected Language** and **Submitted Form**, as shown in the following screenshot:



The Visitor Groups Criteria Pack adds four **Technical Criteria**, three **Time and Place Criteria** named **Event**, **Time on Site**, and **Time Period**, and two **URL Criteria** named **Downloaded file** and **Query String**, as shown in the following screenshots:



Creating a visitor groups criterion

1. Open the solution with the **AlloyAdvanced** project.
2. In **~\Business**, add a folder named **VisitorGroups**.
3. In **~\Business\VisitorGroups**, add a class named **CookieCriterionModel**.
4. Modify its content, as shown in the following code:

```
using EPiServer.Personalization.VisitorGroups;
using System.ComponentModel.DataAnnotations;

namespace AlloyAdvanced.Business.VisitorGroups
{
    public class CookieCriterionModel : CriterionModelBase
```

```
    {
        [Required]
        public string Name { get; set; }

        [Required]
        public string Value { get; set; }

        public override ICriterionModel Copy()
        {
            return base.ShallowCopy();
        }
    }
}
```

5. In **~\Business\VisitorGroups**, add a class named **CookieCriterion**.
6. Modify its content, as shown in the following code:

```
using EPiServer.Personalization.VisitorGroups;
using System.Security.Principal;
using System.Web;

namespace AlloyAdvanced.Business.VisitorGroups
{
    [VisitorGroupCriterion(Category = "Site Criteria", Description =
        "Checks for the existence of a named cookie with a set value.",
        DisplayName = "Cookie")]
    public class CookieCriterion : CriterionBase<CookieCriterionModel>
    {
        public override bool IsMatch(IPrincipal principal,
            HttpContextBase httpContext)
        {
            HttpCookie cookie = httpContext.Request.Cookies.Get(Model.Name);
            if (cookie == null) return false;
            return (cookie.Value == Model.Value);
        }
    }
}
```

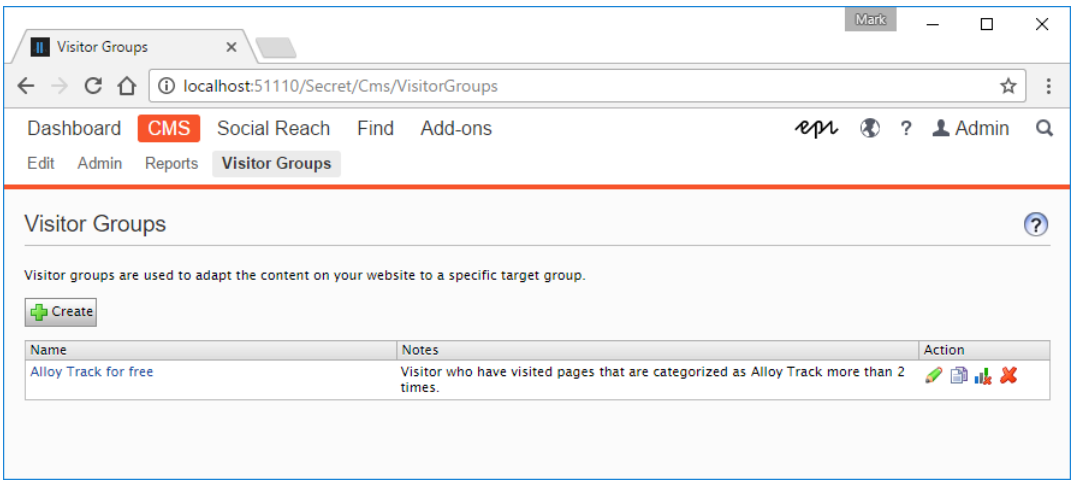
In this implementation, the cookie value must be an exact match i.e. it is case-senstive. You could change the comparison to make it case-insensitive.

7. In **~\Controllers**, open **SearchPageController.cs**.
8. At the bottom of the **Index** method, before the call to **return View()**, add statements as shown in the following code:

```
// a hacky way to quickly set a temporary,
// in-browser-process cookie, by looking for a special
// search query like: epi=>red
if (!string.IsNullOrWhiteSpace(q) && q.Contains("=>"))
{
    string[] parts = q.Split(new string[] { "=>" },
        System.StringSplitOptions.RemoveEmptyEntries);

    if (parts.Length == 2)
    {
        Response.Cookies.Add(new HttpCookie(parts[0], parts[1]));
    }
}

return View(model);
```

Define a visitor group and personalized content

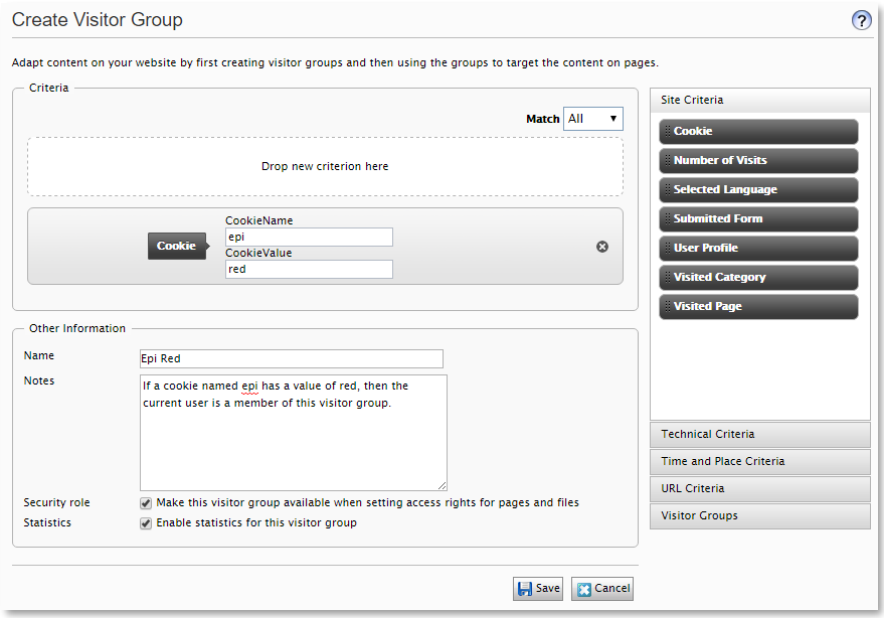
1. Start the site, and log in as **Admin**.
2. Navigate to **CMS** | **Visitor Groups**, as shown in the following screenshot:



3. Click **Create**.
4. Create a visitor group with the following details:
   - Drag and drop the **Cookie** criterion from the **Site Criteria** group.
   - Name: **epi**
   - Value: **red**

Remember that the name and value are case sensitive!

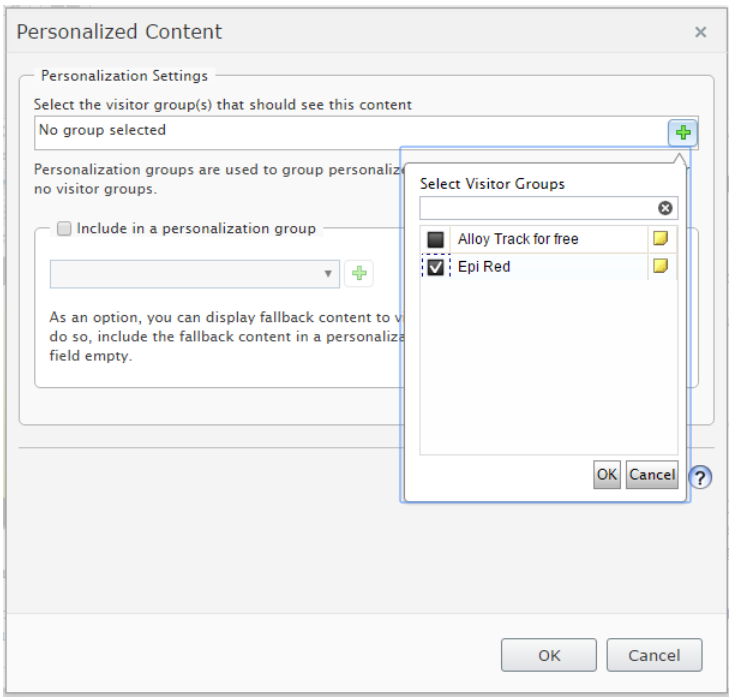
   - Name: **Epi Red**
   - Notes: **If a cookie name epi has a value of red, then the current user is a member of this visitor group.**
   - Security role: checked



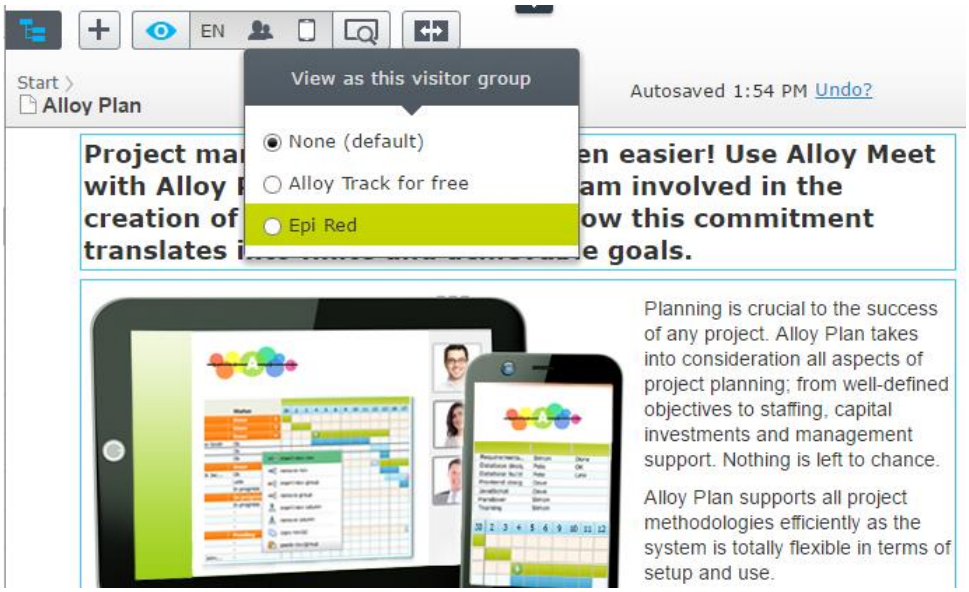
5. Click **Save**.
6. Navigate to **CMS** | **Edit**, and edit the **Alloy Plan** page.

7. In the **Main body** property, add a paragraph saying, **Episerver is RED!**, and select it, as shown in the following screenshot:

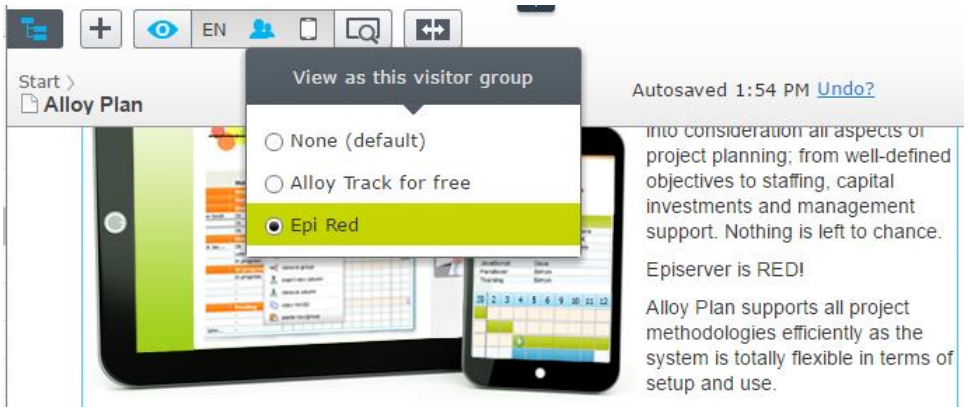
8. In the rich text toolbar, click the green with three white people button to mark the selected test as **Personalized Content**.
9. Select the **Epi Red** visitor group, as shown in the following screenshot:



10. Click **OK**, and then click **OK**.
11. **Publish** the changes to the Alloy Meet page.
12. Note that for an ordinary, default visitor, they cannot see the special message, as shown in the following screenshot:



13. In the **Toggle view settings** toolbar, click the **Select to view as a visitor group** button, and choose **Epi Red**, as shown in the following screenshot, and note the special message is now visible:

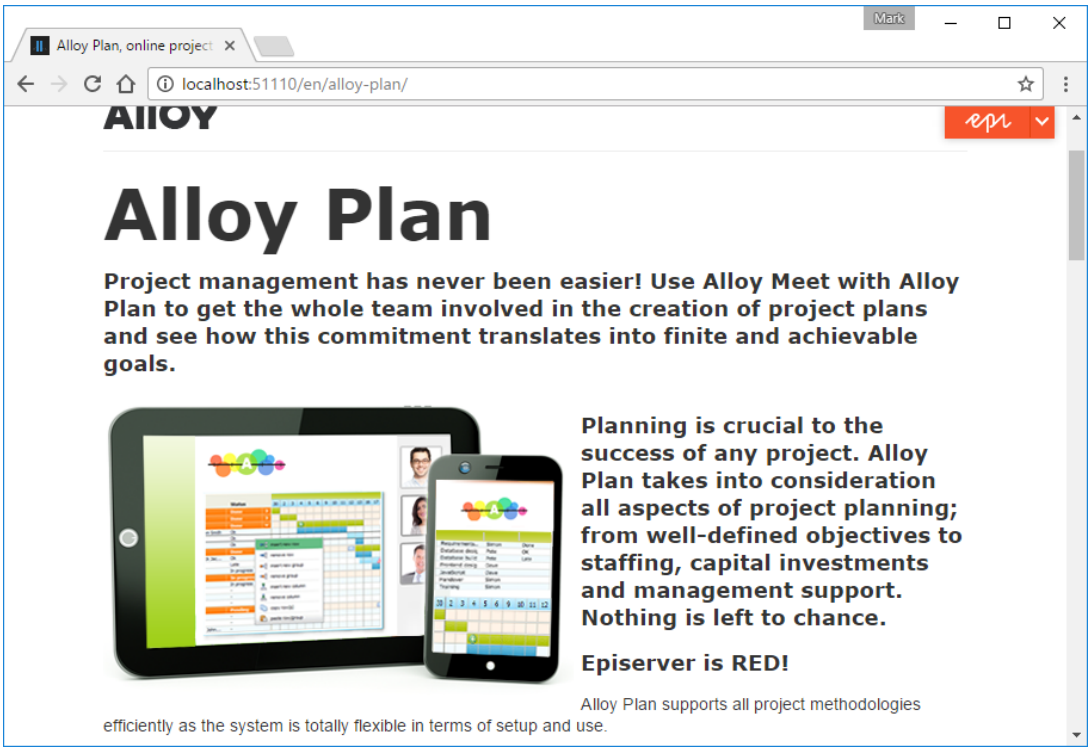


Test the visitor group as a visitor

1. View the site as a visitor.
2. Navigate to the **Alloy Plan** page, and it will NOT show the personalized content, because the cookie doesn't exist (yet).
3. In the Search box, enter **epi=>red**.

Remember that the name and value are case sensitive!

4. Navigate back to the **Alloy Plan** page, and it will show the personalized content, as shown in the following screenshot:



5. Close the browser.

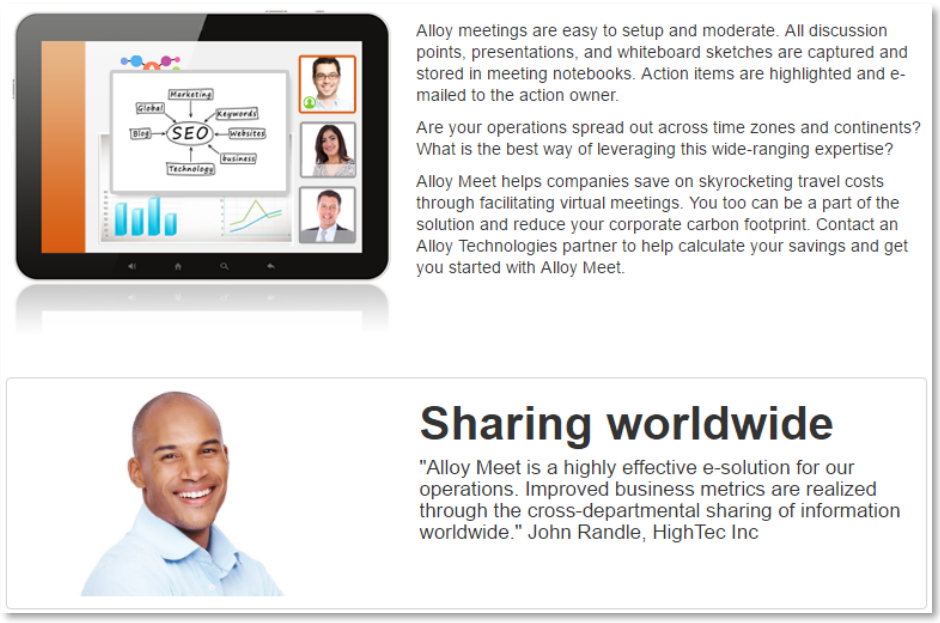
Exercise E4 – Adding fields to Episerver Search

In this exercise, you will add a field to the Episerver Search index.

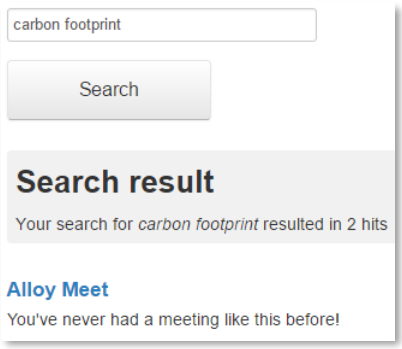
Prerequisites: complete Exercise 0.1.

Reviewing default indexing behaviour

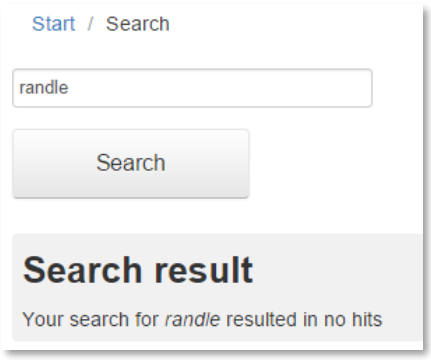
1. Open the solution with the **AlloyAdvanced** project.
2. Start the site.
3. Navigate to the **Alloy Meet** page, and note the page contains the words “carbon footprint” and “John Randle”, as shown in the following screenshot:



4. Click in the **Search** box, and enter **carbon footprint**, and note the **Alloy Meet** page is found, as shown in the following screenshot:



5. Click in the **Search** box, and enter **randle**, and note the **Alloy Meet** page is NOT found, as shown in the following screenshot:



Content in properties of type **string** and **XhtmlString** are indexed, but a page with a content area containing a block are not indexed to refer to that page, even though a content editor performing a Global search for **randle** would find the block itself.

Creating extensions and an initialization module to listen for indexing event

The block containing randle is an instance of a **TeaserBlock**, which has a property named **Text**. You will now customize Episerver Search so that the Text property of a TeaserBlock in a content area is indexed as part of the page.

1. In **~/Helpers**, add a new class named **DocumentExtensions**.
2. Modify the class as shown in the following code:

```
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using Lucene.Net.Documents;
using System;

namespace AlloyAdvanced.Helpers
{
    public static class DocumentExtensions
    {
        public static T GetContent<T>(this Document doc) where T : IContent
        {
            const string fieldName = "EPISERVER_SEARCH_ID";

            string fieldValue = doc.Get(fieldName);

            if (string.IsNullOrWhiteSpace(fieldValue))
            {
                throw new NotSupportedException(
                    $"Specified document did not have a '{fieldName}' value.");
            }

            string[] fieldFragments = fieldValue.Split('|');

            Guid contentGuid;

            if (!Guid.TryParse(fieldFragments[0], out contentGuid))
            {
                throw new NotSupportedException(
                    "Expected first part of ID field to be a valid GUID.");
            }

            return ServiceLocator.Current
                .GetInstance<IContentLoader>()
                .Get<T>(contentGuid);
```

```
        }
    }
}
```

3. In **~/Business/Initialization**, add a new Episerver **Initialization Module** named **SearchBlocksInitializationModule**.
4. Modify the class as shown in the following code:

```
using AlloyAdvanced.Helpers;
using AlloyAdvanced.Models.Blocks;
using AlloyAdvanced.Models.Pages;
using EPiServer;
using EPiServer.Core;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.Search.IndexingService;
using EPiServer.ServiceLocation;
using Lucene.Net.Documents;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AlloyAdvanced.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class SearchBlocksInitializationModule : IInitializableModule
    {
        public void Initialize(InitializationEngine context)
        {
            IndexingService.DocumentAdding += IndexingService_DocumentAdding;
        }

        private void IndexingService_DocumentAdding(object sender, EventArgs e)
        {
            var args = e as AddUpdateEventArgs;

            if (args == null) return;

            Document doc = args.Document;

            PageData page = doc.GetContent<IContent>() as PageData;

            if (page != null && page is ProductPage)
            {
                ProductPage product = page as ProductPage;

                if (product.MainContentArea == null) return;

                var loader = ServiceLocator.Current
                    .GetInstance<IContentLoader>();

                IEnumerable<IContent> items = product.MainContentArea.FilteredItems
                    .Select(item => loader.Get<IContent>(item.ContentLink));

                foreach (IContent item in items)
                {
                    if (item is TeaserBlock)
                    {
                        var teaser = item as TeaserBlock;
                        doc.Add(new Field("TEASERBLOCK_FIELD", teaser.Text,
                            Field.Store.NO, Field.Index.ANALYZED));
```

```
                    }
                }
            }
        }

        public void Uninitialize(InitializationEngine context)
        {
            IndexingService.DocumentAdding -= IndexingService_DocumentAdding;
        }
    }
}
```

5. In **~/Business**, add a new class named **CustomFieldQuery**.
6. Modify the class as shown in the following code:

```
using EPiServer.Search.Queries;
using EPiServer.Search.Queries.Lucene;

namespace AlloyAdvanced.Business
{
    public class CustomFieldQuery : IQueryExpression
    {
        public CustomFieldQuery(string queryExpression, string fieldName)
        {
            Expression = queryExpression;
            Field = fieldName;
        }

        public string GetQueryExpression()
        {
            return string.Format("{0}:({1}{2})",
                Field,
                LuceneHelpers.EscapeParenthesis(Expression),
                string.Empty);
        }

        public string Field { get; set; }

        public string Expression { get; set; }
    }
}
```

Modifying the search controller

You will now modify the existing search controller. Instead of performing a simple text query, you will search in the default field and in a custom field for teaser blocks.

1. Open **~/Controllers/SearchPageController.cs**.
2. At the top of the class, import some namespaces, as shown in the following code:

```
using EPiServer.Search.Queries.Lucene;
using EPiServer.Security;
using EPiServer.ServiceLocation;
```

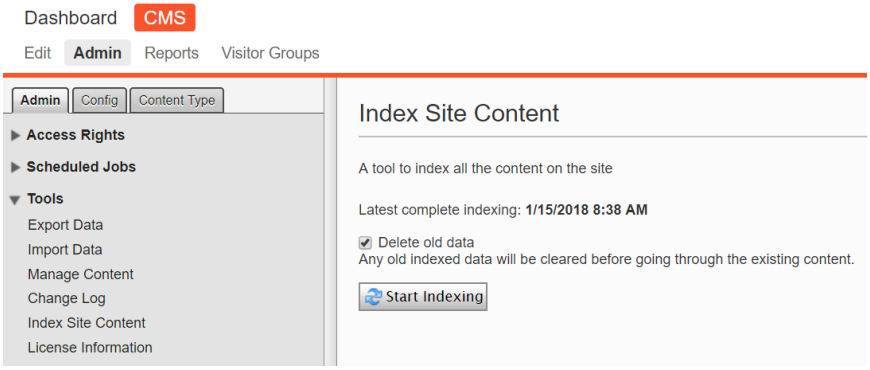
3. Modify the statements inside the **Search** method, as shown in the following code:

```
private IEnumerable<SearchContentModel.SearchHit> Search(
    string searchText, IEnumerable<ContentReference> searchRoots,
    HttpContextBase context, string languageBranch)
{
    var query = new GroupQuery(LuceneOperator.AND);

    query.QueryExpressions.Add(new ContentQuery<PageData>());
```

```
        var keywordsQuery = new GroupQuery(LuceneOperator.OR);

        keywordsQuery.QueryExpressions.Add(new FieldQuery(searchText));

        keywordsQuery.QueryExpressions.Add(
            new CustomFieldQuery(searchText, "TEASERBLOCK_FIELD"));

        query.QueryExpressions.Add(keywordsQuery);

        var accessQuery = new AccessControlListQuery();
        accessQuery.AddAclForUser(PrincipalInfo.Current, HttpContext);

        query.QueryExpressions.Add(accessQuery);

        var searchHandler = ServiceLocator.Current.GetInstance<SearchHandler>();

        var results = searchHandler.GetSearchResults(query, 1, 40);

        return results.IndexResponseItems.SelectMany(CreateHitModel);
    }
```

Refresh the index

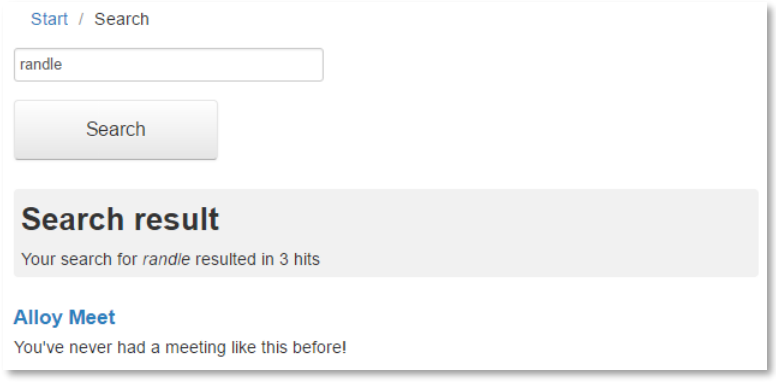
1. Start the site, and log in as **Admin**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Index Site Content**.
3. Click **Start Indexing**, as shown in the following screenshot:



4. Close the browser.

Test the custom teaser field indexing

1. View the site as a visitor.
2. Search for **randle**. You will find at least one hit, as shown in the following screenshot:



Good practice

For more information about the best process to deploy a site that uses Episerver Search, read the following blog article by Henrik Fransas:

http://world.episerver.com/blogs/Henrik-Fransas/Dates/2015/8/setup-episerver-search-in-a-future-safe-and-stable-way/

You have now completed some exercises about rendering, personalization, and advanced searching.

Module F – Extending with Plug-ins and Add-ons

Goal

The overall goal of the exercises in this module is to learn how to implement various types of plug-ins and explore add-ons. You will:

- Explore some existing add-ons and plug-ins.
- Create some scheduled job plug-ins.
- Create an admin tool plug-in.
- Create a report plug-in.
- Create a task plug-in.

Exercise F1 – Exploring existing add-ons and plug-ins

In this exercise, you will install some add-ons that show various ways to extend Episerver with add-ons and plug-ins. The add-ons are:

- Episerver Social Reach: http://webhelp.episerver.com/latest/addons/socialreach.htm
- PowerSlice: http://world.episerver.com/add-ons/powerslice/
- Episerver Self-Optimizing Block: http://webhelp.episerver.com/latest/addons/selfoptimizingblock.htm
- Episerver A/B Testing: http://webhelp.episerver.com/latest/cms-edit/ab-testing.htm

Prerequisites: complete Exercise 0.1.

NuGet package names are not case-sensitive.

Installing the add-ons

After installing some of these add-ons, you will need you to update other project package dependencies and the database schema. As good practice, after installing an add-on, always update packages for your project and then update the database schema.

Exploring Episerver Social Reach

Episerver Social Reach is an example of an add-on that adds itself to the **Global** menu at the product level.

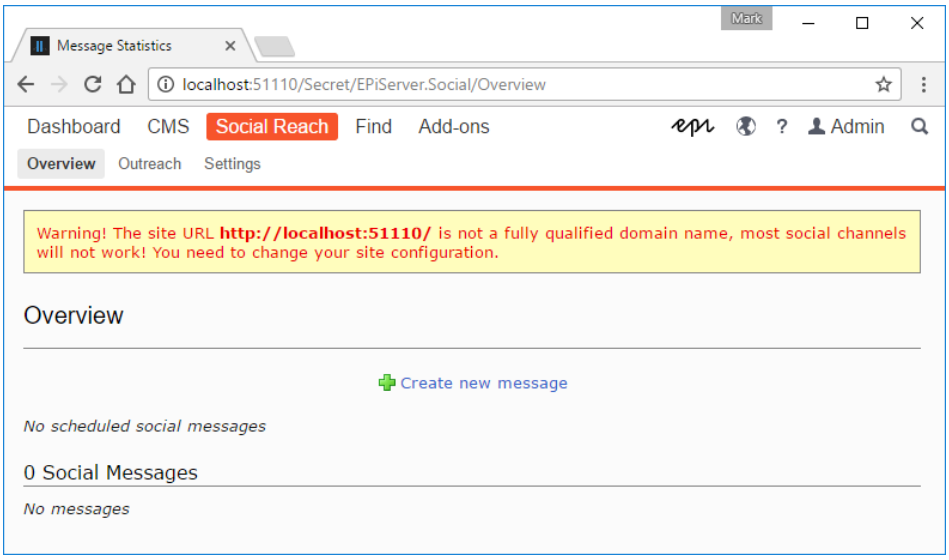
1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**, and set the following
    a. Package source: **All**
    b. Default project: **AlloyAdvanced**.
3. Enter the following commands:

```
Install-Package -ProjectName AlloyAdvanced EPiServer.Social
```

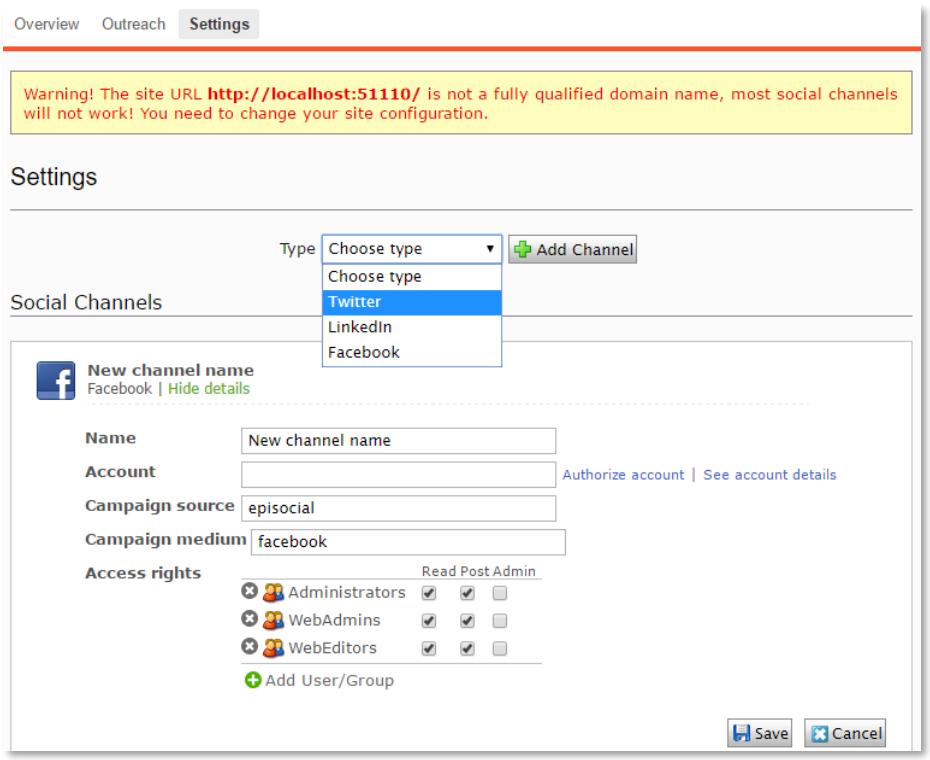
EPiServer.Social package is **Episerver Social Reach**. It is NOT **Episerver Social**!

4. Start the site, and log in as **Admin**.

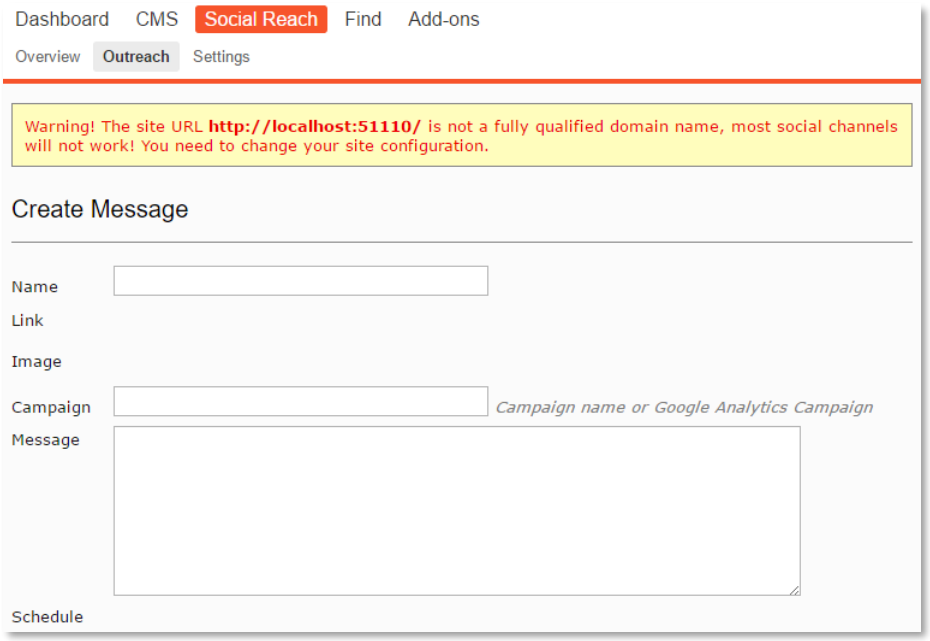
5. Pull down the **Global** menu and note the extra menu option, **Social Reach**, as shown in the following screenshot:



6. Navigate to **Social Reach** | **Settings**, and note that you can create social channels to sites including Twitter, LinkedIn, and Facebook, as shown in the following screenshot:



7. Navigate to **Social Reach** | **Outreach**, and note that after setting up some social channels, you can create a message to push out to all your channels with a link back to one of the pages on your Episerver site, as shown in the following screenshot:



Exploring PowerSlice

Prerequisites: complete the first three tasks in **Exercise G1 – Implementing Episerver Find**: *Registering a Find account*, *Creating a developer index*, and *Installing and configuring Find for a CMS project*.

PowerSlice has a dependency on Episerver Find. Do not install it unless you have configured a Find index.

PowerSlice is an example of an add-on that adds itself as a gadget in the **Assets** pane, and it requires a developer to write some classes before it works correctly.

1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**, and set the following
    a. Package source: **All**
    b. Default project: **AlloyAdvanced**.
3. Enter the following commands:

```
Install-Package -ProjectName AlloyAdvanced PowerSlice
```

4. In **~/Business**, create a folder named **PowerSlices**.
5. In **~/Business/PowerSlices**, add a class named **EverythingSlice**. Modify its contents as shown in the following code:

```
using EPiServer.Core;
using EPiServer.ServiceLocation;
using EPiServer.Shell.ContentQuery;
using PowerSlice;

namespace AlloyAdvanced.Business.PowerSlices
{
    [ServiceConfiguration(typeof(IContentQuery)),
        ServiceConfiguration(typeof(IContentSlice))]
    public class EverythingSlice : ContentSliceBase<IContent>
    {
```

```
        public override string Name
        {
            get { return "Everything"; }
        }
    }
}
```

6. In **~/Business/PowerSlices**, add a class named **ProductPageSlice**. Modify its contents as shown in the following code:

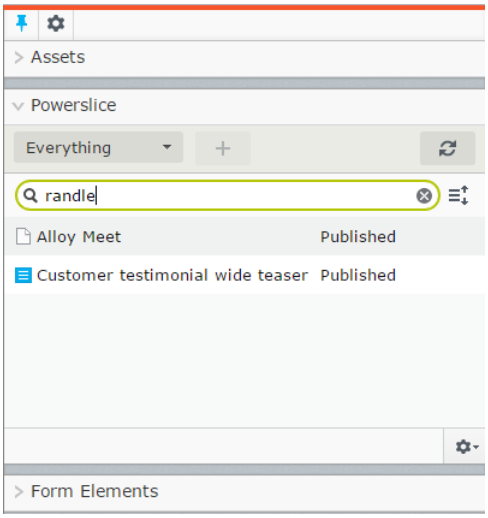
```
using AlloyAdvanced.Models.Pages;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using EPiServer.Shell.ContentQuery;
using PowerSlice;
using System.Collections.Generic;

namespace AlloyAdvanced.Business.PowerSlices
{
    [ServiceConfiguration(typeof(IContentQuery)),
        ServiceConfiguration(typeof(IContentSlice))]
    public class ProductPageSlice : ContentSliceBase<ProductPage>
    {
        public override string Name
        {
            get { return "Product Pages"; }
        }

        public override IEnumerable<CreateOption> CreateOptions
        {
            get
            {
                var contentType = ContentTypeRepository.Load(
                    typeof(ProductPage));

                yield return new CreateOption(
                    label: contentType.LocalizedName,
                    parent: ContentReference.StartPage,
                    contentTypeId: contentType.ID);
            }
        }
    }
}
```

7. In **~/Business/PowerSlices**, add a class named **MySitePagesSlice**. Modify its contents as shown in the following code:

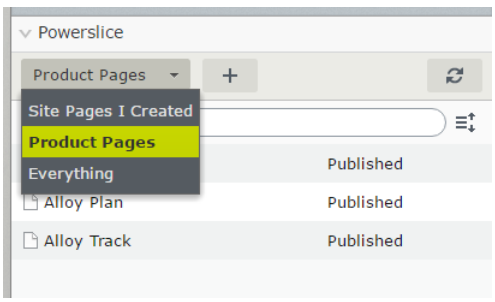
```
using AlloyAdvanced.Models.Pages;
using EPiServer.Cms.Shell.UI.Rest.ContentQuery;
using EPiServer.Core;
using EPiServer.Find;
using EPiServer.ServiceLocation;
using EPiServer.Shell.ContentQuery;
using PowerSlice;
using System.Web;

namespace AlloyAdvanced.Business.PowerSlices
{
    [ServiceConfiguration(typeof(IContentQuery)),
        ServiceConfiguration(typeof(IContentSlice))]
    public class MySitePagesSlice : ContentSliceBase<SitePageData>
    {
        public override string Name
```

```
        {
            get { return "Site Pages I Created"; }
        }

        protected override ITypeSearch<SitePageData> Filter(
            ITypeSearch<SitePageData> searchRequest,
            ContentQueryParameters parameters)
        {
            var userName = HttpContext.Current.User.Identity.Name;

            return searchRequest.Filter(x => x.MatchTypeHierarchy(
                typeof(IChangeTrackable)) &
                ((IChangeTrackable)x).CreatedBy.Match(userName));
        }
    }
}
```

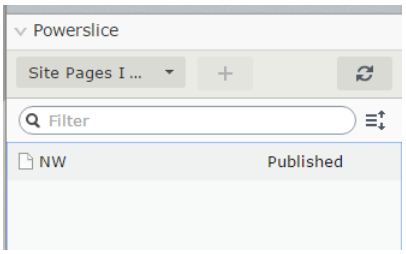
8. Start the site, and log in as **Admin**.
9. Navigate to **CMS** | **Edit**.
10. Show the **Assets** pane, and expand the **Powerslice** gadget.
11. In the drop-down, choose **Everything**, and in the **Search** box, enter **randle**, as shown in the following screenshot, and you will see matches for (1) the teaser block that contains the text **randle**, and (2) the Alloy Meet page that has that block in its main content area (if you completed the exercise to applied the [IndexInContentAreas] attribute to the TeaserBlock class):



12. In the drop-down, choose **Product Pages**, and you will see all product pages listed, and you could click the + button to add a new product page underneath the Start page, as shown in the following screenshot:



13. In the drop-down, choose **Site Pages I Created**, and note that only pages that you created are visible, as shown in the following screenshot:



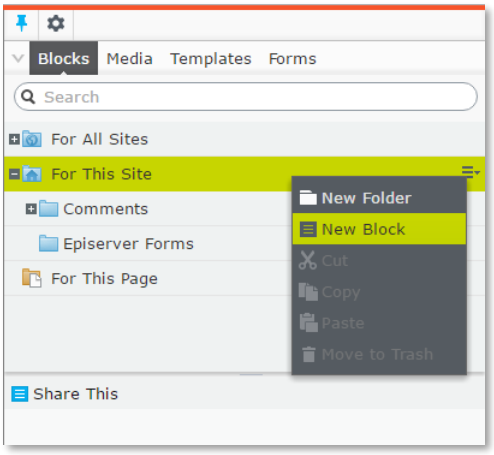
Exploring Episerver Self-Optimizing Block

Episerver Self-Optimizing Block is an example of an add-on that adds itself as a block type.

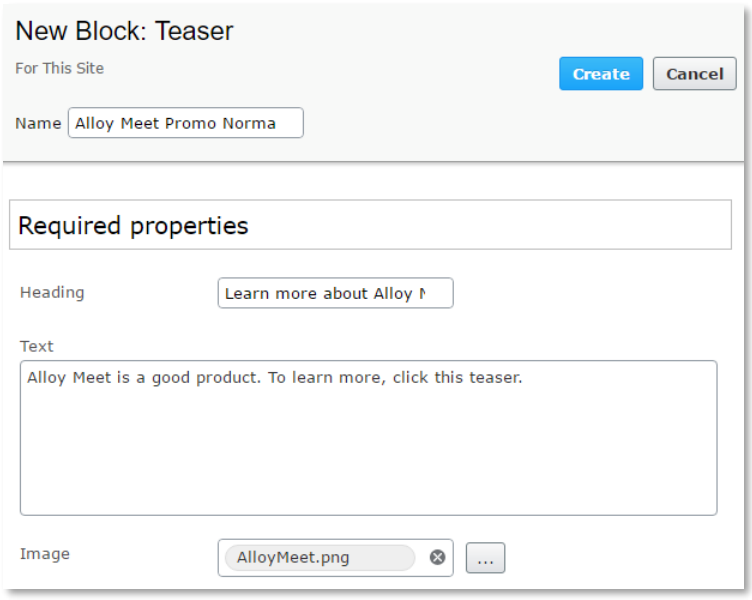
1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**, and set the following
    a. Package source: **All**
    b. Default project: **AlloyAdvanced**.
3. Enter the following commands:

    ```
    Install-Package -ProjectName AlloyAdvanced EPiServer.CMS.AddOns.Blocks
    ```

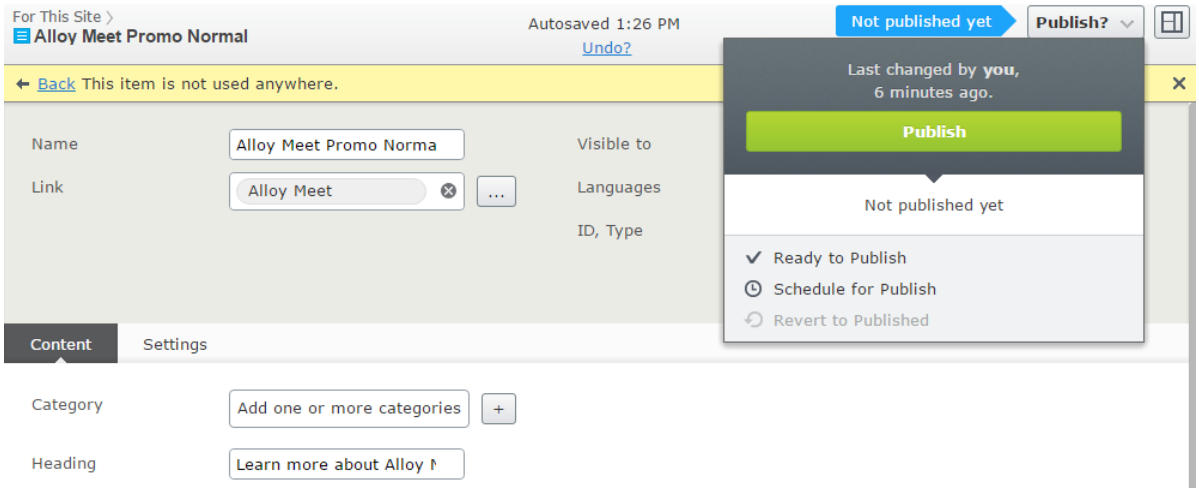
4. Start the website and log in as **Admin**.
5. Navigate to **CMS** | **Edit**, and edit the **About us** page.
6. On the **Assets** pane, click the **Blocks** tab, click **For This Site**, and on its context menu, click **New Block**, as shown in the following screenshot:



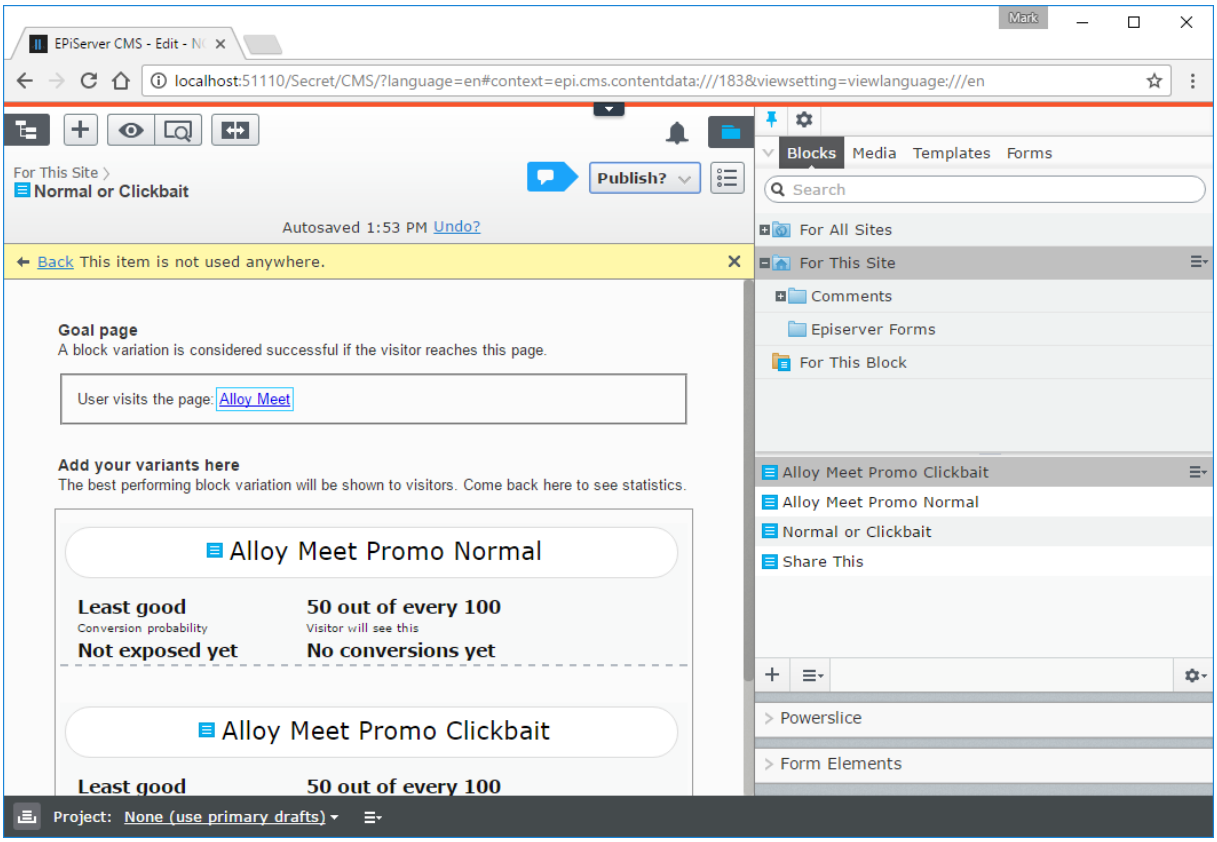
7. Choose **Teaser** as the block type, and enter details, as shown in the following screenshot:
    - Name: **Alloy Meet Promo Normal**
    - Heading: **Learn more about Alloy Meet**
    - Text: **Alloy Meet is a good product. To learn more, click this teaser.**
    - Image: **AlloyMeet.png**



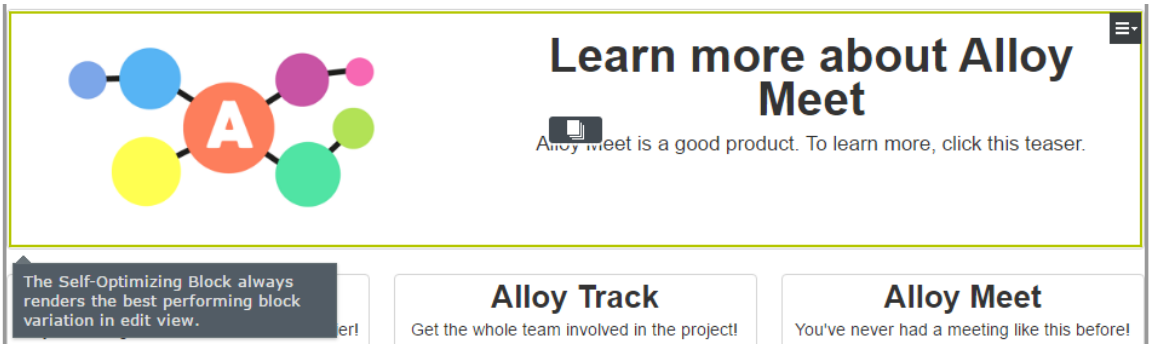
8. Switch to **All Properties** view.
9. Set the **Link** to **Alloy Meet** page, and then **Publish** the block, as shown in the following screenshot:



10. Create a second teaser block, and enter details, as shown in the following bullets:
    - Name: **Alloy Meet Promo Clickbait**
    - Heading: **This woman bought Alloy Meet. You won't believe what happened next!**
    - Text: **You'll be outraged at how easy it was to get you to click on this teaser**
    - Image: **ToddSlayton.png**
11. Switch to **All Properties** view, set the **Link** to the **Alloy Meet** page, and then **Publish** the block.
12. Create a new block, choose **Self Optimizing Block** as the block type, and enter a name of **Normal or Clickbait.**
13. In the Edit view, set the **Goal page** to the **Alloy Meet** page, and drag and drop the two promotional teasers into the **Add your variants here section**, as shown in the following screenshot:



14. **Publish** the block.
15. Edit the Start page, and drag and drop the **Normal or Clickbait** block into the main content area, as shown in the following screenshot:



Exploring Episerver A/B Testing

Episerver A/B Testing is an example of an add-on that adds itself to the Episerver menu system.

1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**, and set the following
    a. Package source: **All**
    b. Default project: **AlloyAdvanced**.
3. Enter the following commands:

```
Install-Package -ProjectName AlloyAdvanced EPiServer.Marketing.Testing
Update-Package -ProjectName AlloyAdvanced
Update-EPiDatabase
```

4. Start the website and log in as **Admin**.
5. Navigate to **CMS** | **Edit**, and edit the **Alloy Meet** page.
6.

Finding Episerver add-ons

The best way to find Episever add-ons is to use the Episerver Nuget Feed Explorer. For example, if you want a list of all add-ons deployed as NuGet packages that were created by Episerver, use the following link:

https://www.david-tec.com/episerver-nuget-feed-explorer/?PublishedBy=&PublishedDate=&Author=EPiServer

Exercise F2 – Creating scheduled job plug-ins

In this exercise, you will create scheduled job plug-ins that (1) keeps the current site alive, and (2) performs automatic regular fixes on content.

Prerequisites: complete Exercise 0.1.

Creating a scheduled job to ping the current site

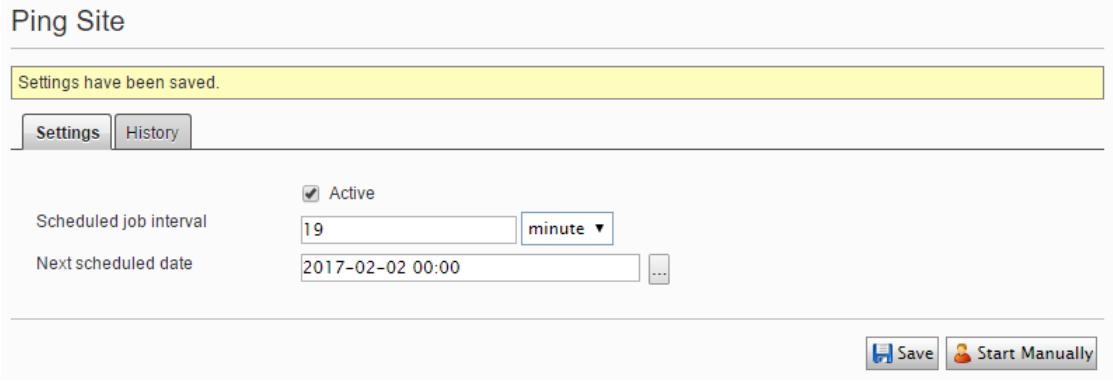
First, you will create a scheduled job that makes requests to the current site to keep itself "alive". Think how you might achieve this goal, and try to implement it yourself. If you get stuck, follow these steps:

1. Open the solution with the **AlloyAdvanced** project.
2. In **~\Business**, add a new folder named **ScheduledJobs**.
3. In **~\Business\ScheduledJobs**, add an **Episerver** | **Scheduled Job** named **PingSiteScheduledJob**.
4. Modify the statements, as shown in the following code:

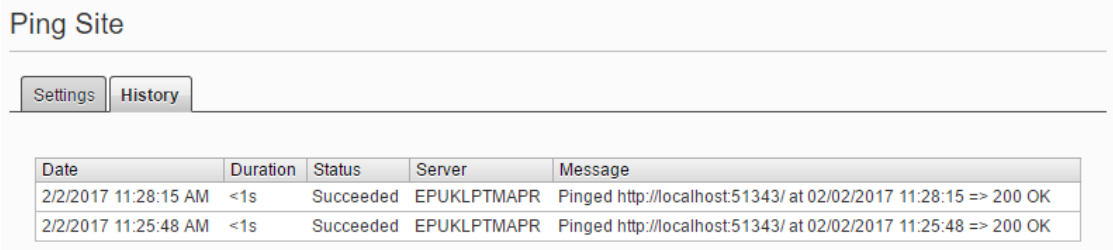
```
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using EPiServer.Web;
using System;
using System.Net.Http;

namespace AlloyAdvanced.Business.ScheduledJobs
{
    [ScheduledPlugIn(DisplayName = "Ping Site")]
    public class PingSiteScheduledJob : ScheduledJobBase
    {
        public PingSiteScheduledJob()
        {
            IsStoppable = false; // hide the Stop button
        }

        public override string Execute()
        {
            var client = new HttpClient();
            client.BaseAddress = SiteDefinition.Current.SiteUrl;
            HttpResponseMessage response = client.GetAsync("").Result;
            return $"Pinged {client.BaseAddress} at {DateTime.Now} =>
{(int)response.StatusCode} {response.ReasonPhrase}";
        }
    }
}
```

5. Start the site, and log in as **Admin**.
6. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Ping Site**.

7. Set the job to execute every 19 minutes (because after 20 minutes of inactivity, IIS shuts down the AppDomain hosting the site), and click **Save**, as shown in the following screenshot:



8. Click **Start Manually**.
9. Click **History**, and note the job has successfully pinged itself, as shown in the following screenshot:



Creating a scheduled job to perform automatic fixes of invalid content

Create a scheduled job that finds pages that contain "iWatch" in the **Name**, **MetaTitle**, **MetaDescription**, or **MetaKeywords**. If found, change to "Apple Watch" and log that the change was made. Think how you might achieve this goal, and try to implement it yourself. If you get stuck, follow these steps:

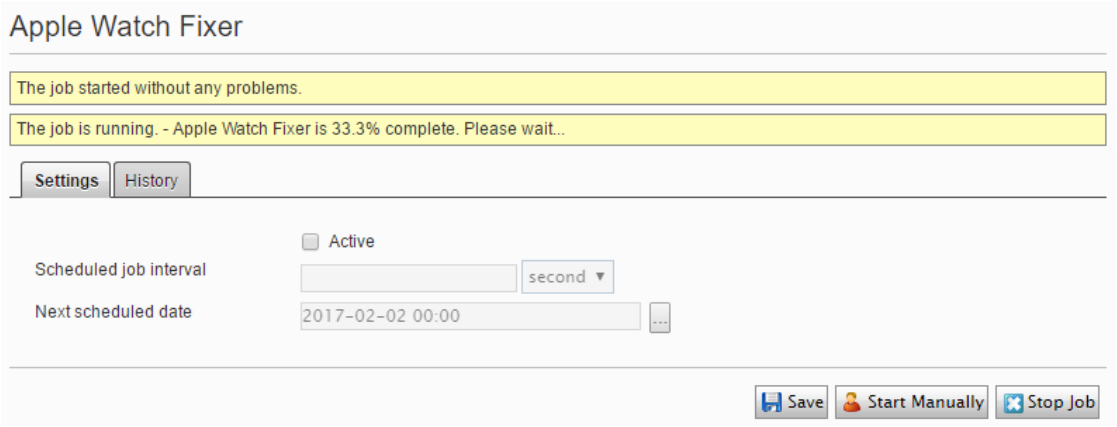
1. In **~\Business\ScheduledJobs**, add an **Episerver** | **Scheduled Job** named **AppleWatchFixerScheduledJob**.
2. Modify the statements, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using EPiServer;
using EPiServer.Core;
using EPiServer.Filters;
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using EPiServer.ServiceLocation;

namespace AlloyAdvanced.Business.ScheduledJobs
{
    [ScheduledPlugIn(DisplayName = "Apple Watch Fixer")]
    public class AppleWatchFixerScheduledJob : ScheduledJobBase
    {
        private const string jobName = "Apple Watch Fixer";
        private bool _stopSignaled;

        public AppleWatchFixerScheduledJob()
        {
            IsStoppable = true;
        }

        public override void Stop()
        {
            _stopSignaled = true;
```

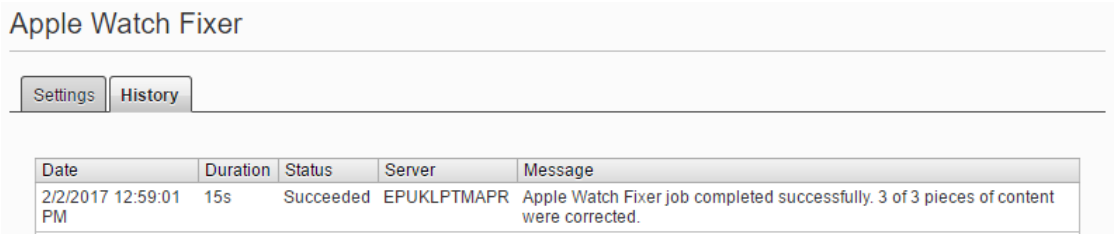
```
        }

        public override string Execute()
        {
            OnStatusChanged($"Starting execution of {jobName} job...");

            var finder = ServiceLocator.Current
                .GetInstance<IPageCriteriaQueryService>();

            var criteria = new PropertyCriteriaCollection();
            criteria.Add(new PropertyCriteria
            {
                Type = PropertyDataType.LongString,
                Name = "PageName", // you cannot use Name
                Condition = CompareCondition.Contained,
                Value = "iWatch"
            });
            criteria.Add(new PropertyCriteria
            {
                Type = PropertyDataType.LongString,
                Name = "MetaTitle",
                Condition = CompareCondition.Contained,
                Value = "iWatch"
            });
            criteria.Add(new PropertyCriteria
            {
                Type = PropertyDataType.LongString,
                Name = "MetaDescription",
                Condition = CompareCondition.Contained,
                Value = "iWatch"
            });

            PageDataCollection matches = finder.FindPagesWithCriteria(
                ContentReference.RootPage, criteria);

            int total = matches.Count;
            int count = 0;
            var repo = ServiceLocator.Current.GetInstance<IContentRepository>();

            foreach (PageData page in matches)
            {
                if (_stopSignaled)
                {
                    return $"{jobName} job was stopped. {count} of {total} pieces of
content were corrected.";
                }

                SitePageData sitepage = page.CreateWritableClone() as SitePageData;
                if (sitepage != null)
                {
                    sitepage.Name = sitepage.Name.Replace("iWatch", "Apple Watch");
                    sitepage.MetaDescription = sitepage.MetaDescription.Replace(
                        "iWatch", "Apple Watch");
                    sitepage.MetaTitle = sitepage.MetaTitle.Replace(
                        "iWatch", "Apple Watch");

                    repo.Save(content: sitepage,
                        action: EPiServer.DataAccess.SaveAction.Publish,
                        access: EPiServer.Security.AccessLevel.NoAccess);

                    count++;
                }
```

```
                OnStatusChanged($"{jobName} is {count/(double)total*100:0.0}%
complete. Please wait...");

                // pause for five seconds to simulate work, and
                // to allow us to test the Stop functionality ;)
                System.Threading.Thread.Sleep(5000);
            }

            return $"{jobName} job completed successfully. {count} of {total} pieces
of content were corrected.";
        }
    }
}
```

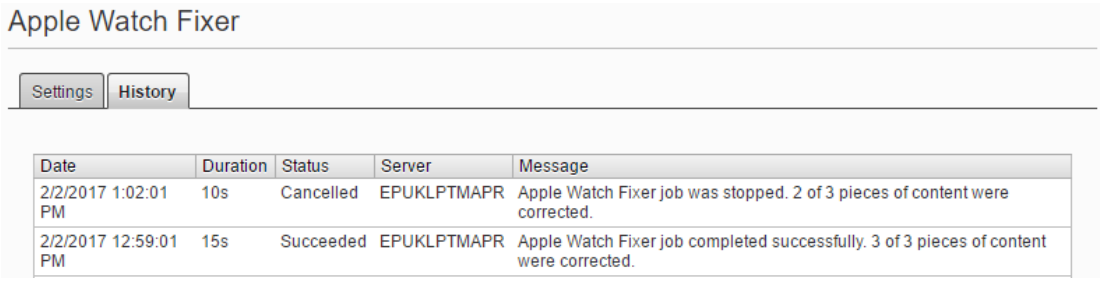
3. Start the site, and log in as **Admin**.
4. Edit the **Start** page, add **iWatch** into the **Name**, and publish the change.
5. Edit the **Alloy Plan** page, add **iWatch** into the **Title** (MetaTitle), and publish the change.
6. Edit the **About us** page, add **iWatch** into the **Page description** (MetaDescription), and publish the change.
7. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Apple Watch Fixer**.
8. Click **Start Manually**, and wait for it to complete successfully, as shown in the following screenshot:



9. Click **History**, and note the job has successfully fixed the three mistakes, as shown in the following screenshot:



10. Try making some more mistakes, by adding iWatch to several pages, and then starting and then stopping the job, as shown in the following screenshot:



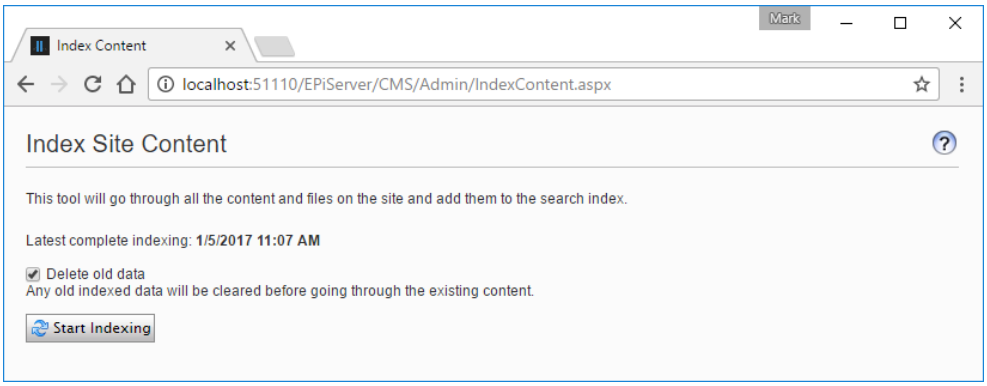
Exercise F3 – Creating an admin tool plug-in

In this exercise, you will create an admin tool that replicates the Episerver Search Index Site Content UI.

Prerequisites: complete Exercise 0.1.

Administrators can manually enter the following URL to a Web Forms page in their browser address bar to access the ability to reindex the Episerver Search content index, as shown in the following screenshot:

`http://localhost:[port]/EPiServer/CMS/Admin/IndexContent.aspx`



You will now create an admin tool plug-in built with MVC that replicates the same functionality.

Create an admin plug-in for Index Site Content

1. Open the solution with the **AlloyAdvanced** project.
2. In **~\Business\Initialization**, add an **Episerver | Initilization Module** named **IndexSiteContentInitializationModule**.
3. Add statements to map a route, as shown in the following code:

```
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using System.Web.Mvc;
using System.Web.Routing;

namespace AlloyAdvanced.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class IndexSiteContentInitializationModule
        : IInitializableModule
    {
        private bool initialized = false;

        public void Initialize(InitializationEngine context)
        {
            if (!initialized)
            {
                RouteTable.Routes.MapRoute(
                    name: "IndexSiteContent",
                    url: "indexsitecontent/{action}",
                    defaults: new { controller = "IndexSiteContent",
                                    action = "Index" });

                initialized = true;
            }
        }
```

```
        public void Uninitialize(InitializationEngine context) { }
    }
}
```

4. In **~\Controllers**, add a new **MVC 5 Controller - Empty** named **IndexSiteContentController**.
5. Add statements to register the controller as a plug-in for the **CMS** | **Admin** | **Admin** | **Tools** section, authorized only for CMS administrators, as shown in the following code:

The last time a complete indexing was executed is not automatically recorded. Dynamic Data Store (DDS) is used by the Web Forms page to record this information, so our plug-in can do the same.

```
using EPiServer.Core;
using EPiServer.Data.Dynamic;
using EPiServer.PlugIn;
using EPiServer.Search;
using EPiServer.ServiceLocation;
using System;
using System.Linq;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    [Authorize(Roles = "CmsAdmins")]
    [GuiPlugIn(
        Area = PlugInArea.AdminMenu,
        Url = "~/indexsitecontent",
        DisplayName = "Index Site Content")]
    public class IndexSiteContentController : Controller
    {
        public ActionResult Index(string startIndexing)
        {
            // indexing information is stored in DDS, so
            // we need to read the date of when
            // the last reindexing was executed.
            var indexings = Store.Items<IndexingInformation>();
            var mostRecent = indexings.OrderByDescending(
                i => i.ExecutionDate).FirstOrDefault();
            ViewBag.LastIndexing = mostRecent?.ExecutionDate;

            // is the admin clicks the Start Indexing button
            if (!string.IsNullOrWhiteSpace(startIndexing))
            {
                // execute a reindex
                ServiceLocator.Current.GetInstance
                    <IReIndexManager>().ReIndex();

                // save the date of when the reindex executed
                var information = new IndexingInformation
                {
                    ExecutionDate = DateTime.Now,
                    ResetIndex = true
                };
                Store.Save(information);
                ViewBag.LastIndexing = information.ExecutionDate;
            }

            return View();
        }

        private static DynamicDataStore Store
        {
            get
```

```
                {
                    return (DynamicDataStoreFactory.Instance.GetStore(
                        typeof(IndexingInformation)) ??
                        DynamicDataStoreFactory.Instance.CreateStore(
                        typeof(IndexingInformation)));
                }
            }
        }
    }
```

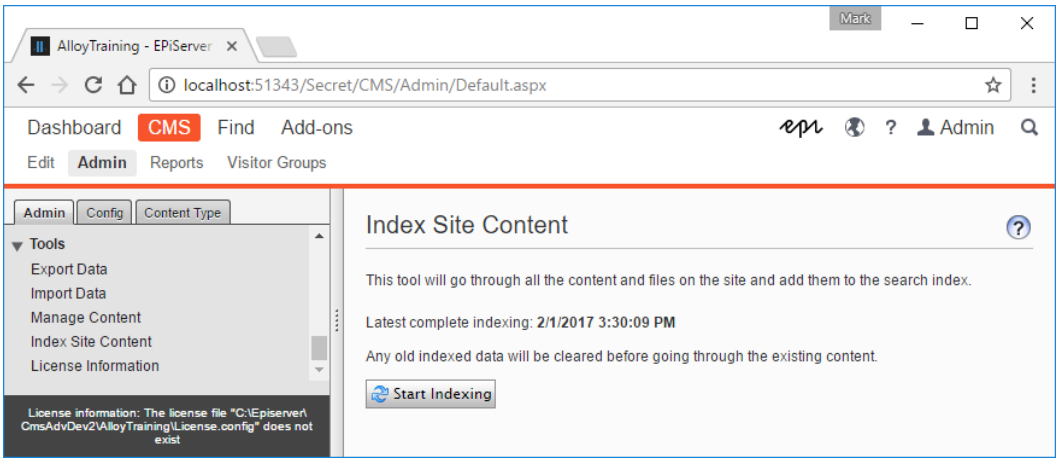
6. In **~\Views**, add a new folder named **IndexSiteContent**.
7. In **~\Views\IndexSiteContent**, add an **Episerver | Page Partial View (MVC Razor)** named **Index.cshtml**.
8. Modify its contents, as shown in the following markup:

The following markup is a good template for any plug-ins that you create because it uses the same layout, styles, and scripts that Episerver does.

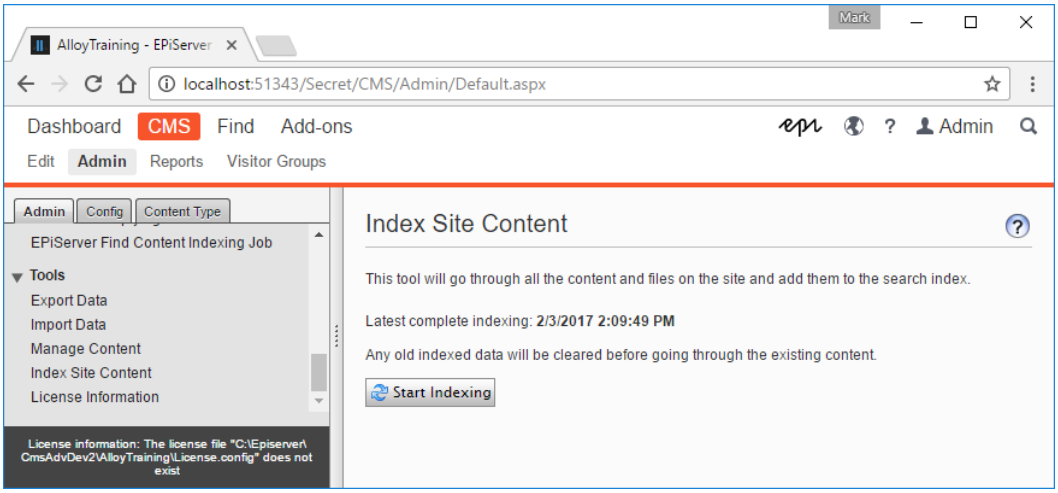
```
@using EPiServer.Framework.Web.Resources
@{
    Layout = null;
}
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
    <title>Index Site Content</title>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <!-- Shell -->
    @Html.Raw(ClientResources.RenderResources("ShellCore"))
    <!-- LightTheme -->
    @Html.Raw(ClientResources.RenderResources("ShellCoreLightTheme"))
    <link href="../../../../App_Themes/Default/Styles/system.css" type="text/css"
rel="stylesheet">
    <link href="../../../../App_Themes/Default/Styles/ToolButton.css" type="text/css"
rel="stylesheet">
</head>
<body id="body">
    @Html.Raw(Html.ShellInitializationScript())
    @using (Html.BeginForm("Index", "IndexSiteContent", FormMethod.Post))
    {
        <div class="epi-contentContainer epi-padding">
            <div class="epi-contentArea">
                <h1 class="EP-prefix">
                    Index Site Content<a
onclick="window.open(&#39;http://webhelp.episerver.com/16-
9/EpiserverUserGuide_csh.htm#indexcontent&#39;,&#39;_blank&#39;, &#39;scrollbars=yes,
height=500, location=no, menubar=no, resizable=yes, toolbar=no, width=840&#39;);return
false;" title="Help" href="http://webhelp.episerver.com/16-
9/EpiserverUserGuide_csh.htm#indexcontent" target="_blank"><img class="EPEdit-
CommandTool" align="absmiddle" src="/App_Themes/Default/Images/Tools/Help.png"
border="0" alt="Help" /></a>
                </h1><p class="EP-systemInfo">This tool will go through all the
content and files on the site and add them to the search index.</p>
                <div id="FullRegion_ValidationSummary" class="EP-validationSummary"
style="color:Black;display:none;"></div>
            </div>
            <div class="epi-contentArea epi-formArea">
                <p>
                    Latest complete indexing:
                    <strong>@ViewBag.LastIndexing</strong>
                </p>
```

```
                Any old indexed data will be cleared before going through the existing
content.
                <div class="epi-buttonDefault">
                    <span class="epi-cmsButton">
                        <input class="epi-cmsButton-text epi-cmsButton-tools epi-
cmsButton-Refresh" type="submit" name="startIndexing" id="startIndexing" value="Start
Indexing" onmouseover="EPi.ToolButton.MouseDownHandler(this)"
onmouseout="EPi.ToolButton.ResetMouseDownHandler(this)" />
                    </span>
                </div>
            </div>
        </div>
    }
</body>
</html>
```

9. Start the site, and log in as **Admin**.
10. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Index Site Content**, as shown in the following screenshot:



11. Click **Start Indexing**, and note the **Last complete indexing** has updated, as shown in the following screenshot:



Exercise F4 – Creating a report plug-in

In this exercise, you will create a report to show the freshness of content.

Prerequisites: complete Exercise 0.1.

Content on web sites should be kept "fresh" to encourage visitors to keep coming back. To help content editors know what pages are in most need of "refreshing", it would be handy to have a report that can be displayed in the browser, and exported as an Excel spreadsheet.

The report will include:

- Top ten pages that are the freshest.
- Top ten pages that are the *least* fresh.
- The content editor heroes who have kept the most pages up-to-date in the past week, month, and year.

EPPlus is an open source NuGet package for programatically generating Microsoft Excel files.

Create a content freshness report

1. Open the solution with the **AlloyAdvanced** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
3. Set **Package source** to **All**.
4. Enter the following command:

   ```
   Install-Package -ProjectName AlloyAdvanced EPPlus
   ```

5. In **~\Business\Initialization**, add an **Episerver** | **Initilization Module** named **PageFreshnessReportInitializationModule**.
6. Add statements to map a route, as shown in the following code:

```
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using System.Web.Mvc;
using System.Web.Routing;

namespace AlloyAdvanced.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class PageFreshnessReportInitializationModule
        : IInitializableModule
    {
        private bool initialized = false;

        public void Initialize(InitializationEngine context)
        {
            if (!initialized)
            {
                RouteTable.Routes.MapRoute(
                    name: "PageFreshnessReport",
                    url: "pagefreshnessreport/{action}",
                    defaults: new { controller = "PageFreshnessReport",
                                    action = "Index" });

                initialized = true;
            }
        }
```

```
            public void Uninitialize(InitializationEngine context) { }
        }
    }
```

7. In **~\Models\ViewModels**, add a new class named **PageFreshnessReportViewModel**.
8. Add statements to define some properties for the report, as shown in the following code:

```
using EPiServer.DataAbstraction;
using System.Collections.Generic;

namespace AlloyAdvanced.Models.ViewModels
{
    public class PageFreshnessReportViewModel
    {
        public string[] Administrators { get; set; }
        public string[] Editors { get; set; }
        public string SelectedUser { get; set; }
        public bool ShowReport { get; set; }
        public IEnumerable<ContentVersion> Top10FreshestPages { get; set; }
        public IEnumerable<ContentVersion> Top10LeastFreshPages { get; set; }
        public string HeroOfTheWeek { get; set; }
        public string HeroOfTheMonth { get; set; }
        public string HeroOfTheYear { get; set; }
    }
}
```

9. In **~\Controllers**, add a new **MVC 5 Controller - Empty** named **PageFreshnessReportController**.
10. Add statements to register the controller as a plug-in for the **CMS** | **Reports** | **Training Reports** | **Page Freshness** section, authorized only for CMS editors or administrators, as shown in the following code:

```
using AlloyAdvanced.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.PlugIn;
using EPiServer.ServiceLocation;
using EPiServer.Shell.Security;
using EPiServer.Web.Mvc.Html;
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    [Authorize(Roles = "CmsAdmins,CmsEditors")]
    [GuiPlugIn(
        Area = PlugInArea.ReportMenu,
        Url = "~/pagefreshnessreport",
        Category = "Training Reports",
        DisplayName = "Page Freshness")]
    public class PageFreshnessReportController : Controller
    {
        public ActionResult Index(string changedBy, string showReport, string
exportReport)
        {
            var roles = ServiceLocator.Current.GetInstance<UIRoleProvider>();

            var model = new PageFreshnessReportViewModel
```

```
        {
            Administrators = roles.GetUsersInRole("WebAdmins").ToArray(),
            Editors = roles.GetUsersInRole("WebEditors").ToArray(),
            SelectedUser = changedBy,
            ShowReport = !string.IsNullOrWhiteSpace(showReport)
        };

        if (model.ShowReport || (!string.IsNullOrWhiteSpace(exportReport)))
        {
            var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
            var versionRepo = ServiceLocator.Current
                .GetInstance<IContentVersionRepository>();

            var childRefs = loader.GetDescendents(ContentReference.StartPage)
                .Union(new ContentReference[] { ContentReference.StartPage });

            var children = childRefs.Select(r => versionRepo.LoadPublished(r));

            if (string.IsNullOrWhiteSpace(model.SelectedUser) ||
                model.SelectedUser == "Anyone")
            {
                model.Top10FreshestPages = children
                    .OrderByDescending(c => c.Saved).Take(10);

                model.Top10LeastFreshPages = children
                    .OrderBy(c => c.Saved).Take(10);
            }
            else
            {
                model.Top10FreshestPages = children
                    .Where(c => c.SavedBy == model.SelectedUser)
                    .OrderByDescending(c => c.Saved).Take(10);

                model.Top10LeastFreshPages = children
                    .Where(p => p.SavedBy == model.SelectedUser)
                    .OrderBy(c => c.Saved).Take(10);
            }

            model.HeroOfTheWeek = children
                .Where(p => p.Saved.AddDays(7) >= DateTime.Now)
                .OrderByDescending(p => p.Saved).FirstOrDefault()?.SavedBy;

            model.HeroOfTheMonth = children
                .Where(p => p.Saved.AddMonths(1) >= DateTime.Now)
                .OrderByDescending(p => p.Saved).FirstOrDefault()?.SavedBy;

            model.HeroOfTheYear = children
                .Where(p => p.Saved.AddYears(1) >= DateTime.Now)
                .OrderByDescending(p => p.Saved).FirstOrDefault()?.SavedBy;

            if (!string.IsNullOrWhiteSpace(exportReport))
            {
                Export(model, HttpContext.Response);
            }
        }

        return View(model);
    }

    private void Export(PageFreshnessReportViewModel model,
        HttpResponseBase response)
    {
```

```
            using (var package = new ExcelPackage())
            {
                AddWorksheet(package, "Top10FreshestPages", model.Top10FreshestPages);
                AddWorksheet(package, "Top10LeastFreshPages",
                    model.Top10LeastFreshPages);

                response.ContentType =
                  "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

                response.AddHeader("content-disposition",
                    $"attachment; filename=pages{0}.xlsx",
                    DateTime.Now.ToString("yyyyMMdd")));

                response.BinaryWrite(package.GetAsByteArray());
                response.Flush();
                response.End();
            }
        }

        private void AddWorksheet(ExcelPackage package, string name,
            IEnumerable<ContentVersion> pages)
        {
            ExcelWorksheet ws = package.Workbook.Worksheets.Add(name);

            ws.Cells[1, 1].Value = "PageId";
            ws.Cells[1, 2].Value = "PageName";
            ws.Cells[1, 3].Value = "PageUrl";
            ws.Cells[1, 4].Value = "Published Date";

            ws.Row(1).Style.Font.Bold = true;
            ws.Row(1).Style.Locked = true;

            int row = 2;

            foreach (var page in pages)
            {
                ws.Cells[row, 1].Value = page.ContentLink.ID;
                ws.Cells[row, 2].Value = page.Name;
                ws.Cells[row, 3].Value = Url.ContentUrl(page.ContentLink);
                ws.Cells[row, 4].Value = page.Saved.ToString("yyyy-MM-dd HH:mm");
                row++;
            }
        }
    }
}
```

11. In **~\Views**, add a new folder named **PageFreshnessReport**.
12. In **~\Views\PageFreshnessReport**, add an **Episerver | Page Partial View (MVC Razor)** named **Index.cshtml**.
13. Modify its contents, as shown in the following markup:

The following markup is a good template for any report plug-ins that you create because it uses the same layout, styles, and scripts that Episerver does.

```
@model PageFreshnessReportViewModel
@using EPiServer.Framework.Web.Resources
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
```

```
<head>
    <title>@Html.Translate("/reportcenter/report[@name='Page
Freshness']/name")</title>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <!-- Shell -->
    @Html.Raw(ClientResources.RenderResources("ShellCore"))
    <!-- LightTheme -->
    @Html.Raw(ClientResources.RenderResources("ShellCoreLightTheme"))
    <link href="../../../../App_Themes/Default/Styles/system.css" type="text/css"
rel="stylesheet">
    <link href="../../../../App_Themes/Default/Styles/ToolButton.css" type="text/css"
rel="stylesheet">
</head>
<body>
    @Html.Raw(Html.ShellInitializationScript())
    <div class="epi-contentContainer epi-padding">
        <div class="epi-contentArea">
            <div class="EP-systemImage" style="background-image:
url('/App_Themes/Default/Images/ReportCenter/ChangedPages.gif');">
                <h1 class="EP-prefix">
                    @Html.Translate("/reportcenter/report[@name='Page
Freshness']/name")
                    <a onclick="window.open(&#39;http://webhelp.episerver.com/16-
9/EpiserverUserGuide_csh.htm#changedpages&#39;,&#39;_blank&#39;, &#39;scrollbars=yes,
height=500, location=no, menubar=no, resizable=yes, toolbar=no, width=840&#39;);return
false;" title="Help" href="http://webhelp.episerver.com/16-
9/EpiserverUserGuide_csh.htm#changedpages" target="_blank"><img class="EPEdit-
CommandTool" align="absmiddle" src="/App_Themes/Default/Images/Tools/Help.png"
border="0" alt="Help" /></a>
                </h1>
                <p class="EP-systemInfo">
                    @Html.Translate("/reportcenter/report[@name='Page
Freshness']/description")
                </p>
            </div>
            <div id="FullRegion_ValidationSummary" class="EP-validationSummary"
style="color: Black; display: none;">
            </div>
        </div>
        @using (Html.BeginForm("Index", "PageFreshnessReport", FormMethod.Post))
        {
            <script src="/Util/javascript/episerverscriptmanager.js"
type="text/javascript"></script>
            <script src="../../../../javascript/system.js"
type="text/javascript"></script>
            <script src="../../../../javascript/dialog.js"
type="text/javascript"></script>
            <script src="../../../../javascript/system.aspx"
type="text/javascript"></script>
            <input type="hidden" id="doExport" name="doExport" value="False">
            <div class="epi-formArea">
                <fieldset>
                    <legend>
                        Report Criteria
                    </legend>
                    <div class="epi-size10">
                        <label for="changedBy">Changed by</label>
                        <select name="changedBy" id="changedBy">
                            <option value="Anyone">Anyone</option>
                            <optgroup label="Administrators">
                                @foreach (string user in Model.Administrators)
                                {
```

```
                                    <option value="@user" @(user == Model.SelectedUser
? "selected=selected" : "")>@user</option>
                                }
                            </optgroup>
                            <optgroup label="Editors">
                                @foreach (string user in Model.Editors)
                                {
                                    <option value="@user" @(user == Model.SelectedUser
? "selected=selected" : "")>@user</option>
                                }
                            </optgroup>
                        </select>
                    </div>
                </fieldset>

                <div class="epitoolbuttonrow">
                    <span class="epi-cmsButton"><input class="epi-cmsButton-text epi-
cmsButton-tools epi-cmsButton-Report" type="submit" name="showReport" id="showReport"
value="Show Report" onmouseover="EPi.ToolButton.MouseDownHandler(this)"
onmouseout="EPi.ToolButton.ResetMouseDownHandler(this)" /></span>
                    <span class="epi-cmsButton"><input class="epi-cmsButton-text epi-
cmsButton-tools epi-cmsButton-Report" type="submit" name="exportReport"
id="exportReport" value="Export Report"
onmouseover="EPi.ToolButton.MouseDownHandler(this)"
onmouseout="EPi.ToolButton.ResetMouseDownHandler(this)" /></span>
                </div>

            </div>
        }

        @if (Model.ShowReport)
        {
            <div class="epi-floatLeft epi-marginVertical-small"><h4>Heroes</h4></div>
            <div class="epi-contentArea epi-clear">
                <div>
                    <dl>
                        <dd>Hero of the week</dd>
                        <dt>@Model.HeroOfTheWeek</dt>
                        <dd>Hero of the month</dd>
                        <dt>@Model.HeroOfTheMonth</dt>
                        <dd>Hero of the year</dd>
                        <dt>@Model.HeroOfTheYear</dt>
                    </dl>
                </div>
            </div>

            if (Model.Top10FreshestPages != null
                && Model.Top10FreshestPages.Count() > 0)
            {
                <div class="epi-floatLeft epi-marginVertical-small"><h4>Top 10
Freshest Pages</h4></div>
                <div class="epi-contentArea epi-clear">
                    <div>
                        <table class="epi-default epi-default-legacy" cellspacing="0"
id="FullRegion_MainRegion_ReportView" style="border-style: None; width: 100%; border-
collapse: collapse;">
                            <tr>
                                <th scope="col">ID</th>
                                <th scope="col">Name</th>
                                <th scope="col">Link</th>
                                <th scope="col">Saved</th>
                                <th scope="col">Saved By</th>
```

```
                            </tr>
                            @foreach (var page in Model.Top10FreshestPages)
                            {
                                <tr>
                                    <td>@page.ContentLink.ID</td>
                                    <td>@page.Name</td>
                                    <td>@Url.ContentUrl(page.ContentLink)</td>
                                    <td>@page.Saved.ToString("yyyy-MM-dd HH:mm")</td>
                                    <td>@page.SavedBy</td>
                                </tr>
                            }
                        </table>
                    </div>
                </div>
            }

            if (Model.Top10LeastFreshPages != null &&
Model.Top10LeastFreshPages.Count() > 0)
            {
                <div class="epi-floatLeft epi-marginVertical-small"><h4>Top 10 Least
Fresh Pages</h4></div>
                <div class="epi-contentArea epi-clear">
                    <div>
                        <table class="epi-default epi-default-legacy" cellspacing="0"
id="FullRegion_MainRegion_ReportView" style="border-style: None; width: 100%; border-
collapse: collapse;">
                            <tr>
                                <th scope="col">ID</th>
                                <th scope="col">Name</th>
                                <th scope="col">Link</th>
                                <th scope="col">Saved</th>
                                <th scope="col">Saved By</th>
                            </tr>
                            @foreach (var page in Model.Top10LeastFreshPages)
                            {
                                <tr>
                                    <td>@page.ContentLink.ID</td>
                                    <td>@page.Name</td>
                                    <td>@Url.ContentUrl(page.ContentLink)</td>
                                    <td>@page.Saved.ToString("yyyy-MM-dd HH:mm")</td>
                                    <td>@page.SavedBy</td>
                                </tr>
                            }
                        </table>
                    </div>
                </div>
            }
        }

    </div>
    <script type="text/javascript">
        document.getElementById("exportReport").onclick = function () {
            document.getElementById("doExport").value = "True";
        };
        document.getElementById("showReport").onclick = function () {
            document.getElementById("doExport").value = "False";
        };
    </script>
</body>
</html>
```

14. In **~\Resources\LanguageFiles**, add an XML file named **PageFreshnessReport.xml**.

15. Modify its contents, as shown in the following markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <reportcenter>
      <report name="Page Freshness">
        <name>Page Freshness</name>
        <description>Displays the top 10 freshest and least fresh pages.</description>
      </report>
    </reportcenter>
  </language>
  <language name="Svenska" id="sv">
    <reportcenter>
      <report name="PageFreshnessReport">
        <name>Sida Friskhet</name>
        <description>Visar de 10 färskaste och minst färska sidor.</description>
      </report>
    </reportcenter>
  </language>
</languages>
```

Optional task: Create a localization report

Content on web sites that need to be localized into multiple languages can be tricky to manage. Which pages have been translated? Create a report that shows a list of pages that have not yet been localized into the web site's enabled languages.


Hint: use the **LanguageBranches** API.

Exercise F5 – Integrating with Tasks in the Navigation pane

In this exercise, you will create a plug-in that adds a new option to the Tasks list in the Navigation pane.

Prerequisites: complete Exercise 0.1.

Create a type of task

1. Open the solution with the **AlloyAdvanced** project.
2. Read the following blog post to implement the new type of task: https://niteco.com/blogs/plugging-into-the-tasks-list/

Module G – Episerver Find

Goal

The overall goal of the exercises in this module is to learn how to index content using *Episerver Find*.

You will:

- Register and configure an Episerver Find index.
- Configure, implement, and customize Episerver Find for CMS.

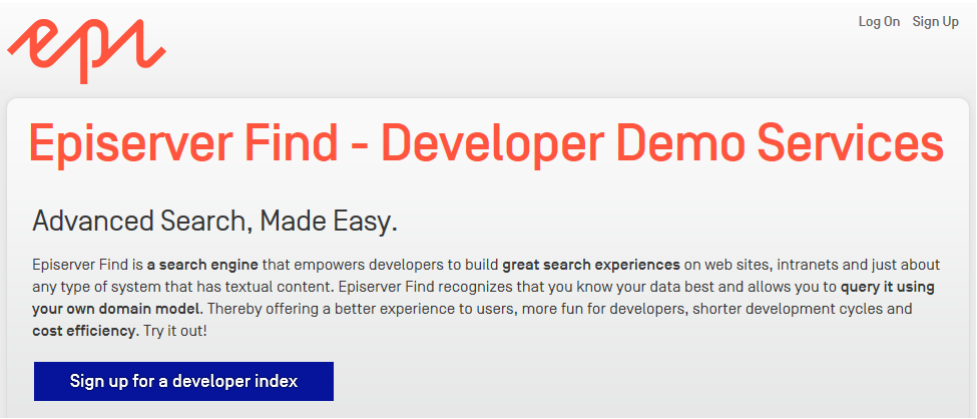
Exercise G1 – Implementing Episerver Find

In this exercise, you will configure an Episerver Find index for use with the AlloyAdvanced site, and then implement searching functionality using Episerver Find, including optimizations like Best Bets.

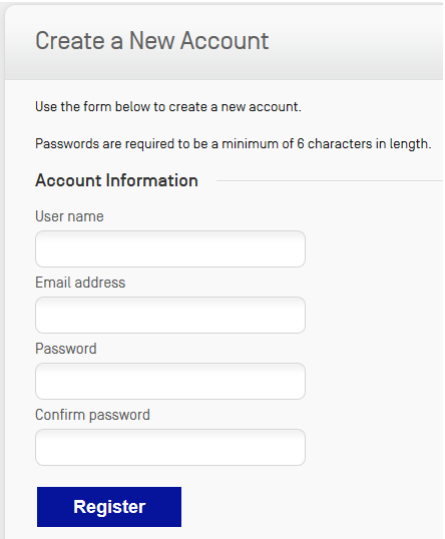
Prerequisites: complete Exercise 0.1.

Registering a Find account

1. Open a web browser and navigate to `http://find.episerver.com/`

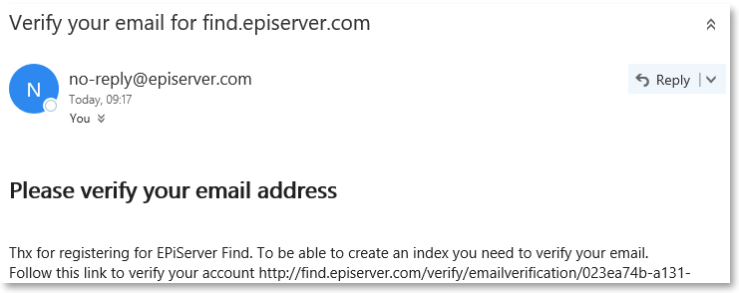


2. Click **Sign up for a developer index**, or **Sign Up** in the top right corner.



3. Fill in the required information, and click **Register**.

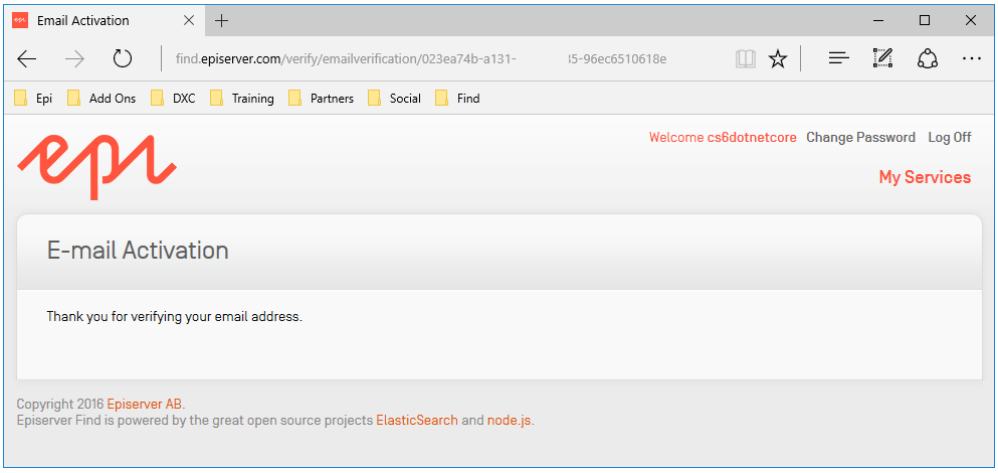
4. When the verification email arrives, copy and paste the link into your browser's address box and press ENTER to confirm your address and activate the index.



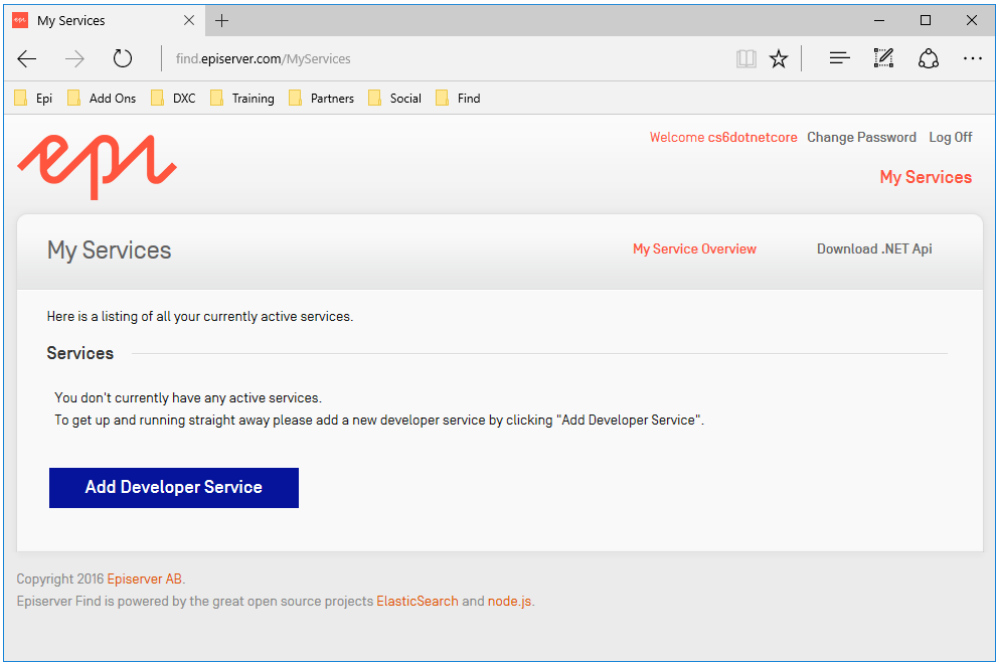
If you don't receive your verification email, check your junk mail folder!

Creating a developer index

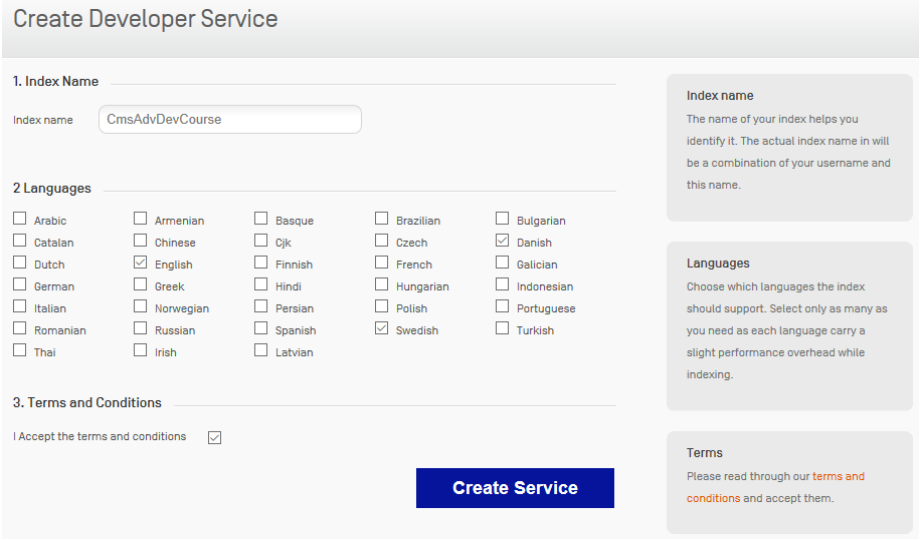
1. On the **E-mail Activation** verification page, click **My Services**.



2. In **My Services**, click **Add Developer Service**:

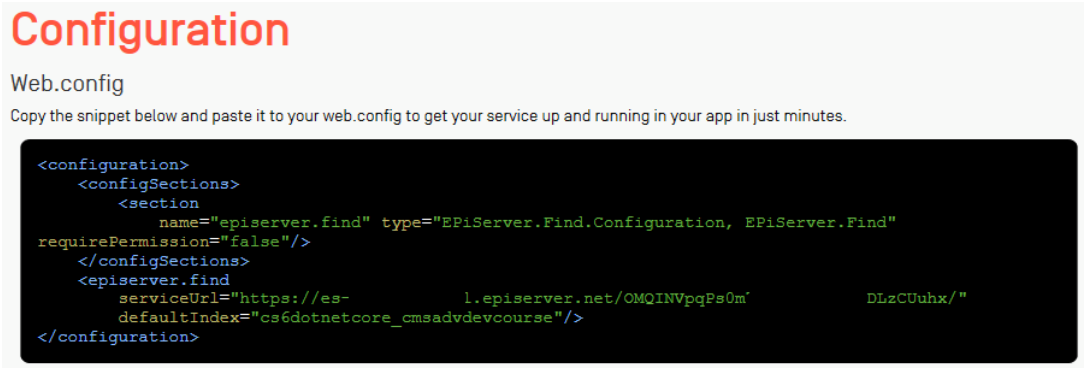


3. Fill in a name for your index. The name is technically irrelevant but should preferably represent what you're using the index for, such as, ***CustomerName*TestIndex**.
4. Select at least **English** from the list of languages. You may also select another language, preferably one that you know.
5. Check the terms and conditions checkbox, and click **Create Service**.



6. You should now see a list of details about your index. Note the **Status**: it will probably be **NotCreated**, as shown in the following screenshot. If you wait a minute and refresh the page it should say **CreatedWithStats**.

7. Note the **Configuration**, as shown in the following screenshot. In the next section, you will copy and paste this into the site's Web.config.



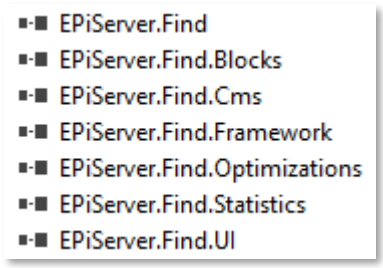
Installing and configuring Find for a CMS project

1. Open the solution with the **AlloyAdvanced** project.

2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
3. Enter the following command:

```
Install-Package -ProjectName AlloyAdvanced EPiServer.Find.Cms
```

4. In **Solution Explorer**, expand **References** and note the new assembly references:



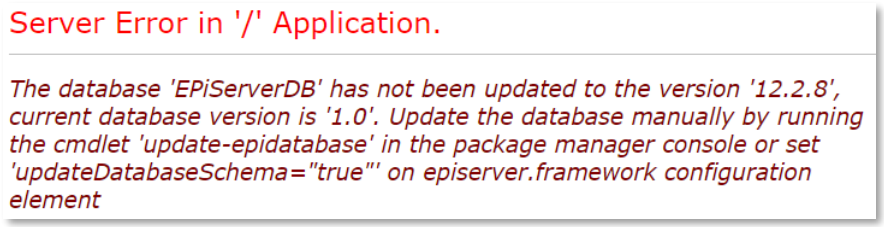
5. Open **~\Web.config**.
6. Add the following to the **<configSections>** element (you can copy and paste it from the Find Configuration page):

```
<section name="episerver.find" requirePermission="false"
         type="EPiServer.Find.Configuration, EPiServer.Find" />
```

7. Add the following to the **<configuration>** element (you must copy and paste it from your **Find Configuration** page because your **serviceURL** contains a unique private account key):

```
<episerver.find
    serviceUrl="https://es-?.episerver.net/OMQINVp...zCUuhx/"
    defaultIndex="cmsadvdevcourse"/>
```

8. Start the site.
9. Press *F5* to force a refresh, and you should see an exception, as shown in the following screenshot:

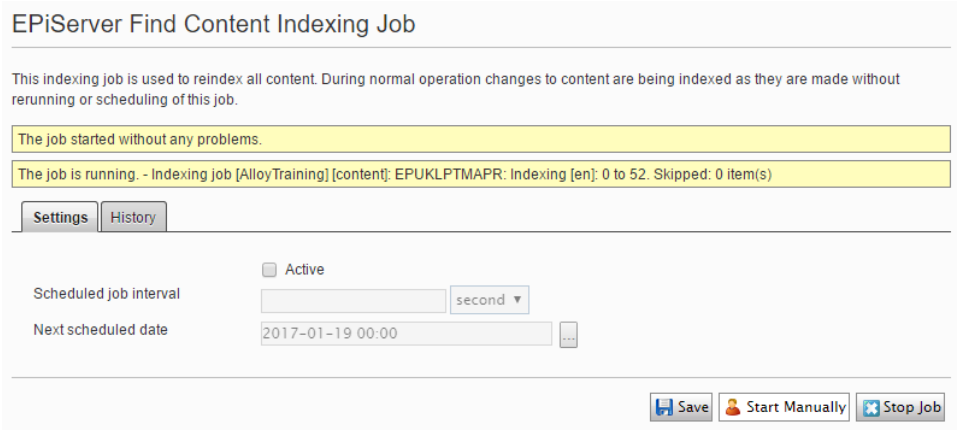


10. Close the browser.
11. Use **Package Manager Console** to run the `Update-EPiDatabase` command, and note it executes two SQL scripts to update **Find** components in the CMS database, as shown in the following output:

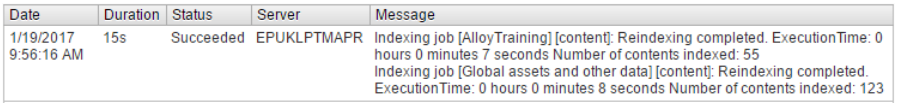
```
Processing C:\Episerver\CmsAdvDev\packages\EPiServer.Find.Cms.12.3.2\tools\epiupdates\sql\1.0.1.sql
Processing C:\Episerver\CmsAdvDev\packages\EPiServer.Find.Cms.12.3.2\tools\epiupdates\sql\12.2.8.sql
```

12. Start the site, and log in as **Admin**.

13. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **EPiServer Find Content Indexing Job**.
14. Click **Start Manually**, and wait for the job to complete, as shown in the following screenshot:

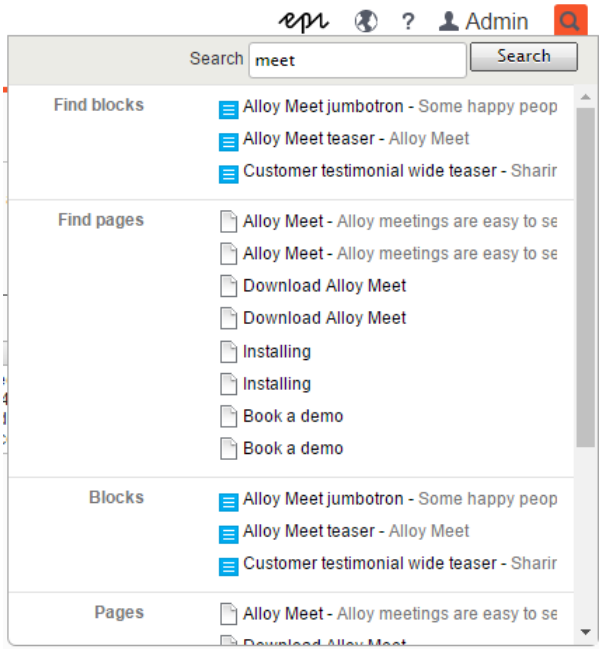


15. Click **History**. Note that lots of AlloyAdvanced content items were indexed, and lots of global assets and other data were indexed, as shown in the following screenshot:



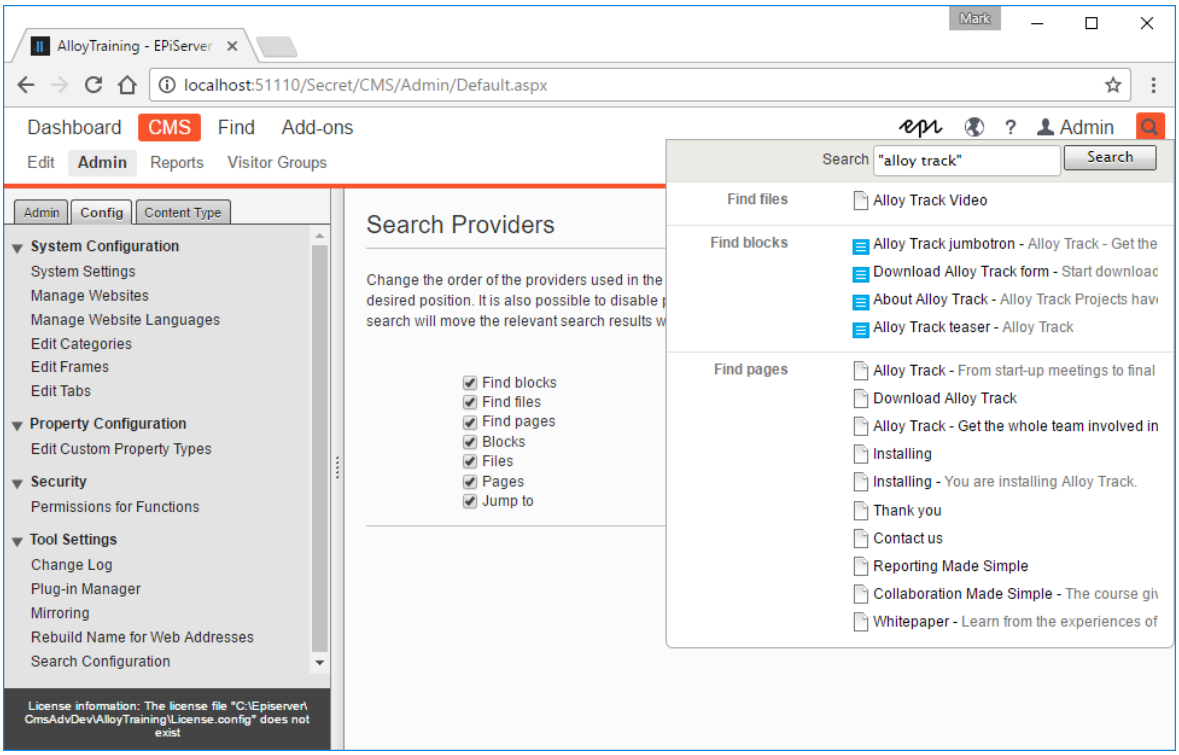
Date	Duration	Status	Server	Message
1/19/2017 9:56:16 AM	15s	Succeeded	EPUKLPTMAPR	Indexing job [AlloyTraining] [content]: Reindexing completed. ExecutionTime: 0 hours 0 minutes 7 seconds Number of contents indexed: 55 Indexing job [Global assets and other data] [content]: Reindexing completed. ExecutionTime: 0 hours 0 minutes 8 seconds Number of contents indexed: 123

16. In the **Global** menu, click inside the **Search** box, and enter **meet**. You should see multiple matches, including shared blocks and pages, with results shown from both Episerver Search (labeled **Blocks** and **Pages**) and Episerver Find (labeled **Find blocks** and **Find pages**), as shown in the following screenshot:



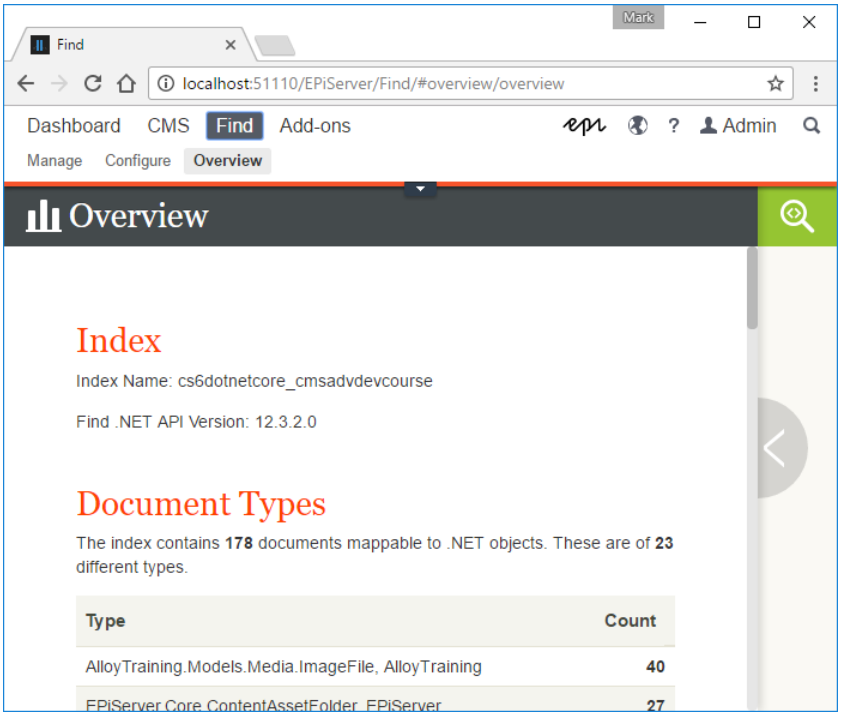
17. Navigate to **CMS** | **Admin** | **Config** | **Search Configuration**. Note how the **Search Providers** have check boxes to disable them in **Global Search**, and they can be dragged and dropped to change the

order in which they are called and displayed in the search results, as shown in the following screenshot:

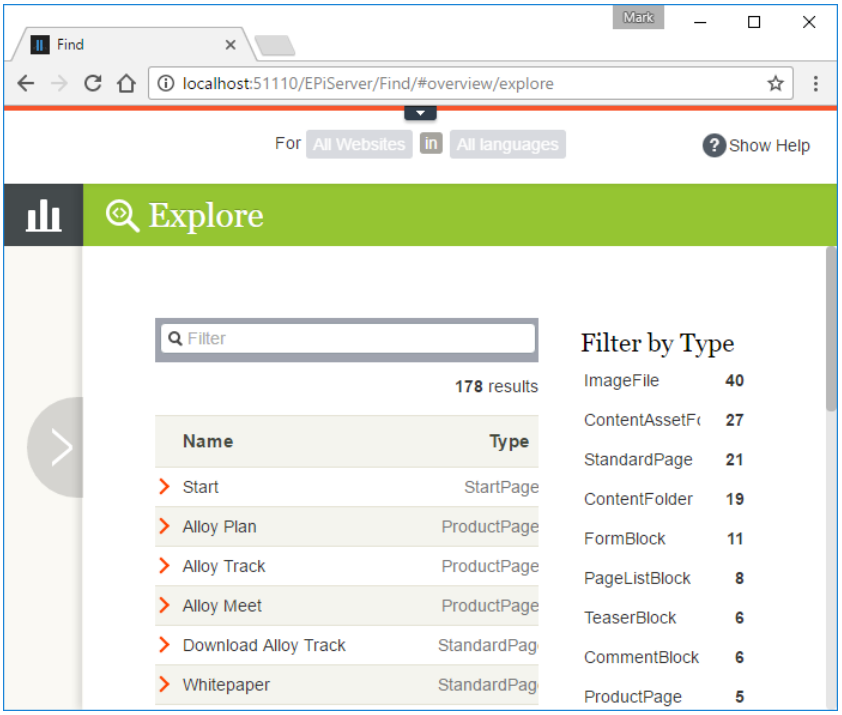


Reviewing the administrator's view of Find

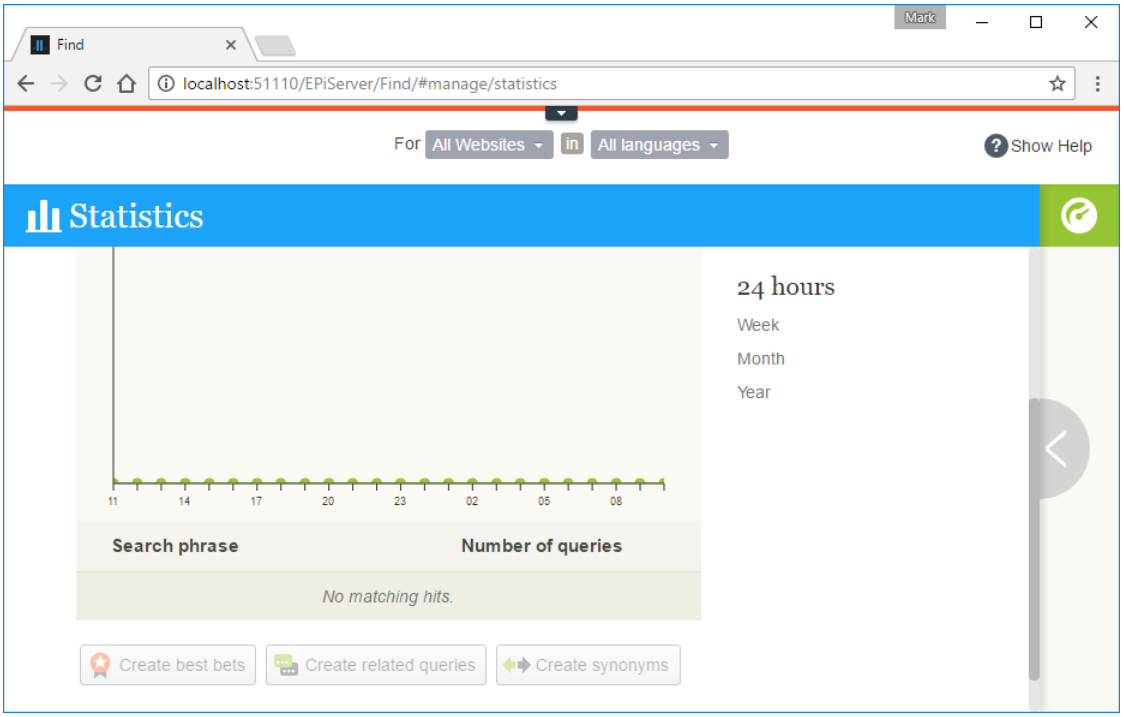
1. In the **Global** menu, navigate to **Find** | **Overview**, as shown in the following screenshot:



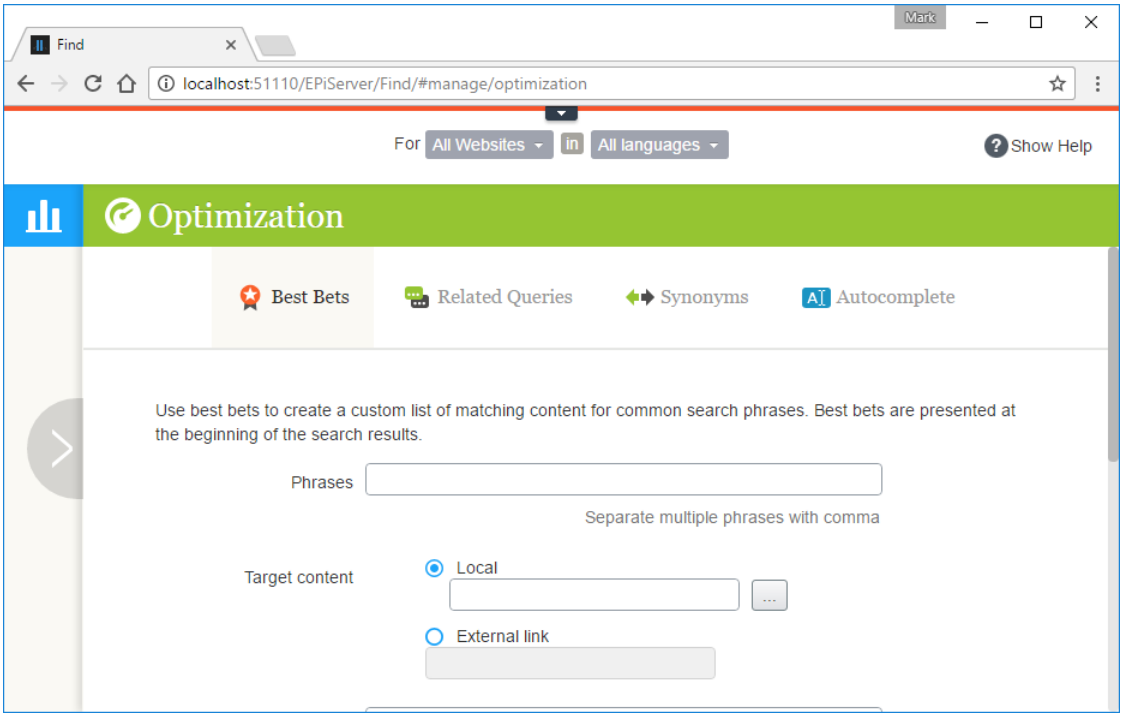
2. Click the green box to slide out the **Explore** area, as shown in the following screenshot:



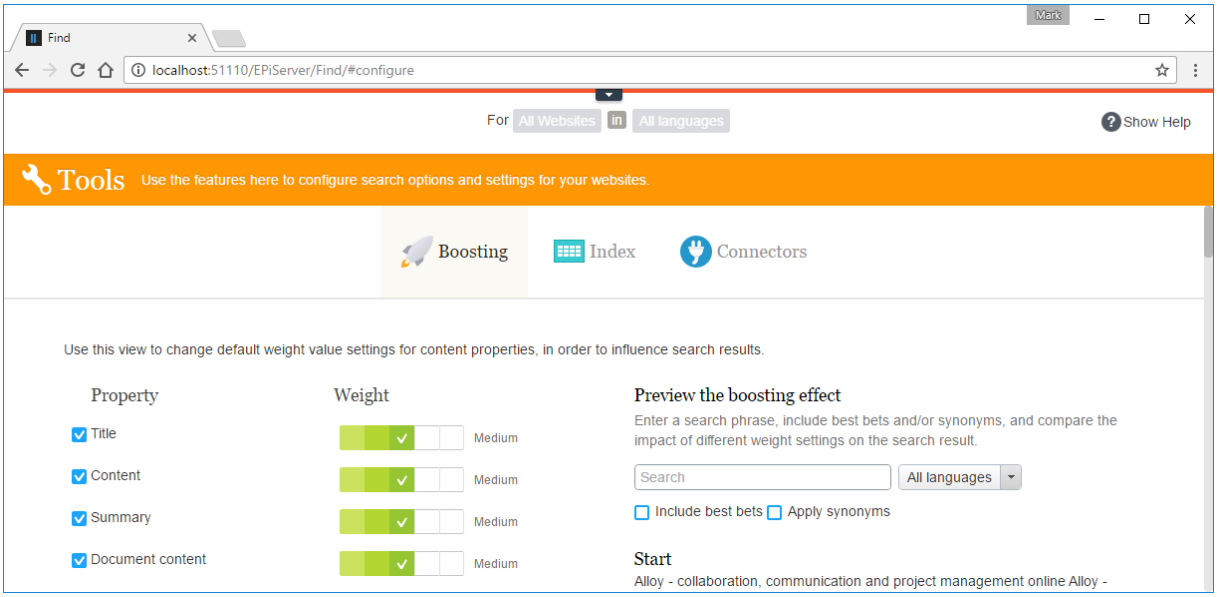
3. Navigate to **Find** | **Manage**, and note the **Statistics** area, as shown in the following screenshot:



4. Click the green box to slide out the **Optimization** area, as shown in the following screenshot:



5. Navigate to **Find** | **Configure**, and note the **Tools** area, as shown in the following screenshot:



Implementing indexed search for CMS with Find

You will implement basic free text search functionality using using Episerver Find's Unified Search. You will be using the SearchPage models that are already part of the Alloy site.

1. Open the solution with the **AlloyAdvanced** project.
2. Add a class to **~/Models/ViewModels** named **FindSearchPageViewModel**, as shown in the following code:

```
using AlloyAdvanced.Models.Pages;
using EPiServer.Find.UnifiedSearch;

namespace AlloyAdvanced.Models.ViewModels
{
```

```
        public class FindSearchPageViewModel : PageViewModel<SearchPage>
        {
            public FindSearchPageViewModel(SearchPage currentPage,
                string searchQuery) : base(currentPage)
            {
                SearchQuery = searchQuery;
            }

            public string SearchQuery { get; private set; }

            public UnifiedSearchResults Results { get; set; }
        }
    }
```

When creating controllers in the Alloy site, you should inherit from the site-specific class named `PageControllerBase<T>`, instead of the usual Episerver class named `PageController<T>`.

3. Add an Episerver **Page Controller (MVC)** to **~/Controllers** named **FindSearchPageController**, as shown in the following code. Note the template descriptor attribute to make this controller replace the existing one for **SearchPage** instances:

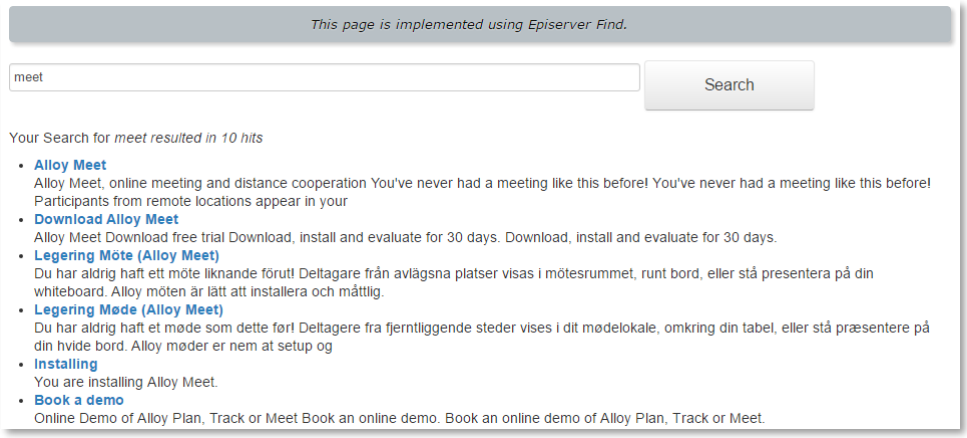
```
using AlloyAdvanced.Models.Pages;
using AlloyAdvanced.Models.ViewModels;
using EPiServer.Find;
using EPiServer.Find.Framework;
using EPiServer.Framework.DataAnnotations;
using System.Web.Mvc;

namespace AlloyAdvanced.Controllers
{
    [TemplateDescriptor(Default = true)]
    public class FindSearchPageController : PageControllerBase<SearchPage>
    {
        public ActionResult Index(SearchPage currentPage, string q)
        {
            var model = new FindSearchPageViewModel(currentPage, q);

            if (!string.IsNullOrWhiteSpace(q))
            {
                var unifiedSearch = SearchClient.Instance.UnifiedSearchFor(q);
                model.Results = unifiedSearch.GetResult();
            }
            return View(model);
        }
    }
}
```

4. Add a new folder to **~/Views** named **FindSearchPage**.
5. To the folder, add a new Episerver **Page Partial View (MVC Razor)** named **Index.cshtml**:

```
@model AlloyAdvanced.Models.ViewModels.FindSearchPageViewModel

<div class="alert alert-info">This page is implemented using Episerver Find.</div>

@using (Html.BeginForm(null, null,
    EPiServer.Editor.PageEditing.PageIsInEditMode ? FormMethod.Post : FormMethod.Get,
    new { @action = Model.Layout.SearchActionUrl }))
{
    <input type="text" tabindex="1" name="q" value="@Model.SearchQuery" />
    <input type="submit" tabindex="2" class="btn" value="Search" />
}
@if (Model.Results != null)
```

```
{
    <p>
        Your Search for <i> @Model.SearchQuery resulted
        in @Model.Results.Count() hits</i>
    </p>
    <div class="listResult">
        @foreach (var item in Model.Results)
        {
            <h4><a href="@item.Url">@Html.Raw(item.Title)</a></h4>
            @Html.Raw(item.Excerpt)
        }
    </div>
}
```

6. Start the site.
7. Search for **meet**. Note that if you completed the location exercises earlier, and you added the Swedish and Danish languages, then you will see matching results for multiple languages, as shown in the following screenshot:



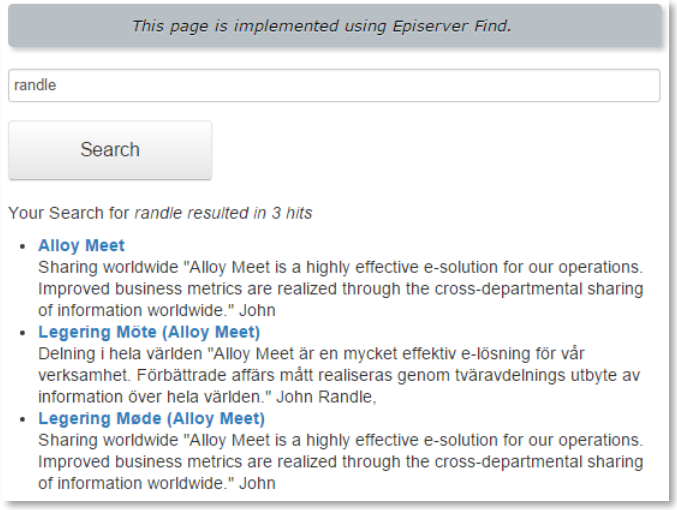
8. If you search for **randle** you will find no results, even though the Alloy Meet page appears to a visitor to have a teaser quote from *John Randle*. This is because, by default, although blocks are indexed for Global search by editors, if a block is added to a content area on a page, it is not indexed as part of that page.

9. Close the browser.
10. Open **~/Models/Blocks/TeaserBlock.cs**. Apply the **[IndexInContentAreas]** attribute to the block class, as shown in the following code:

```
[SiteImageUrl]
[EPiServer.Find.Cms.IndexInContentAreas]
public class TeaserBlock : SiteBlockData
```

11. Start the site, and log in as **Admin**.
12. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Episerver Find Content Indexing Job**.
13. Click **Start Manually**, and wait for the job to complete.
14. View the site as a visitor.

15. Search for **randle**. You will now find at least one hit, depending on if you localized the Alloy Meet page into Danish and Swedish, as shown in the following screenshot:



Blocks do not have URLs. When you apply the **[IndexInContentAreas]** attribute to a block, every page that contains that block in a content area we be included in the search results.

Applying Best Bets

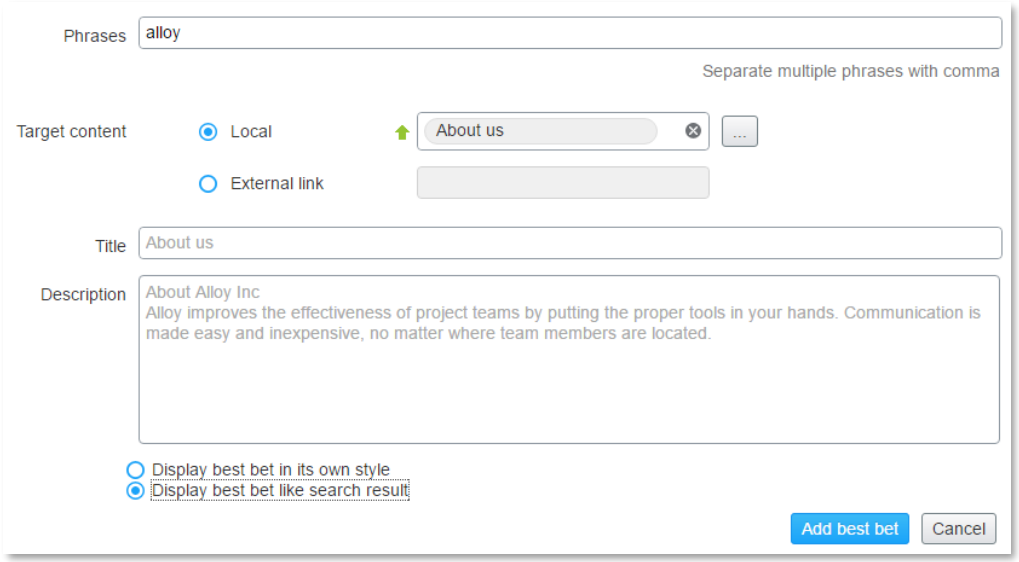
1. Open **~/Controllers/FindSearchPageController.cs**.
2. Import the following namespace to enable the **Track()** extension method:

```
using EPiServer.Find.Framework.Statistics;
```

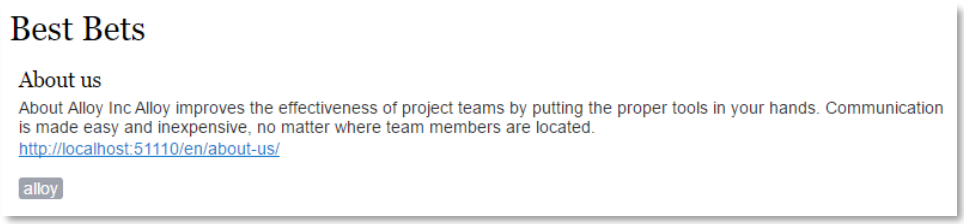
3. Modify the statement that gets the search results to track statistics for this query and apply best bets, as shown in the following code:

```
model.Results = unifiedSearch.Track().ApplyBestBets().GetResult();
```

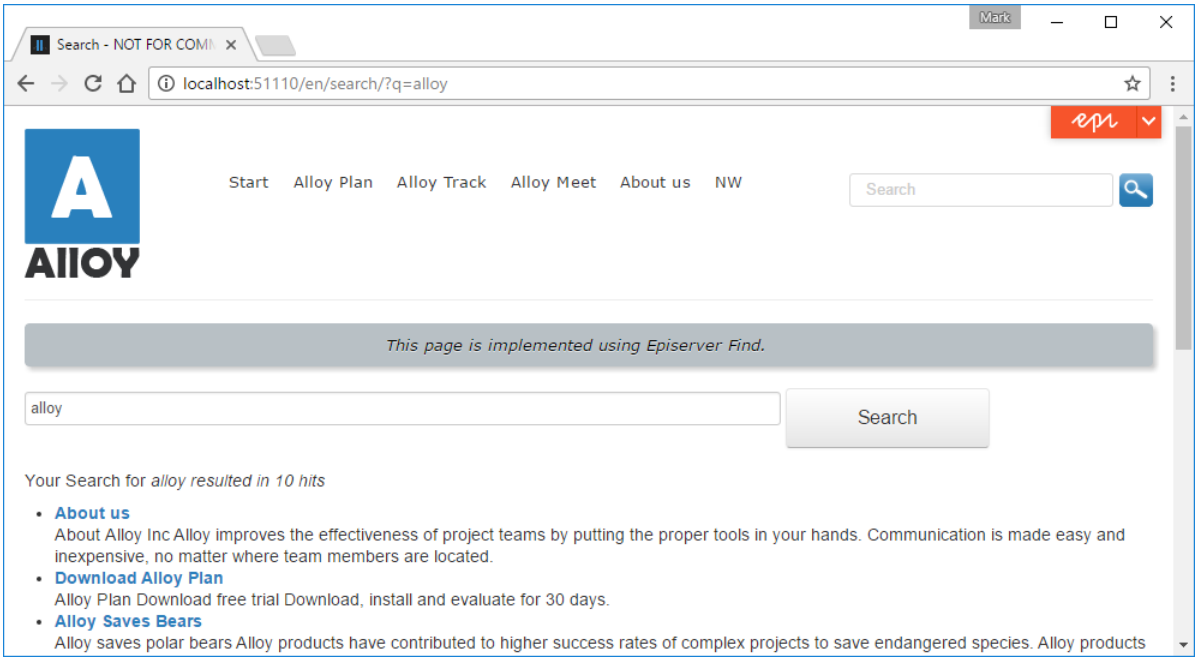
4. Start the site, and log in as **Admin**.
5. Navigate to **Find** | **Manage** | **Optimization** | **Best Bets**.
6. Create a best bet, as shown in the following screenshot, and click **Add best bet**:
   - Phrases: **alloy**
   - Target content, Local: **About us**
   - Title and Description: leave as default
   - **Display best bet like search result**



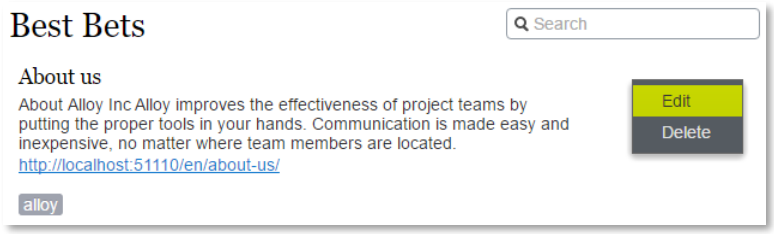
7. After saving the best bet, it will be shown in a list at the bottom of the page, as shown in the following screenshot:



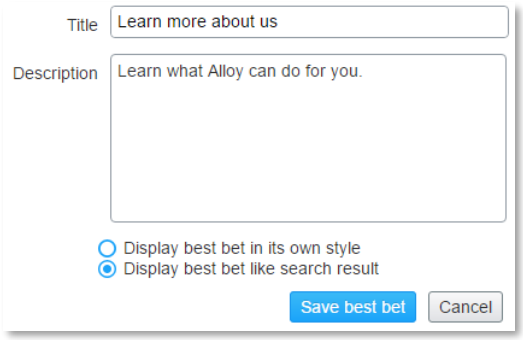
8. View the web site as a visitor, and search for **alloy**. The first result will be **About us**, as shown in the following screenshot:



9. Navigate to **Find** | **Manage** | **Optimization** | **Best Bets**.
10. In the **Best Bets** list, click the context menu for **About us**, and click **Edit**, as shown in the following screenshot:



11. Enter the following, as shown in the following screenshot, and click **Save best bet**:
    - Title: **Learn more about us**
    - Description: **Learn what Alloy can do for you.**



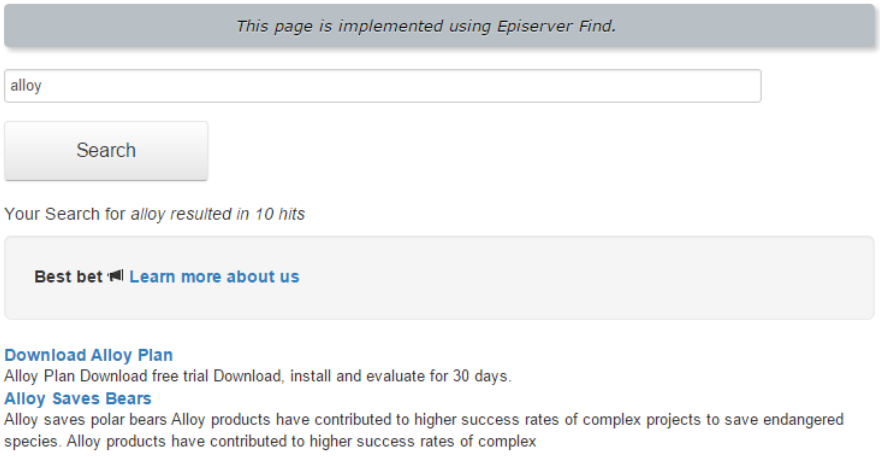
12. Close the browser.
13. Open **~/Views/FindSearchPage/Index.cshtml**.
14. Import the following namespace:

```
@using EPiServer.Find
```

15. Modify the output of each search result item, as shown in the following code:

```
if (item.IsBestBet())
{
    <div class="well well-large">
        <h4>Best bet <span class="icon-bullhorn"></span>
        <a href="@item.Url">@Html.Raw(item.Title)</a></h4>
    </div>
}
else
{
    <h4><a href="@item.Url">@Html.Raw(item.Title)</a></h4>
    @Html.Raw(@item.Excerpt)
}
```

16. View the site as a visitor and search for **alloy**. The best bet is found, as shown in the following screenshot:



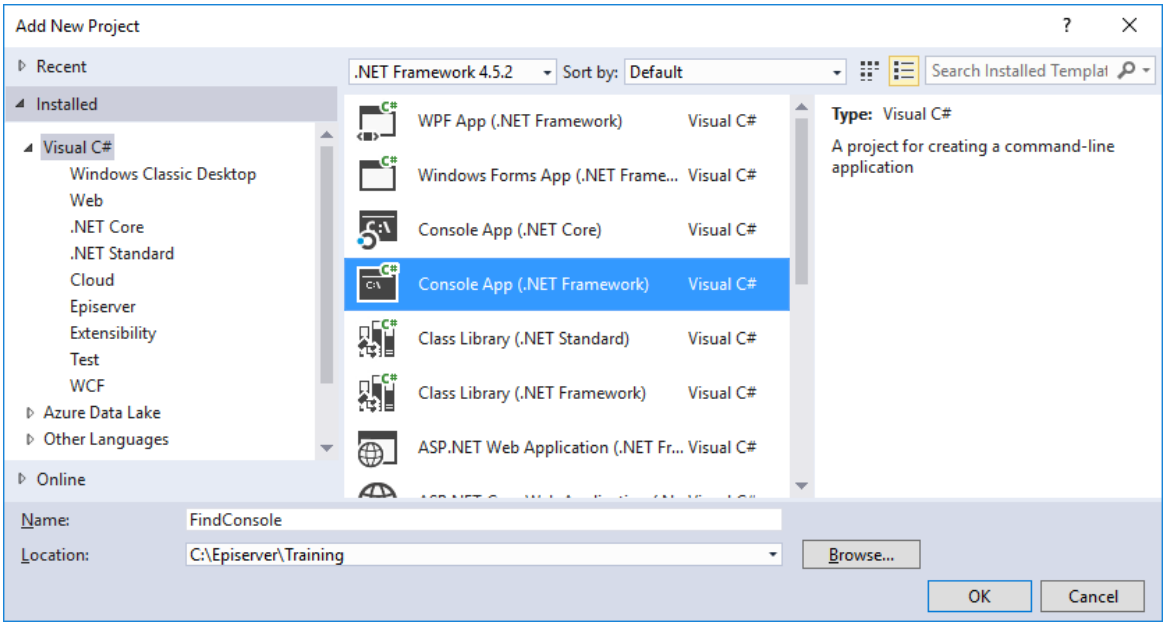
Exercise G2 – Exploring Episerver Find APIs

In this exercise, you will build a console application to explore some Find APIs.

Prerequisites: complete the first two tasks in **Exercise G1 – Implementing Episerver Find**: *Registering a Find account*, and *Creating a developer index*.

Creating a console application configured to use Episerver Find

1. Open the solution with the **AlloyAdvanced** project.
2. Add a new **Console App (.NET Framework)** project to the solution, named **FindConsole**, as shown in the following screenshot:



3. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
4. Set **Package source** to **All**, and enter the following command:

```
Install-Package -ProjectName FindConsole EPiServer.Find
```

5. Open **App.config**, and copy and paste the configuration of the `<episerver.find>` section, as shown in the following markup:

```
<configSections>
  <section name="episerver.find"
           type="EPiServer.Find.Configuration, EPiServer.Find"
           requirePermission="false" />
```

6. Copy and paste the configuration of your Find service and its index, like that shown in the following markup:

```
<episerver.find
  serviceUrl="https://es-eu-api01.episerver.net/PlpMSGvt...KWTJ8khTkxGRv"
  defaultIndex="episervertraining_index99999" />
```

7. Add the existing class named **FindExtensions.cs** from the solution ZIP. You will call these extension methods to save you time later in the exercise, for example, **Output** extension methods for **IClient** and **IndexResult**.
8. Add the existing class file named **People.cs** from the solution ZIP, and note that it defines **Person** and **Student** classes, as shown in the following code:

```
namespace FindConsole
{
```

```
        public class Person
        {
            [EPiServer.Find.Id] // without an ID, GUID string is generated
            public int PersonID { get; set; }
            public string FirstName { get; set; }
            public string LastName { get; set; }
        }

        public class Student : Person
        {
            public string Course { get; set; }
        }
    }
```

9. Add the existing class file named **Books.cs** from the solution ZIP, and note that it defines **Book** and **BookRepository** classes, as partially shown in the following code:

```
using System.Collections.Generic;

namespace FindConsole
{
    public class Book
    {
        [EPiServer.Find.Id]
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Author { get; set; }
    }

    public static class BookRepository
    {
        // information about the books is from http://thegreatestbooks.org/

        public static IEnumerable<Book> Books { get; private set; }

        static BookRepository()
        {
            var books = new HashSet<Book>();

            books.Add(new Book
            {
                Id = 1,
                Title = "The Lord of the Rings",
                Author = "J. R. R. Tolkien ",
                Description = "The Lord of the Rings is an epic high fantasy novel
written by philologist and Oxford University professor J. R. R. Tolkien. The story
began as a sequel to Tolkien's earlier, less complex children's fantasy novel The
Hobbit (1937), but eventually developed into a much larger work. It was written in
stages between 1937 and 1949, much of it during World War II. Although generally known
to readers as a trilogy, the work was initially intended by Tolkien to be one volume
of a two-volume set along with The Silmarillion; however, the publisher decided to
omit the second volume and instead published The Lord of the Rings in 1954-55 as three
books rather than one, for economic reasons. It has since been reprinted numerous
times and translated into many languages, becoming one of the most popular and
influential works in 20th-century literature."
            });
```

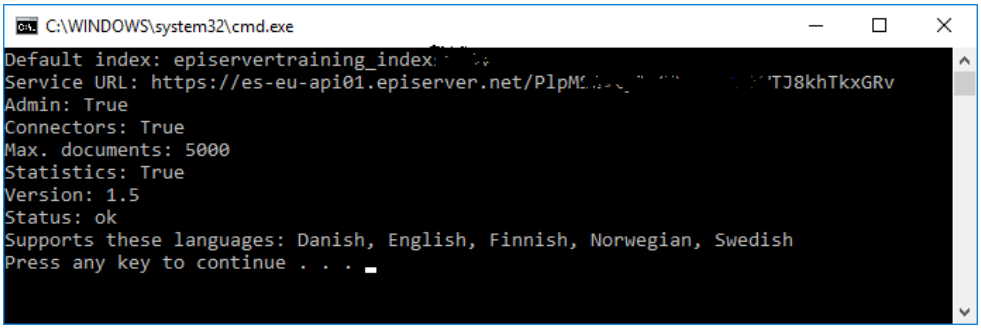

The BookRepository adds 21 books to the collection.

Adding objects to the Find index

1. Open **Program.cs**, and add the following statements to the **Main** method:

```
var client = Client.CreateFromConfig();
client.Output();
```

2. Start the console app, and review the output, as shown in the following screenshot:



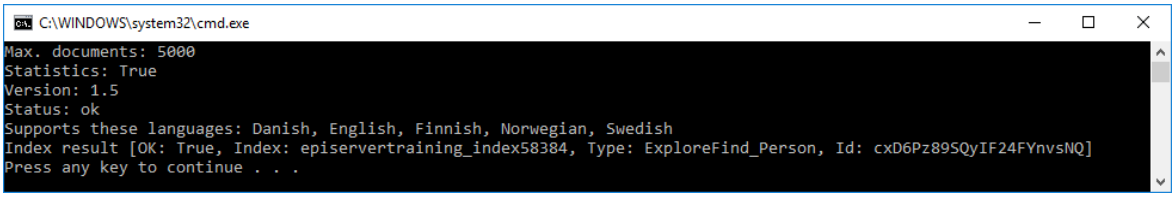
3. Close the console app.
4. Open **People.cs**, and add comment characters `//` to temporarily remove the **Id** attribute, as shown in the following code:

```
//[EPiServer.Find.Id] // without an ID, GUID string is generated
```

5. Add the following statements to the **Main** method:

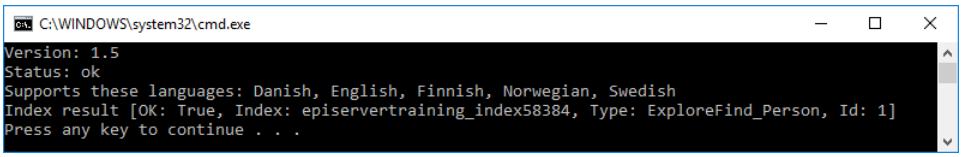
```
var p1 = new Person
{
    PersonID = 1,
    FirstName = "Alice",
    LastName = "Smith"
};

var response = client.Index(p1);
response.Output();
```

6. Start the console app and review the output, as shown in the following screenshot:



Episerver Find generates a GUID for newly indexed objects to act as the Document ID for identifying that object. If you want to supply your own value, we must tell Episerver Find which class member to use.

7. Open **People.cs**, and remove the comment characters.
8. Start the console app and review the output, as shown in the following screenshot:



The equivalent of the "primary key" for an item stored in an Episerver Find index is a combination of the **Type** and **Id**, e.g. **ExploreFind_Person** and **1**.

9. Add the following statements to the end of the **Main** method:

```
var s1 = new Student
{
    PersonID = 2,
    FirstName = "Bob",
    LastName = "Jones",
    Course = "CMS Advanced Development"
};

response = client.Index(s1, command =>
{
    command.Refresh = true; // so it appears in results immediately
    command.TimeToLive = TimeSpan.FromMinutes(30);
    command.Id = s1.PersonID; // manually set the DocumentId for the item
});
response.Output();
```

When indexing an object, you can optionally specify a command that controls features like: forcing an immediate refresh of the index, removing the object from the index automatically after a set time period (useful for testing), and explicitly setting the value of the Document ID.

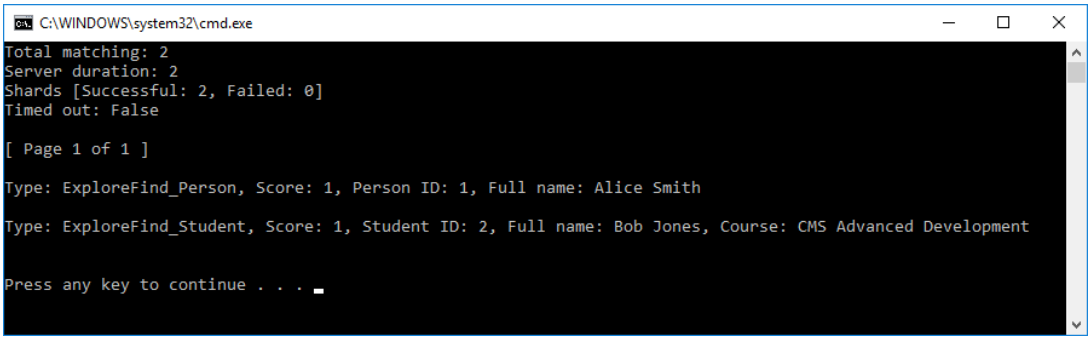
10. Start the console app and review the output.

Querying the Find index for people

1. Comment out the statements that index the person and student by using `/*` and `*/`.
2. Add the following statements to the end of the **Main** method:

```
ITypeSearch<Person> results = client.Search<Person>();
results.Output();
```

3. Start the console app and review the output, as shown in the following screenshot:



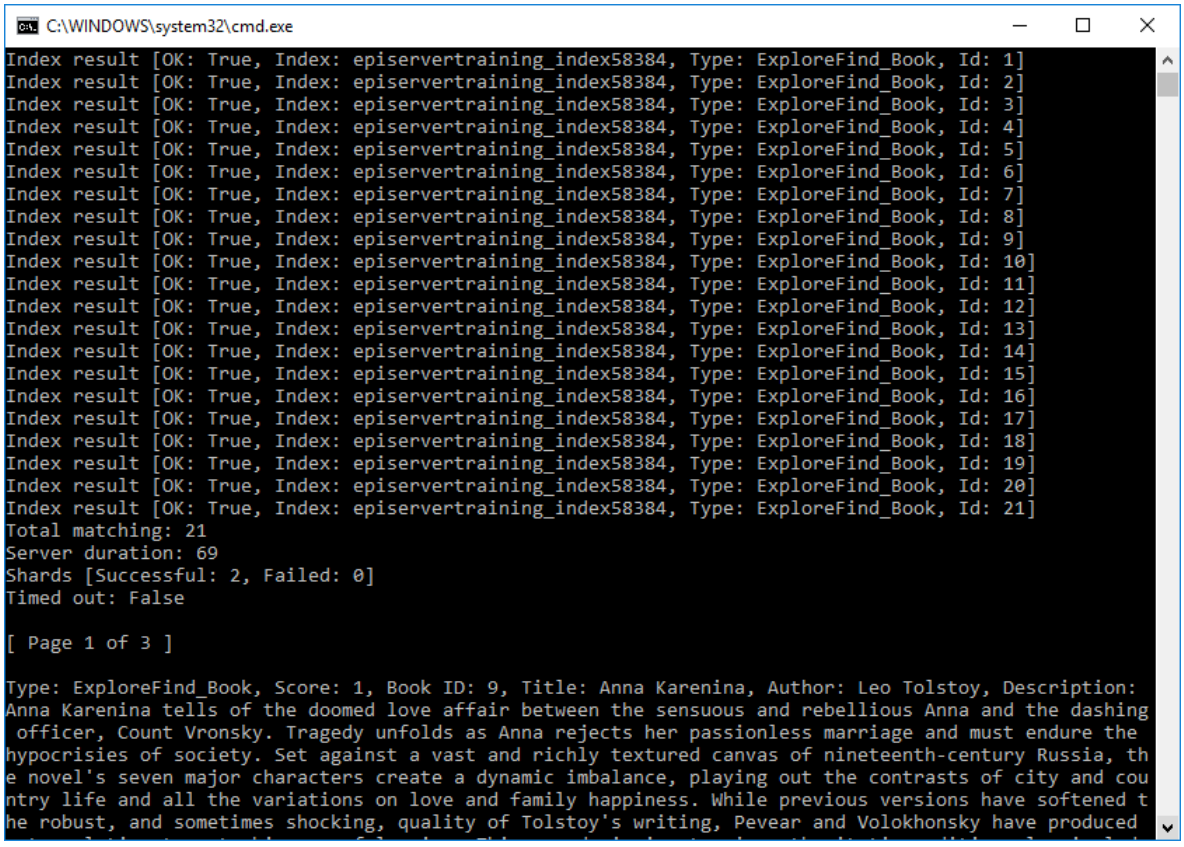
Both the person and the student are shown in the results because when you index an object, its inheritance hierarchy is indexed with it, so the student knows that it is also a person. Therefore, if you ever want to return all objects in the index, simply create a query for System.Object or the C# **object** keyword!

Querying for books

1. Add the following statements to the end of the **Main** method:

```
foreach (Book item in BookRepository.Books)
{
    client.Index(item, command =>
        { command.Refresh = true; }).Output();
}

ITypeSearch<Book> books = client.Search<Book>();
books.Output();
```

2. Start the console app and review the output, as shown in the following screenshot:



Each of the 21 books is indexed, and then a search of all **Book** instances is executed, shown in pages of 10 books each.

3. Comment out the foreach statement that indexes the books.
4. Add the following statements to the end of the **Main** method:

```
client.SearchBooksFor("lord");
client.SearchBooksFor("lord of the rings");
client.SearchBooksFor("\"lord of the rings\"");
client.SearchBooksFor("stories");
client.SearchBooksFor("politics");
client.SearchBooksFor("time and space");
```

5. Start the console app, and review the output.
6. Use this console app and its extension methods as a starting point to explore Episerver Find features, including some of the other queries and filters that you can apply.

Module H – Episerver Social

Goal

The overall goal of the exercises in this module is to learn how to program *Episerver Social*.

You will:

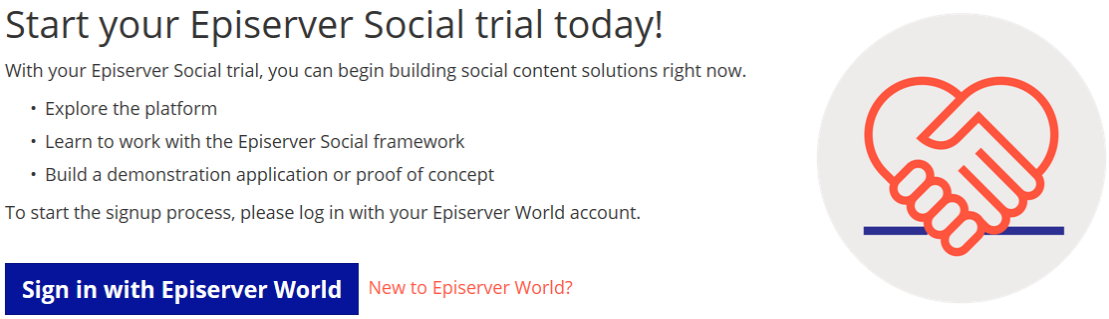
- Explore the SocialAlloy reference site.

Exercise H1 – Exploring the SocialAlloy reference site

In this exercise, you will set up the SocialAlloy project and explore how it has been implemented.

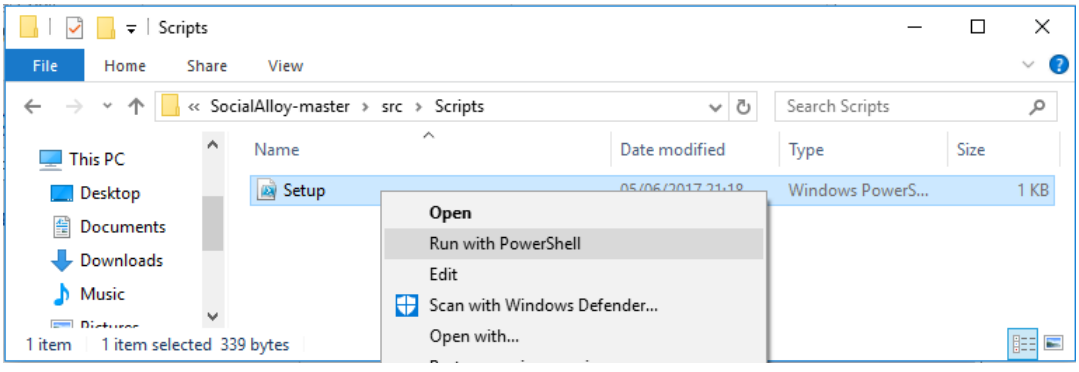
Signing up for a free Episerver Social trial account

1. Navigate to: http://demo.social.episerver.net/
2. Click **Sign in with Episerver World**, as shown in the following screenshot:



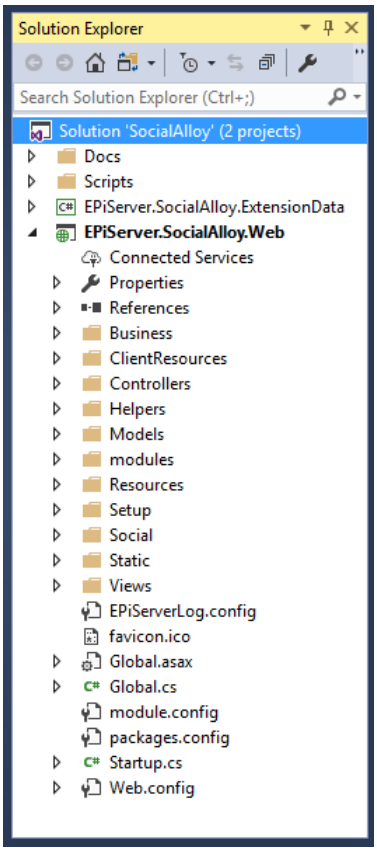
Setting up SocialAlloy

1. Right-click **cmsadvdev_exercises_socialalloy.zip**, and click **Properties**.
2. Click **Unblock**, and then click **OK**.
3. Extract the folders and files in **cmsadvdev_exercises_socialalloy.zip** to drive **C:**
4. In File Explorer, open the **C:\SocialAlloy-master\src\Scripts** folder, right-click **Setup.ps1**, and click **Run with PowerShell**, as shown in the following screenshot:

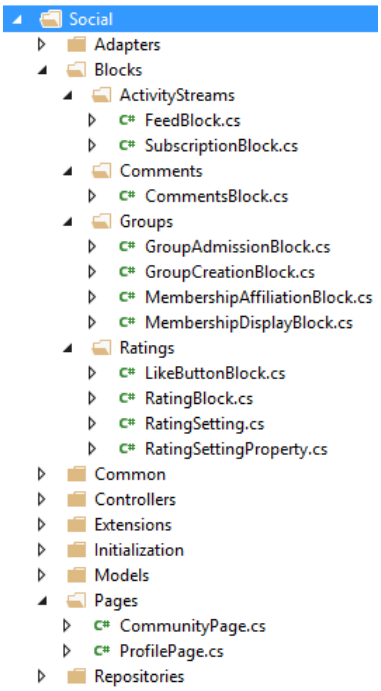


5. In Visual Studio, open **C:\SocialAlloy-master\src\SocialAlloy.sln**.

6. In **Solution Explorer**, note the **EPiServer.SocialAlloy.Web** project, as shown in the following screenshot:



7. Expand the **Social** folder, and note the page and block content types that have been defined, as shown in the following screenshot:



8. Expand **\Repositories\Comments**, and open **PageCommentRepository.cs**, and note the following:

   - The field for the comment service:

     ```
     private readonly ICommentService commentService;
     ```

- The statement adding a new comment using the comment service:

```
addedComment = this.commentService.Add(newComment);
```

- The multiple catch statements for Social exceptions, for example:

```
catch (SocialAuthenticationException ex)
```

- The statement to create a reference to the target content:

```
var parent = Reference.Create(filter.Target);
```

- The statement to get the comments for the target content using a filter that enables paging, filtering by the parent, and sorting:

```
comments = this.commentService.Get(
    new Criteria<CommentFilter>
    {
        PageInfo = new PageInfo
        {
            PageSize = filter.PageSize
        },
        Filter = new CommentFilter
        {
            Parent = parent
        },
        OrderBy = { new SortInfo(CommentSortFields.Created, false) }
    }
).Results.ToList();
```

Configuring Episerver Social

1. Open **Web.config**.
2. Add an entry to the **configSections** element, to allow Episerver Social to be configured, as shown in the following markup:

```
<configuration>
  <configSections>
    <section name="episerver.social"
             type="EPiServer.Social.Common.Rest.Configuration.SocialConfiguration,
                   EPiServer.Social.Common.Rest.Configuration"/>
```

3. Add the **episerver.social** element, to configure your Episerver Social account, as shown in the following markup:

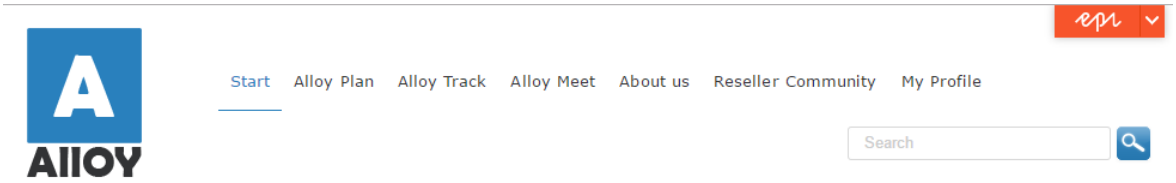
```
<episerver.social>
  <settings timeout="100000"/>
  <authentication appId="{replace with your account info}"
                  secret="{replace with your account info}"/>
  <endpoints>
    <add name="Comments" value="https:{replace with your account info}/" />
    <add name="Ratings" value="https:{replace with your account info}/" />
    <add name="Moderation" value="https:{replace with your account info}/" />
    <add name="ActivityStreams" value="https:{replace with your account info}/" />
    <add name="Groups" value="https:{replace with your account info}/" />
  </endpoints>
</episerver.social>
```

Exploring comments, ratings, and activity feeds

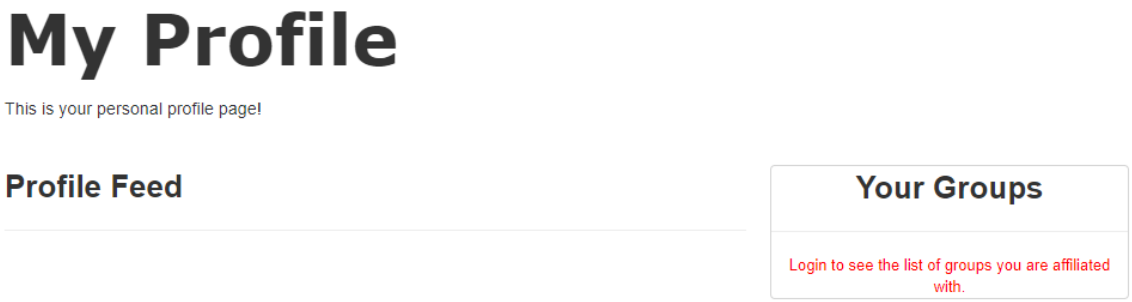
1. Start the **EPiServer.SocialAlloy.Web** website.
2. Register an administrator account:

a. Username: **Admin**
b. Email: **admin@alloy.com**
c. Password: **Pa$$w0rd**

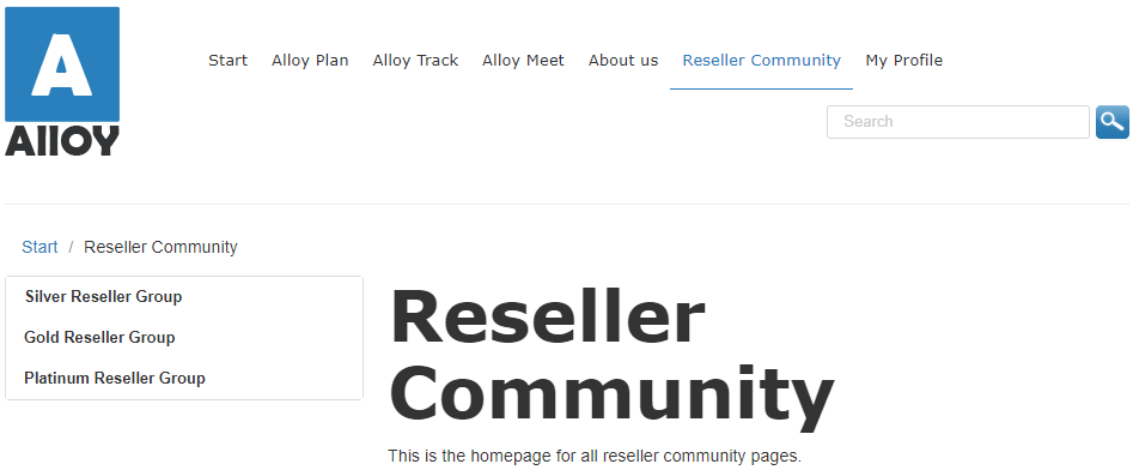
The **SocialAlloy** reference site is almost identicial to the **Alloy (MVC)** reference site, but it has two extra pages named **Reseller Community** and **My Profile**, as shown in the following screenshot:



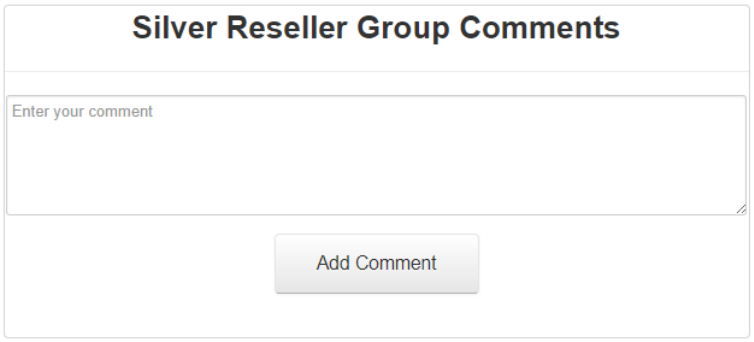
3. In the footer, click **Log out**, to become an anonymous visitor.
4. Click **My Profile**, and note the message to log in to see groups, as shown in the following screenshot:



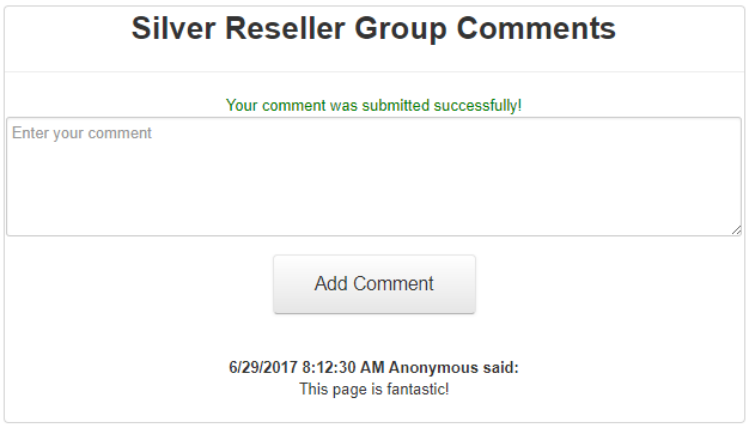
5. Click **Reseller Community**, and note there are three levels of reseller group, Silver, Gold, and Platinum, as shown in the following screenshot:



6. Click **Silver Reseller Group**, and note the page has a block for anonymous visitors to add a comment, as shown in the following screenshot:



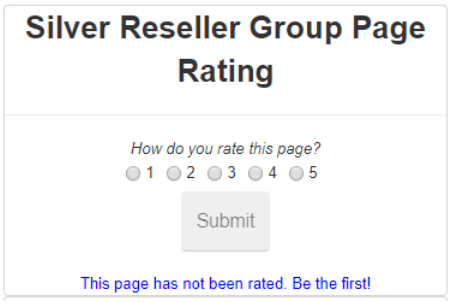
7. Enter a comment, for example, **This page is fantastic!**, click **Add Comment**, and note it is successfully submitted, as shown in the following screenshot:



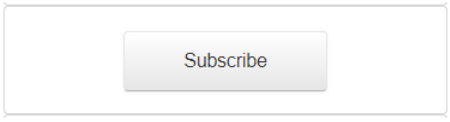
8. In the footer, click **Log in**, and log in as **Admin**.
9. Note that when a visitor is logged in, they have more blocks available to engage with the page:

The choice of what features require a user to be logged in (or not) is up to you.

a. Submitting a rating out of five for the page:



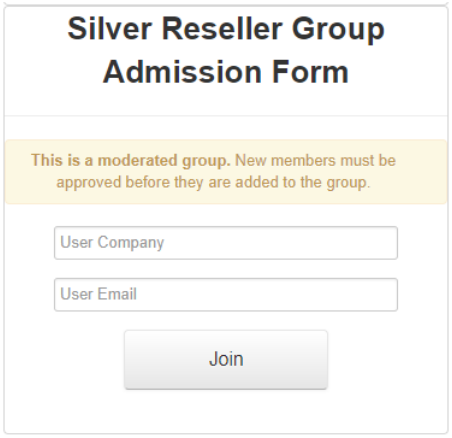
b. A button to subscribe to activity related to the page:



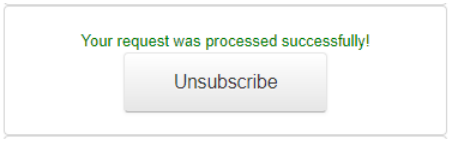
c. A list of members of the group for the page:

The previous screenshot shows a page with some existing members. Your list will be empty.

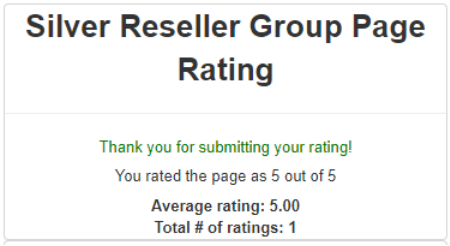
d. An admission request form to join the group for a logged in user:



10. Click **Subscribe**, and note Admin has now been subscribed to a stream of activities that happen to the Silver Reseller Group page, as shown in the following screenshot:



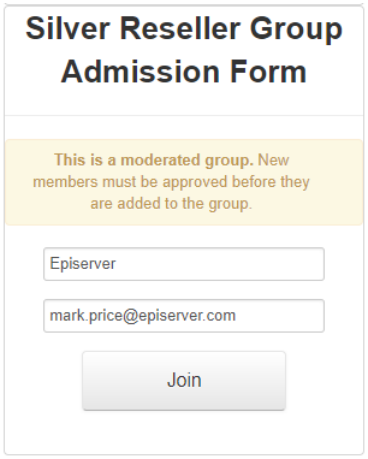
11. Enter a comment, for example, **Has anyone seen Guardians of the Galaxy Volume 2**?, and click **Add Comment**.
12. Submit a rating for the page, and note the rating blocks shows Admin's rating and some statistics about all ratings, as shown in the following screenshot:



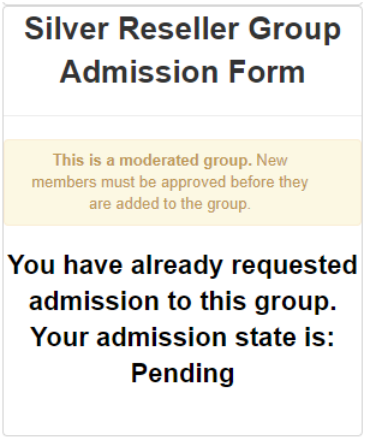
13. Click **My Profile**, and note **Your Activity Feed** has two entries, and you do not belong to any groups, as shown in the following screenshot:

Exploring groups, membership, and moderation workflows

1. Click **Reseller Community**.
2. Click **Silver Reseller Group**.
3. Enter your company and email in the **Admission Form**, as shown in the following screenshot:



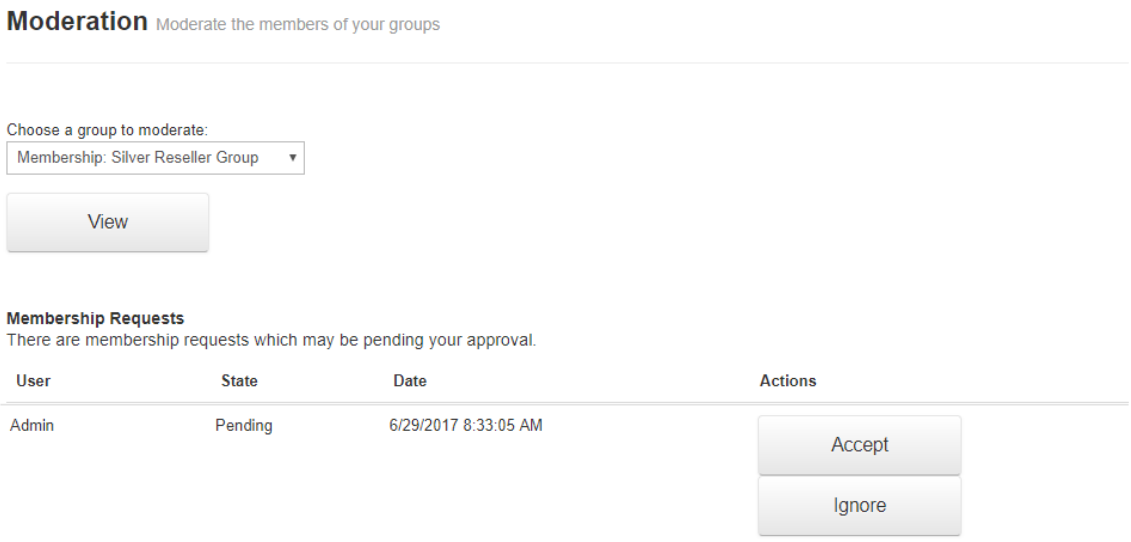
4. Click **Join**, and note that your admission is **Pending**, as shown in the following screenshot:



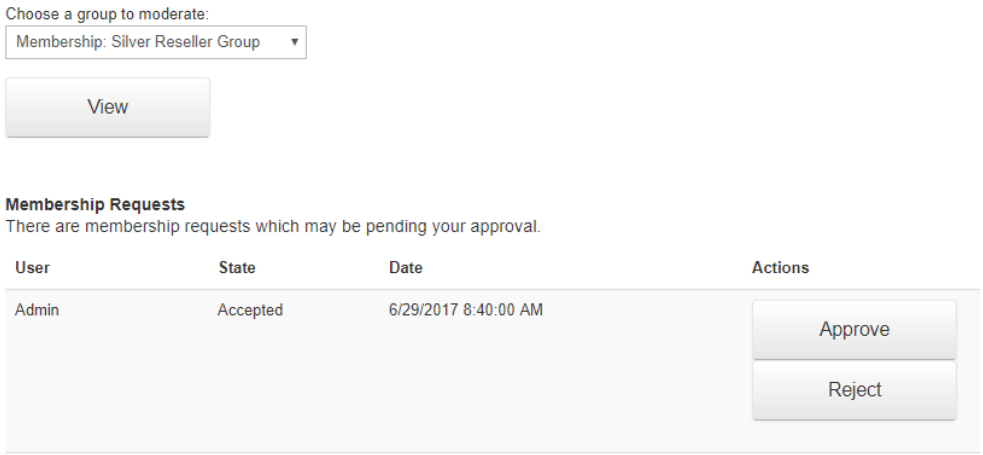
5. In your browser address bar, enter **moderation**, as shown in the following screenshot:

In a real site, the moderation page would be secure and accessible as an Admin plug-in.

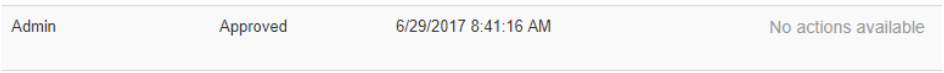
6. Choose **Membership: Silver Reseller Group** to moderate, click **View**, and note that there is a membership request that in the first step of a moderation workflow, must either be accepted or ignored, as shown in the following screenshot:



7. Click **Accept**, and note that the request has moved onto the second step of the moderation workflow, where it must be either approved or rejected:

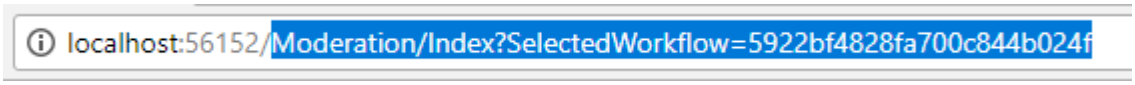


8. Click **Approve**, and note that there are no more steps, as shown in the following screenshot:

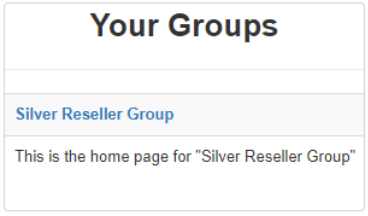


This is just an example of a two-step moderation workflow. Your moderation processes can be simpler or more complex.

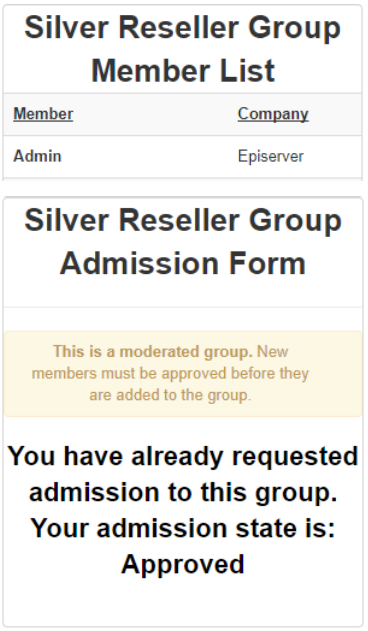
9. Delete the **/Moderation/** path from the address box to return to SocialAlloy site, as shown in the following screenshot:



10. Click **My Profile**, and note **Admin** is now a member of **Silver Reseller Group**, as shown in the following screenshot:



11. Navigate to **Reseller Community** | **Silver Reseller Group**, and note **Admin** is now an **Approved** member of **Silver Reseller Group**, as shown in the following screenshot:



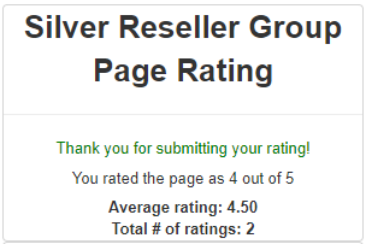
Exploring rating statistics

1. Navigate to **CMS** | **Admin** | **Admin** | **Administer Groups**.
2. Add a group named **Raters**.
3. Create a user named **Marie**, with password of **Pa$$w0rd**, email of **marie@alloy.com**, make her active, and a member of **Raters**.
4. Close the browser.
5. Open **Web.config**.
6. Press *Ctrl* + *F*, and find the attribute `path="EPiServer"`
7. Add **Raters** to the <authorization><allow> element for this path, as shown in the following markup:

```
<authorization>
  <allow roles="WebEditors, WebAdmins, Administrators, Raters" />
```

This will allow Marie to log in without being an editor or administrator of the Episerver site.

8. Start the **SocialAlloy** site, and log in as **Marie**.
9. Navigate to **Reseller Community** | **Silver Reseller Group**.
10. Give the page a different rating to before, and note the average and total ratings, as shown in the following screenshot:



Remember, Episerver Social is a collection of micro-service building blocks. The SocialAlloy reference site shows some ways to combine those micro-services into Episerver Blocks, but you can implement the functionality in any way you choose! 😊

This is the end of the exercises book.