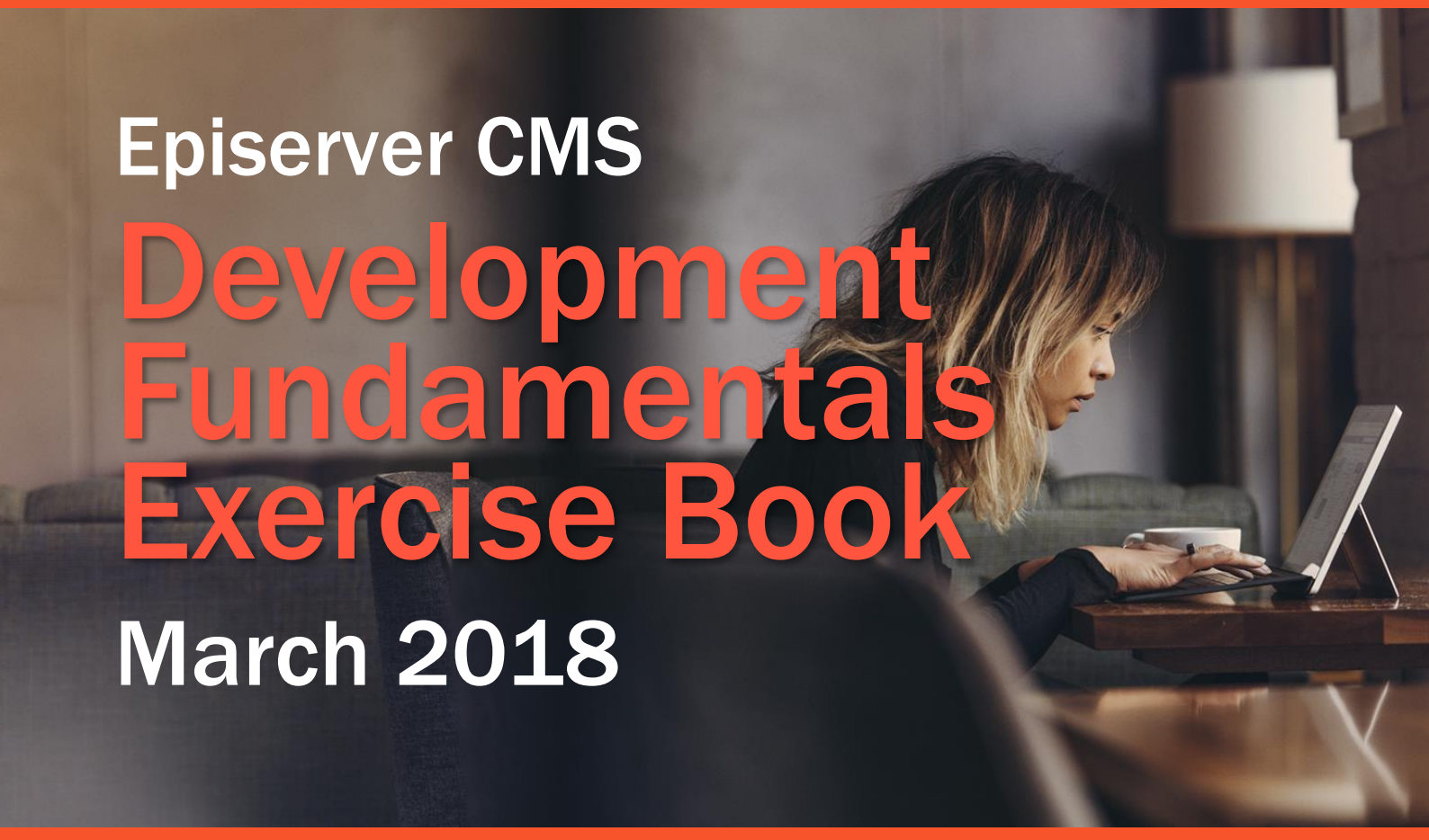


Episerver CMS

Development Fundamentals Exercise Book

March 2018

Product version: Update 204
Course version: 18.03

Course title: *Episerver CMS – Development Fundamentals* Course code: 170-3020
Course version: 18.03, 5th March 2018 Product Update 204, 5th March 2018

Episerver CMS Visual Studio Extension version: 11.1.0.326
Episerver CMS packages: EPiServer.CMS.Core 11.4.0, EPiServer.CMS.UI 11.3.1

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

Episerver - update 204

Included packages: CMS Core 11.4.0, CMS UI 11.3.1, Commerce 11.8.2, Marketing Automation Forms 1.3.1 and 1.4.1, Campaign 6.75

Mar 05 2018

New release of Episerver CMS Core and Episerver Campaign. Bug fixes for Episerver CMS UI, Episerver Commerce, and Marketing Automation Forms.

Table of Contents

Episerver CMS - Development Fundamentals

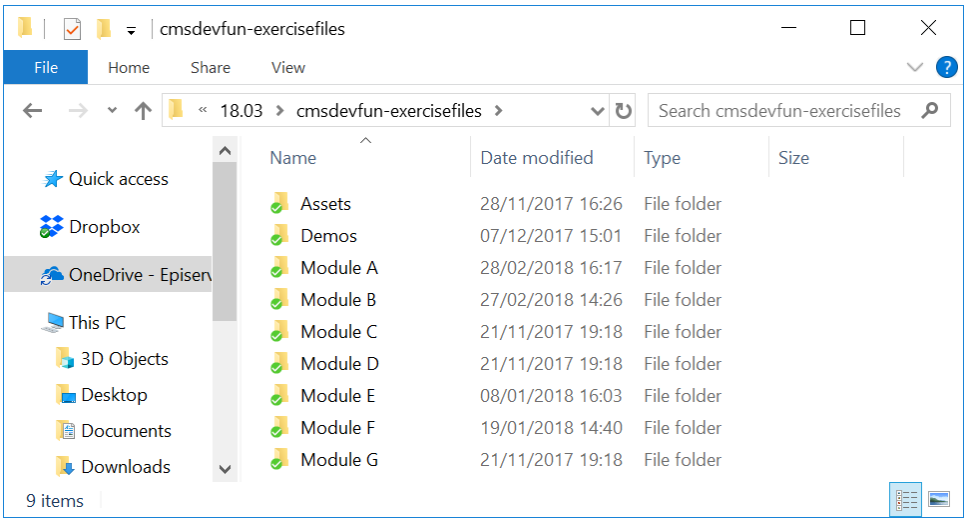
Student prerequisites

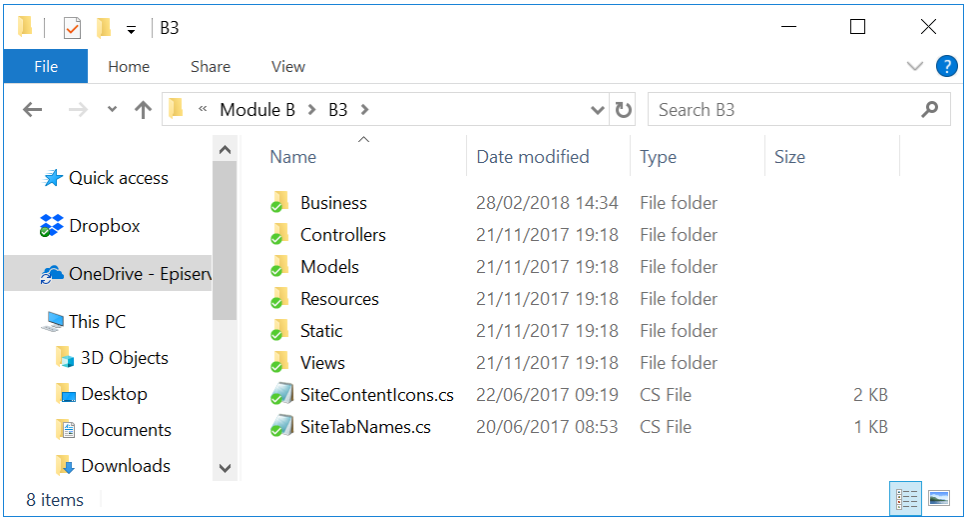
You should have previous experience with Microsoft Visual Studio and ASP.NET MVC development.

Software requirements

To complete these exercises, you will need:

- **Microsoft Visual Studio 2015 or 2017** with latest updates.
- **Episerver CMS Visual Studio Extension** version 11.1.0.326 or later.
- **cmsdevfun_exercisefiles.zip** containing starter, solution, and support files for the exercises, as shown in the following screenshots:





Module A – Getting Started with Episerver CMS

Goal

The overall goal of the exercises in this module is to learn how to use the core features of Episerver CMS for editors and admins, and how to set up new Episerver websites. You will:

1. Set up an Alloy (MVC) sample website, update it to the latest version of Episerver CMS, and install some useful add-ons.
2. Review authentication and authorization good practice, create common groups and users, and set access rights.
3. Practice creating, editing, and publishing content.
4. Define content approval sequences, approve content, and perform A/B testing.
5. Localize content for English, Swedish, and Danish languages, localize choices in the styles drop-down menu in the TinyMCE editor, and localize names and descriptions of content types.
6. Implement functionality to reset the administrator account in case of a forgotten password.

Exercise A1 – Setting up the AlloyDemo website

In this exercise, you will set up an Alloy (MVC) site, update it to use the latest version of Episerver CMS, and you will install some add-ons that extend Episerver with extra features:

- Episerver Forms: http://webhelp.episerver.com/latest/addons/episerver-forms/episerver-forms.htm
- Episerver A/B Testing: http://webhelp.episerver.com/latest/cms-edit/ab-testing.htm

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.

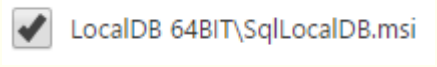
1. Download and install **Microsoft Visual Studio Community 2017** from the following link:

   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**.

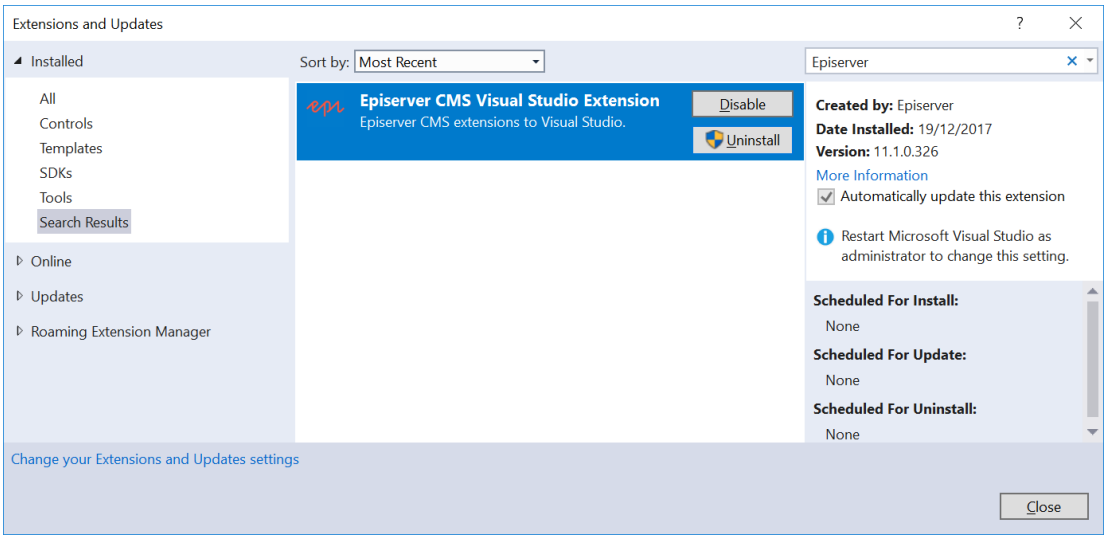
Microsoft SQL Server Express LocalDb should be included with Visual Studio, but you can also install it separately from the following link: https://www.microsoft.com/en-gb/download/details.aspx?id=42299
Click **Download**, check the box for **LocalDB 64BIT\SqlLocalDB.msi**, and click **Next**.



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. We recommend that you automatically update this extension. 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 it to complete installing, and then click **Close**.

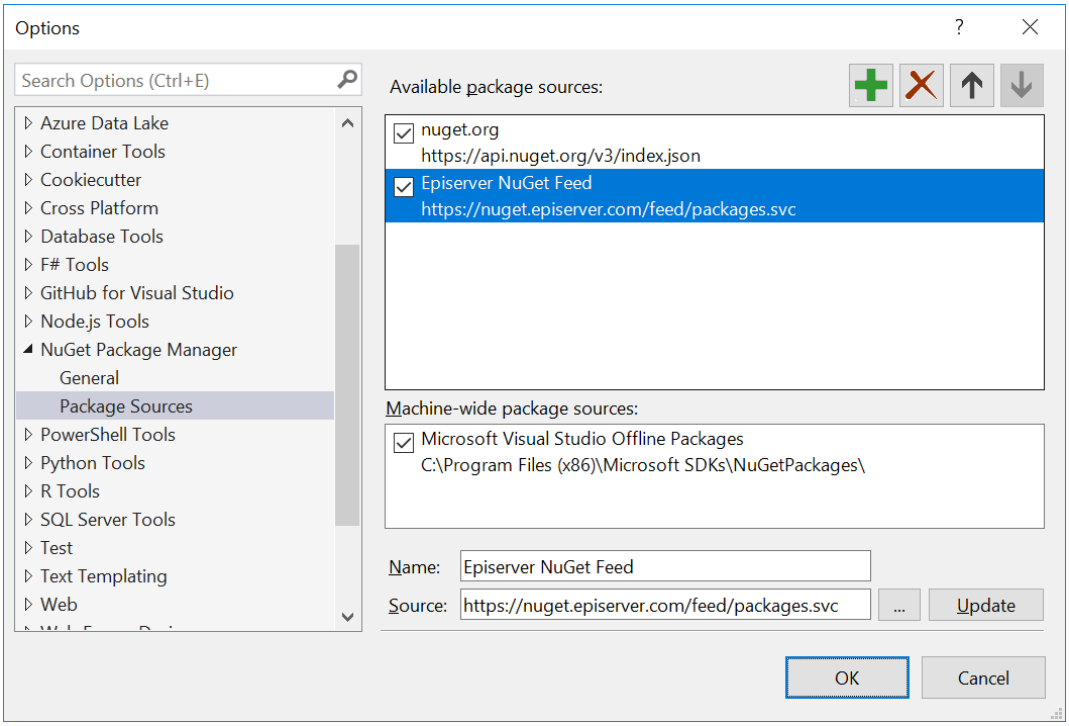
Configuring the Episerver NuGets package source

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/`



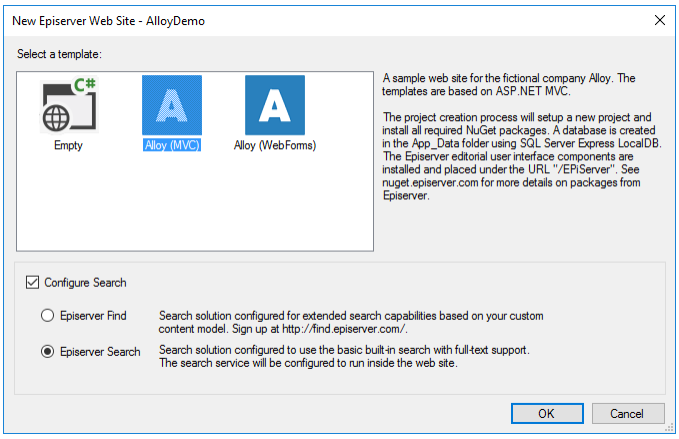
Creating an Alloy (MVC) Episerver website with sample content

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

For compatibility with Episerver CMS 11 you must choose a minimum target of .NET Framework 4.6.1 because it requires .NET Standard 2.0.

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

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

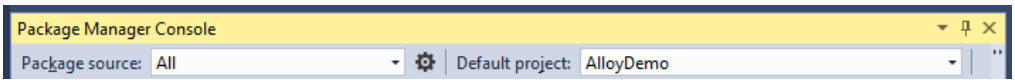


6. Click **OK** and wait for the project to be created.

Updating the Episerver NuGet packages

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, as shown in the following screenshot:
   a) Package source: **All**
   b) Default project: **AlloyDemo**



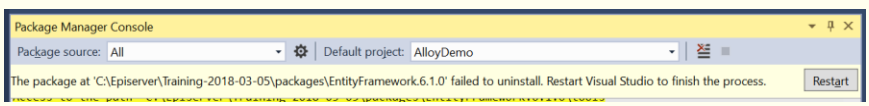
If you cannot see **All** as a **Package source**, then close Visual Studio, use **File Explorer** to open **%appdata%\NuGet**, rename **NuGet.config** to **NuGet.backup**, and restart Visual Studio. That should recreate **NuGet.config**, including the **All** option. You can redefine the **Episerver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

3. Enter the following command to update all packages referenced by the project **AlloyDemo** using restrictions defined in packages.config:

```
Update-Package -ProjectName AlloyDemo -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

You might need to restart Visual Studio a couple of times to update Entity Framework, as shown in the following screenshot:



Updating the Episerver database

1. Start the site by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.

2. If you see an exception message about the **EPiServerDB** database, as shown in the following screenshot, then changes need to be made to the database schema:

   Server Error in '/' Application.

   The database schema for 'CMS' has not been updated to version '7052.0', current database version is '7050.0'. Update the database manually by running the cmdlet 'Update-EPiDatabase' in the package manager console or set updateDatabaseSchema="true" on episerver.framework configuration element.

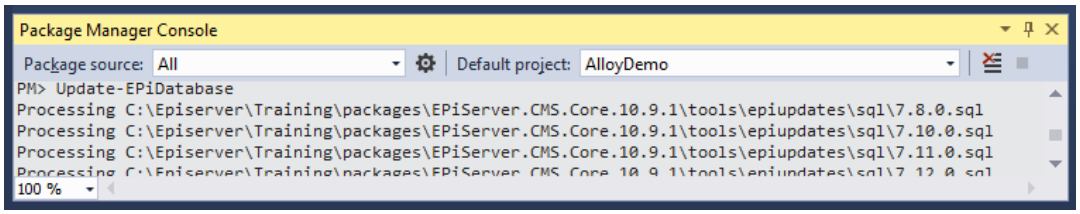
3. Close the browser.

You can check if an update to a NuGet package would require an update to the database schema at the following link: https://nuget.episerver.com/en/Comparedatabase/

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

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

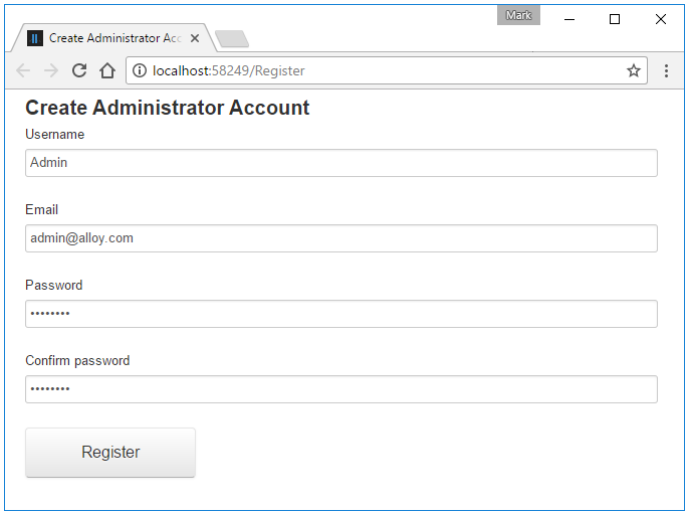
   ```
   Update-EPiDatabase
   ```



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

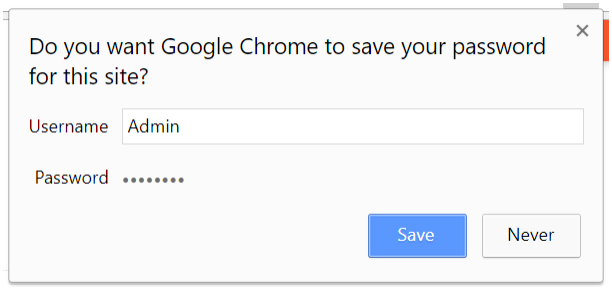
Testing the Episerver website and registering an administrator account

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:

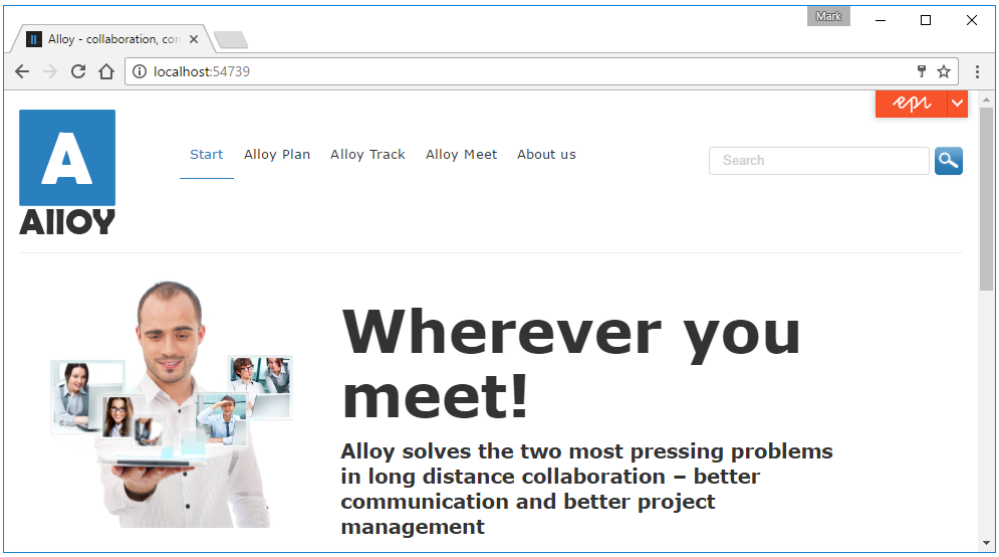


- Username: **Admin** or something else but make a note of it!
- Email: **admin@alloy.com** or some other e-mail address (it does not need to be real).
- Password: **Pa$$w0rd** or something else but make a note of it!

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



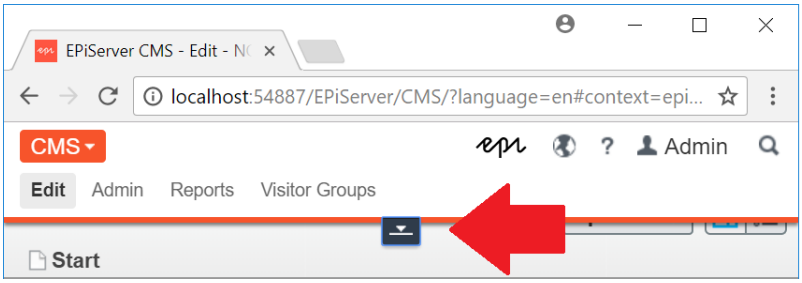
4. You should now see the **Alloy** sample site's **Start** page, as shown in the following screenshot:



5. Click the **epi** menu in the top-right corner, as shown in the following screenshot:

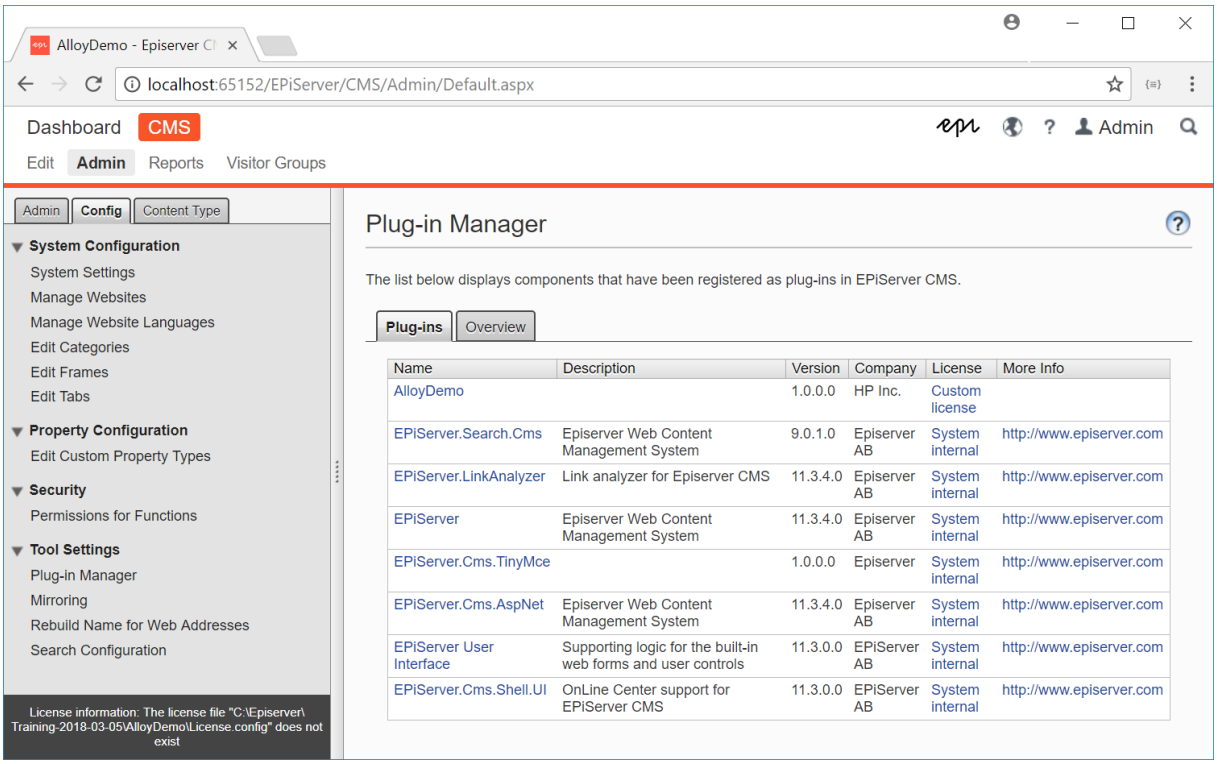
If you can't see the **epi** quick access menu, scroll down to the bottom of the page, and in the footer, click **Log In**, and log in as Admin.

6. At the top of the page, pull down the orange **Global** menu, and optionally click the pull-down menu a second time if you would like to pin it, as shown in the following screenshot:



The Global menu pinning functionality used to be distributed as an add-on named **MenuPin**. It is now part of the core product with Episerver CMS UI 11.2 or later. You won't see it if you use the packages that are distributed with Episerver CMS Visual Studio Extension 11.1.0.326. To use the pinning feature you must upgrade to the latest packages as you have now done.

7. Navigate to **CMS** | **Admin** | **Config** | **Plug-in Manager** to confirm the version of Episerver CMS you are using, as shown in the following screenshot:
    - **EPiServer** and **EPiServer.Cms.AspNet** version 11.3.4: these are the CMS Core APIs.
    - **EPiServer User Interface** and **EPiServer.Cms.Shell.UI** version 11.3.0: these are the CMS user interface including Edit and Admin views.



8. Close the browser.

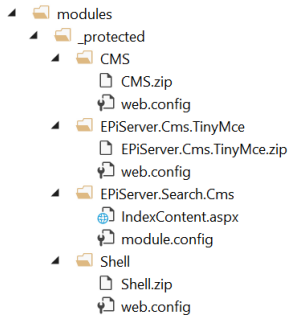
Windows 7 users

If you are using Windows 7, or your operating system has web sockets disabled, then in Edit view you will see warning messages about no support for real-time communication. To disable these warning messages, add the following element to Web.config:

```
<appSettings>
  <add key="Epi.WebSockets.Enabled" value="false"/>
```

Installing some useful add-ons

1. Open the **AlloyDemo** project and view **Solution Explorer**.
2. Expand **~\modules_protected** folder, as shown in the screenshot:



Episerver products and add-ons are deployed as *shell modules*. In a new Alloy (MVC) project, you will have four modules already installed: **CMS** is Episerver CMS user interface, **EPiServer.Cms.TinyMce** is the default rich text editor, **EPiServer.Search.Cms** is the (optional) integration with Episerver Search, and **Shell** is the shared UI including **Dashboard** and **Global** menu. The ZIP files contain the built-in functionality of the CMS, like Admin and Edit view. You can open **CMS.zip** and look at the contents, but don't change anything!

3. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.

4. Enter the following command to install **Episerver Forms**:

> In **Package Manager Console**, you can press the up arrow ↑ to repeat a previous command. You can then edit the command before pressing *ENTER*. Package Manager Console NuGet commands and package names are not case-sensitive.

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

5. Enter the following command to install **A/B testing**:

```
Install-Package -ProjectName AlloyDemo EPiServer.Marketing.Testing
```

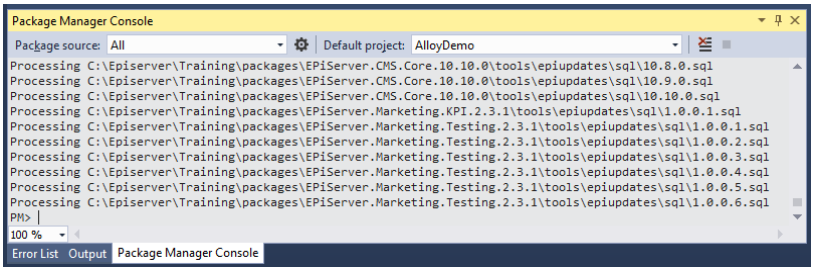
> After installing add-ons, you may have to update some packages and the database schema. A/B Testing, the **EPiServer.Marketing.Testing** package, is an example of an add-on that requires this.

6. Enter the following command to update dependent packages:

```
Update-Package -ProjectName AlloyDemo -ToHighestMinor
```

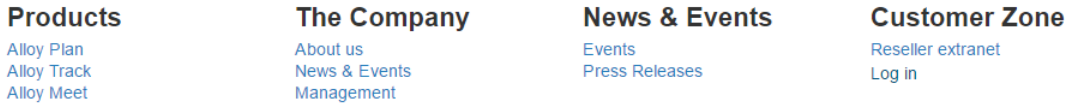
7. Enter the following command to update the database schema, as shown in the following screenshot:

```
Update-EPiDatabase
```

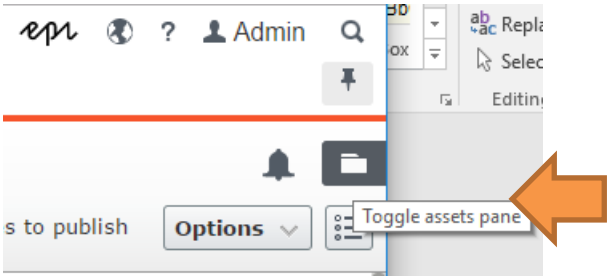


Exploring Forms and A/B Testing

1. Start the website by navigating to **Debug** | **Start Without Debugging** or pressing *Ctrl* + *F5*.
2. Scroll down the Start page, and in the **Customer Zone**, click **Log in**, as shown in the following screenshot:

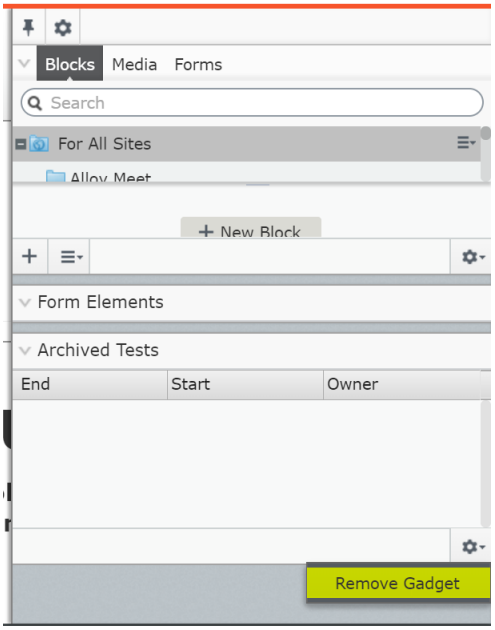


3. Log in as **Admin** and click the **epi** menu to switch to Edit view for the current page.
4. In Edit view, click **Toggle assets pane**, as shown in the following screenshot:



5. In the **Assets** pane, note the **Forms** tab, next to **Blocks** and **Media**, and the **Form Elements** and **Achived Tests** gadgets.

6. At the bottom of the **Assets** pane, in **Archived Tests**, click the **Settings** button, and select **Remove Gadget**, as shown in the following screenshot:



Some add-ons, like A/B Testing, automatically add gadgets, like **Archived Tests** that take up valuable space. It is safe to remove them, because they can easily be added back later if the logged in user needs them. Gadgets are specific to the logged in user, so you won't affect anyone else's gadgets.

7. Close the browser.

Congratulations! You have now created an Episerver CMS website, updated it to the latest version, and installed some useful add-ons.

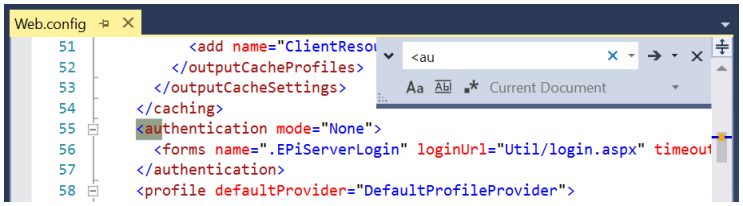
Exercise A2 – Reviewing and creating groups and users

In this exercise, you will create users and groups with different access rights on the website.

Prerequisites: complete Exercise A1.

Reviewing authentication and authorization in an Alloy (MVC) site

1. In the **AlloyDemo** project, open **~/Web.config**.
2. Navigate to **Edit** | **Find and Replace** | **Quick Find**, or press *Ctrl* + *F*, and enter **<au** to find the `<authentication>` element, as shown in the following screenshot:



3. Note the **mode** and **loginUrl** attributes.

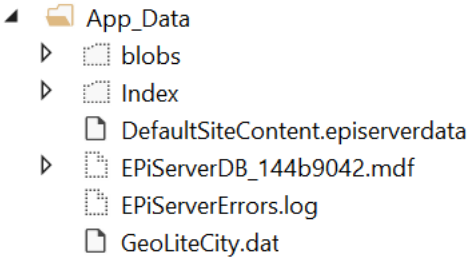
Alloy (MVC) project template sets authentication mode to none in the configuration file. This does not mean the Alloy site does not authenticate. For historical reasons, you must set authentication mode to **None** to enable federated authentication systems like **ASP.NET Identity**.

4. Find the `<membership>` and `<roleManager>` elements, and note they are cleared because Alloy (MVC) does not use ASP.NET Membership.
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
```

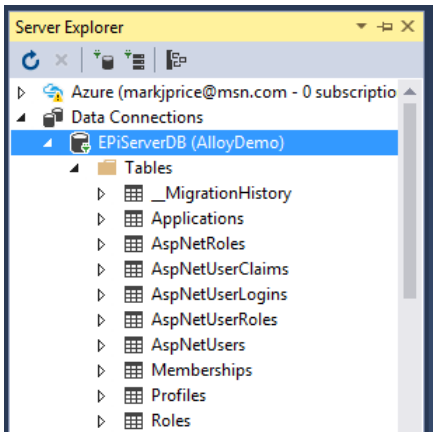
Alloy (MVC) project template uses **ASP.NET Identity** with accounts stored in Episerver CMS database to authenticate users and authorize roles, and uses cookies for re-authentication. It has a custom extension method named **UseAdministratorRegistrationPage** that is only displayed to the visitor if, (1) the browser is executing on the web server i.e. it is a local request, and (2) there are no users in the CMS database.

6. In **Solution Explorer**, click **Show All Files**.
7. Expand the **~\App_Data** folder, and note the **blobs** and **Index** folders, and the SQL database file, as shown in the following screenshot:

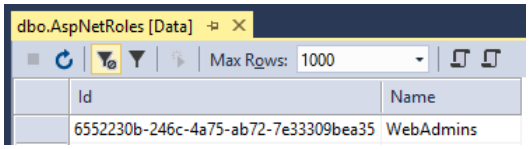


8. Double-click **EPiServerDB_{GUID}.mdf** to open a database connection to it.

9. In **Server Explorer**, expand **Tables**, and note the tables that begin with **AspNet**, as shown in the following screenshot:



10. Right-click **AspNetRoles**, and click **Show Table Data**, and note the **WebAdmins** role has been created for you, as shown in the following screenshot:



11. Right-click **AspNetUsers**, and click **Show Table Data**, and note the **Admin** user (or whatever you entered on the register page) has been created for you, and that it has columns for the following to implement best practice for security: **IsApproved**, **LastLoginDate**, **Email**, **PasswordHash** (the original password is not stored), **AccessFailedCount**, **UserName**.
12. Close both tables.
13. In **Server Explorer**, right-click **EPiServerDB (AlloyDemo)**, and click **Close Connection**.

Reviewing virtual roles and changing the default mapping for CmsEditors

You will now follow good practice by reviewing and modifying the configuration of the virtual roles that give access to the working areas of Episerver CMS. Most projects should not leave the default configuration as-is so you will make some changes to learn some common modifications that you might make.

1. In the **AlloyDemo** project, open **~\Web.config**.
2. 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" />
```

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.

3. Find the existing entry for **CmsAdmins**, and modify it so that a stored role named **AccessToAdminView** as well as **WebAdmins** and **Administrators** maps to **CmsAdmins**, as shown underlined in the following markup:

```
<add name="CmsAdmins"
     type="EPiServer.Security.MappedRole, EPiServer.Framework"
     roles="WebAdmins, Administrators, AccessToAdminView" mode="Any" />
```

You will create a stored role aka group named **AccessToAdminView** later in this exercise.

4. Find the existing entry for **CmsEditors**, and modify it so that a stored role named **AccessToEditView** maps to **CmsEditors** instead of **WebEditors**, as shown underlined in the following markup:

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

You will create a stored role aka group named **AccessToEditView** later in this exercise.

5. Add an entry at the bottom of the list of virtual role providers, that maps members of a stored role named **Personalizers** to the virtual role named **VisitorGroupAdmins**, as shown in the following markup:

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

You will create a stored role aka group named **Personalizers** later in this exercise. Any member of this group will have access to the **Visitor Groups** administration user interface.

6. Add an entry at the bottom of the list of virtual role providers, that maps members of a stored role named **Developers** to the virtual role named **EPiBetaUsers**, as shown in the following markup:

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

You will create a stored role aka group named **Developers** later in this exercise. Any member of this group will have access to any beta features of the user interface.

7. Find the `<location path="EPiServer">` element, and 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>
```

8. Replace the three <allow> roles with **CmsEditors** and **CmsAdmins**, as shown in the following markup:

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

9. Find the `<location path="EPiServer/CMS/admin">` element, and note the `<authorization>` element allows members of **WebAdmins** or **Adminstrators** to access the **EPiServer/CMS/admin** path, as shown in the following markup:

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

10. Replace the two <allow> roles with **CmsAdmins**, as shown in the following markup:

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

11. Find the `<location path="Views/Plugins">` element, and 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>
```

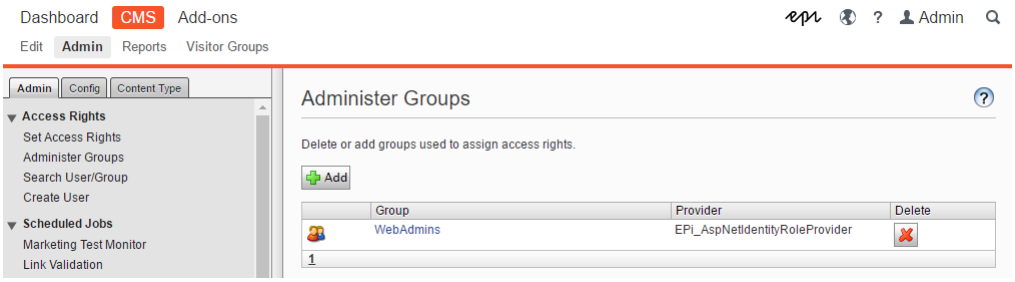
12. Replace the three entries with **CmsEditors** and **CmsAdmins**, as shown in the following markup:

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

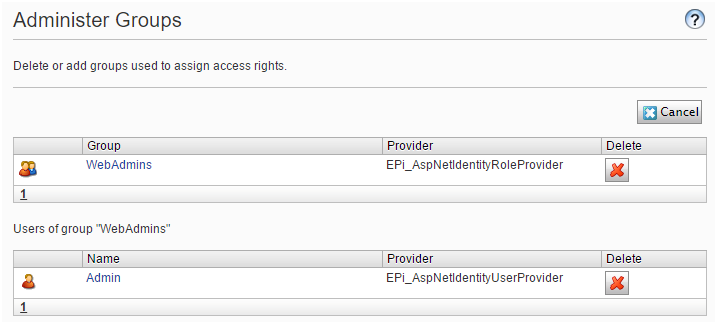
You have now removed any dependency on the existance of stored roles or groups named **WebAdmins**, **Administrators**, and **WebEditors**. Instead, any member of the stored role or group named **AccessToEditView** will have access to the Edit view, and any member of a stored role or group that maps to **CmsAdmins** will have access to Admin view.

Reviewing the current groups and users

1. Start the **AlloyDemo** website, and log in as **Admin**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Administer Groups**, as shown in the following screenshot:

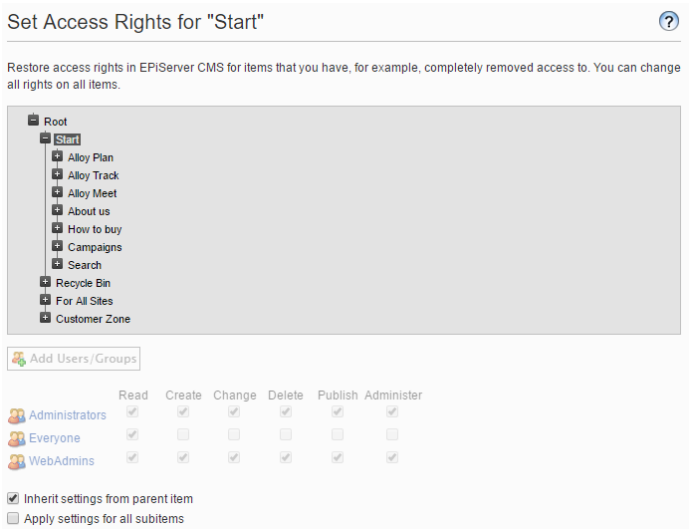


3. Click **WebAdmins**, to show the members of that group, as shown in the following screenshot:

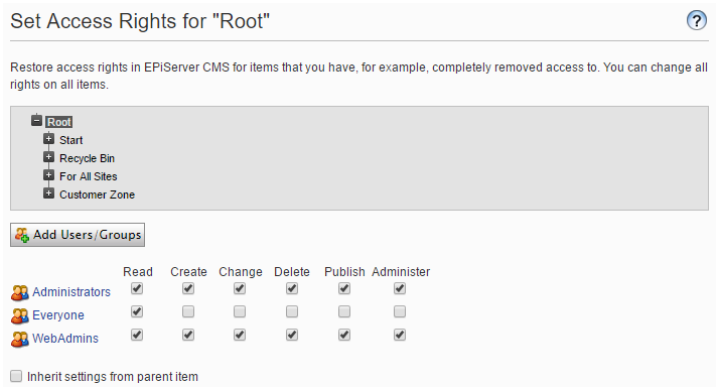


In a newly created Alloy (MVC) website, **WebAdmins** has a single member, that is the user that you registered when the site is first started.

4. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Set Access Rights**, and note that by default the **Start** page (and all other pages) inherit their access rights from their parent item, as shown in the following screenshot:



5. Click **Root**, and note that it has entries for **Administrators** (a Windows-stored group), **Everyone** (a virtual role) and **WebAdmins** (an SQL-stored group in the CMS database), as shown in the following screenshot:



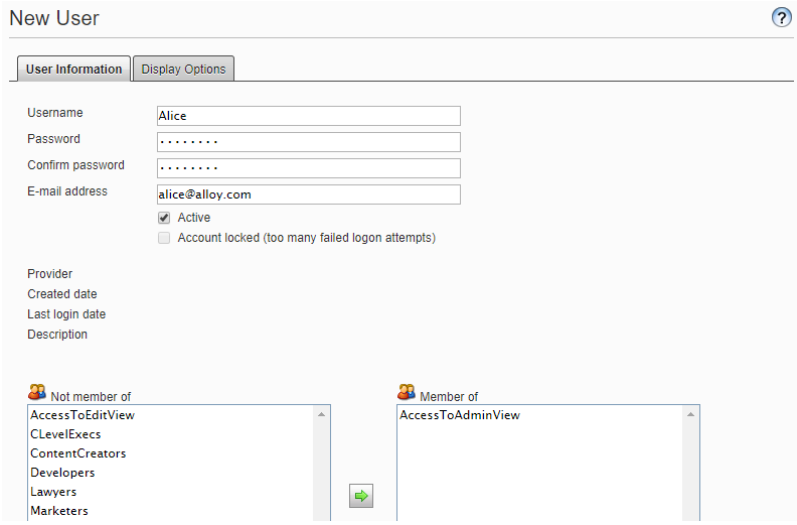
Everyone represents anonymous visitors, so by default they only have **Read** access rights to all content.

Creating some new groups and a user using Admin view

1. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Administer Groups**.
2. Click **+ Add**.
3. Enter the name **AccessToEditView** and click **Save**.

Remember that **AccessToEditView** is mapped to **CmsEditors** which gives access to the Edit view and Reports working areas.

4. Add a few more groups:
    a. **AccessToAdminView**: this is mapped to **CmsAdmins** which gives access to all working areas including Admin view.
    b. **ContentCreators**: members of this group will be able to create, change, and publish content throughout the site, except in the **News & Events** section.
    c. **NewsEditors**: members of this group will be the only editors who can create, change, and publish pages in the **News & Events** section.
    d. **Marketers**: members of this group will be the only editors who can create **Product** pages.
    e. **Developers**: members of this group can access beta features.
5. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Create User**, and fill in the following information, as shown in the following screenshot:
    a. Username: **Alice**
    b. Password: **Pa$$w0rd**
    c. E-mail address: **alice@alloy.com**
    d. Active: selected
    e. Member of: **AccessToAdminView**

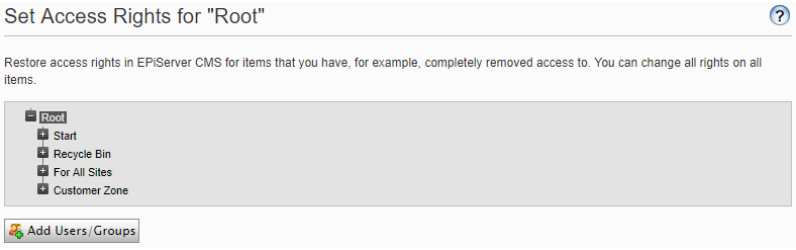


6. Click **Save**.

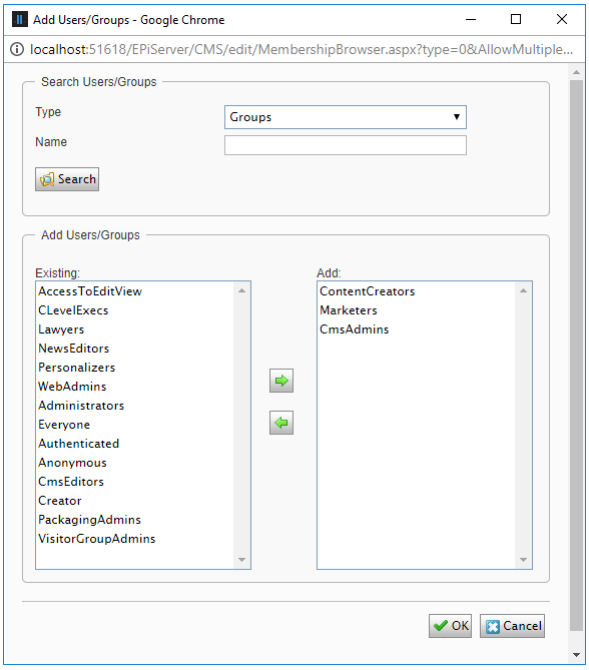
Assigning access rights using Admin view

1. Navigate to **CMS** | **Admin** | **Admin** | **Access Rights** | **Set Access Rights**.
2. In the content tree, select **Root**.

3. Click **Add Users/Groups**, as shown in the following screenshot:

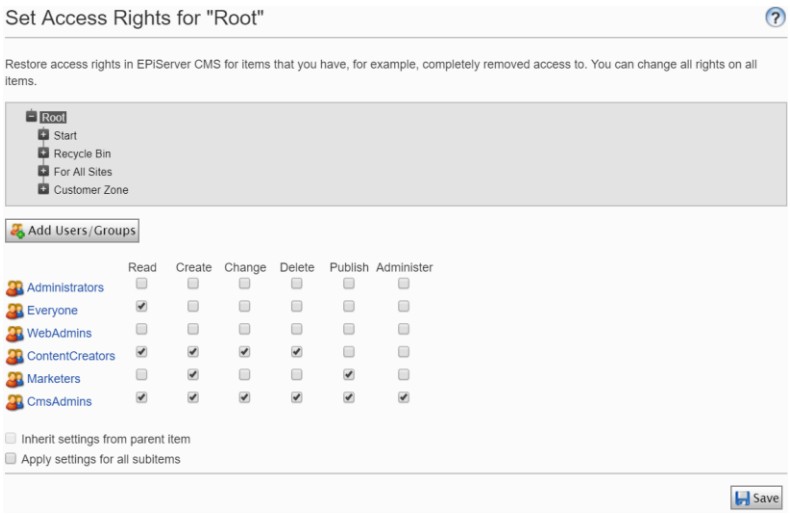


4. Search for **Groups**, add **ContentCreators**, **Marketers**, and **CmsAdmins**, and click **OK**, as shown in the following screenshot:



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

5. 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. **CmsAdmins**: set all access rights.

d. **ContentCreators**: set Read, Create, Change, Delete access rights.

e. **Marketers**: set Create and Publish access rights.

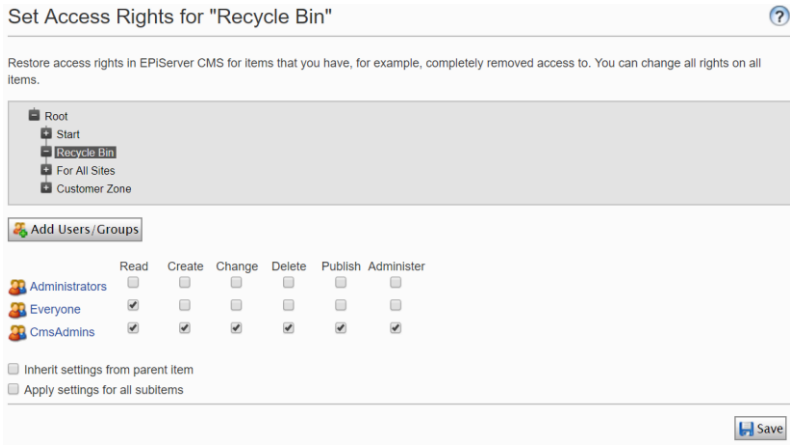
Clearing all access rights will remove the access control entry from the list when you click **Save**.

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**.

6. Click **Save**.
7. In the **Set Access Rights** content tree, select **Recycle Bin**.

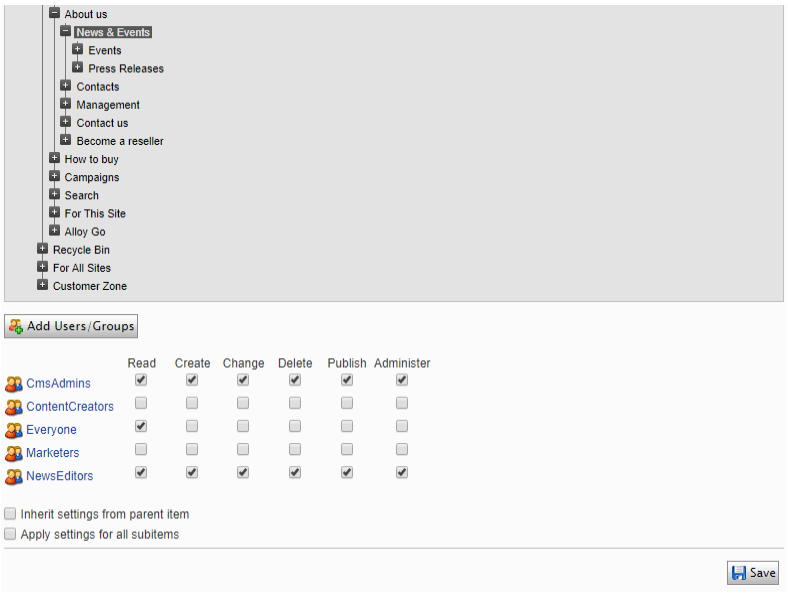
By default, only members of the Windows group Administrators can work with the Recycle Bin!

8. Click **Add Users/Groups**.
9. Search for **Groups**, add the **CmsAdmins** group, and click **OK**.
10. Clear all access rights from **Administrators**, and assign all access rights to **CmsAdmins**, as shown in the following screenshot:



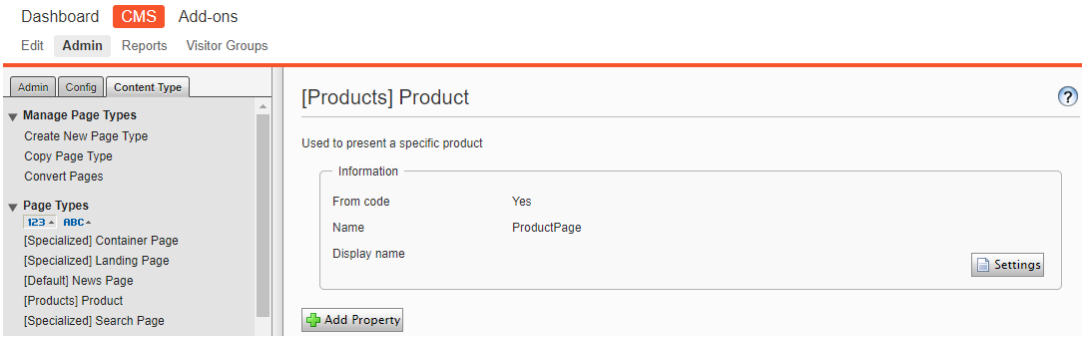
11. Click **Save**.
12. In the **Set Access Rights** content tree, expand **Start**, expand **About us**, and select **News & Events**.
13. Clear the **Inherit settings from parent item** check box.
14. Click **Add Users/Groups**.
15. Search for **Groups**, add the **NewsEditors** group, and click **OK**.

16. Clear all access rights from **ContentCreators** and **Marketers**, and assign all access rights to **NewsEditors**, as shown in the following screenshot:



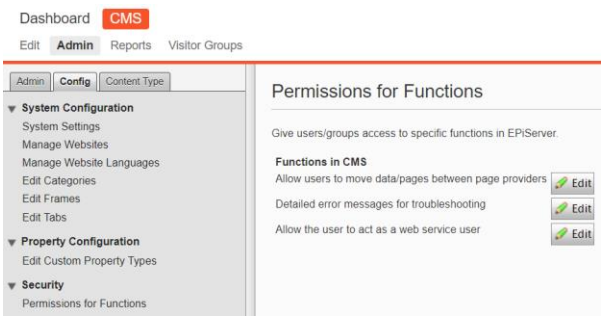
17. Click **Save**.

18. In Admin view, click **Content Type**, click **[Products] Product**, and then click **Settings**, as shown in the following screenshot:



19. At the bottom of the **Information** tab, click **Add Users/Groups**, search and add the **Marketers** group, clear the check box for **Everyone**, and click **Save**, as shown in the following screenshot:

20. In Admin view, click **Config**, click **Permissions for Functions**, and then next to **Detailed error messages for troubleshooting**, click **Edit**, as shown in the following screenshot:



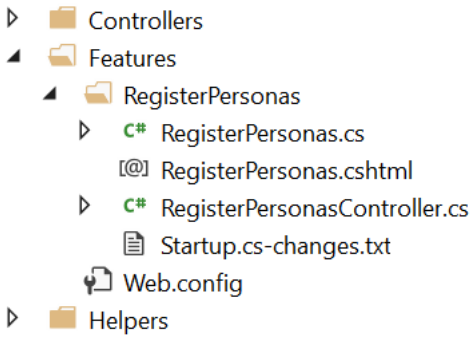
21. Click **Add Users/Groups**, search and add the **CmsAdmins** and **Developers** groups, clear the check box for **Administrators**, and click **Save**, as shown in the following screenshot:

22. Close the browser.

Automating the management of groups, users, and access rights

Now that you have seen how to manage authorization by manually using Admin view, you will write some code to automate the process to create some more groups and users and set their access rights.

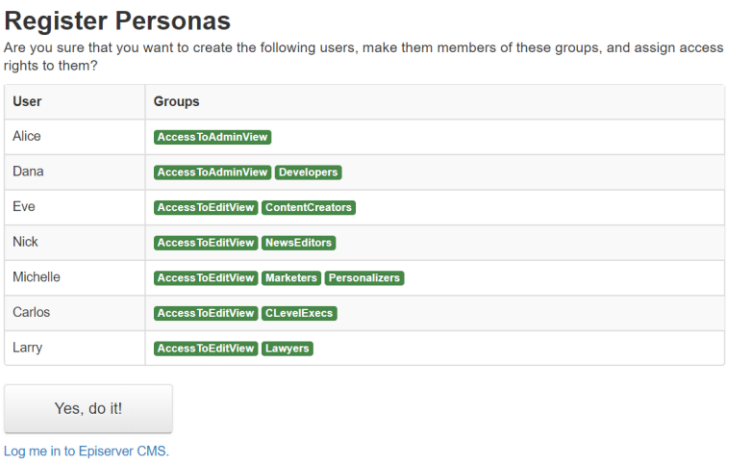
1. If you haven't done so already, extract the folders and files in **cmsdevfun_exercisefiles.zip**.
2. Drag and drop the **\cmsdevfun-exercisefiles\Module A\A2\Features** folder into the **AlloyDemo** project.
3. Expand the **Features** folder and review the files included, as shown in the following screenshot:
    a. **RegisterPersonas.cs**: a static class to enable the feature.
    b. **RegisterPersonas.cshtml**: a Razor file for the user interface of the feature.
    c. **RegisterPersonasController.cs**: a controller that performs the work of the feature.



4. In **AlloyDemo**, in the root of the website, open **Startup.cs**, and import the `AlloyDemo.Features.RegisterPersonas` namespace.
5. Add a statement to the `Configuration` method to call the `UseRegisterPersonas` extension method that enables registering of personas, as shown in the following code:

```
app.UseRegisterPersonas(() => HttpContext.Current.Request.IsLocal);
```

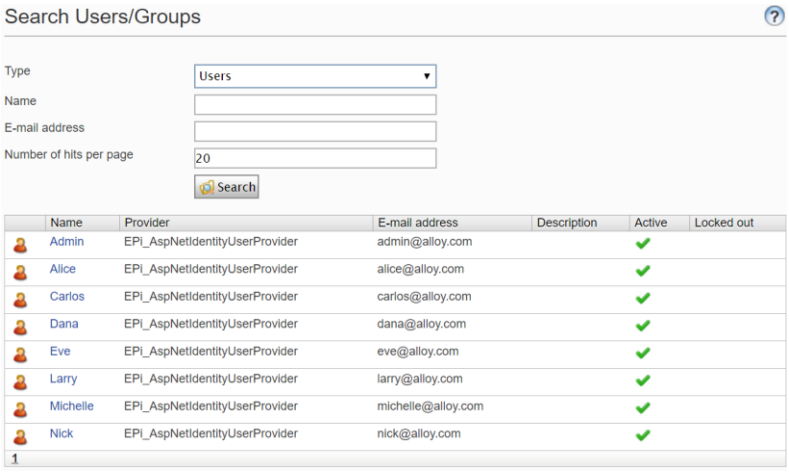
6. Start the website, and note that a page appears offering to register some personas for you, as shown in the screenshot:



> If all the users already exist, then this page will not appear. If necessary, you can use Admin view to delete one of the users to force it to appear.

7. Click **Yes, do it!**, and note the success message.
8. Log in as **Admin** or **Alice** and navigate to **CMS** | **Admin** | **Admin** | **Search User/Group**.

9. Click **Search** to list all the users, as shown in the following screenshot:



10. Click each new user to see which group(s) they have been made a member of and note that three new groups that were created: **CLevelExecs**, **Lawyers**, and **Personalizers**.
11. Open **Startup.cs**, comment out the statement that calls the **UseRegisterPersonas** extension method, and save changes.

Optional: Review alternative authentication providers

Review the following topics in the documentation and popular blog articles:

- Federated security: http://world.episerver.com/documentation/developer-guides/CMS/security/federated-security/
- OWIN authentication: http://world.episerver.com/documentation/developer-guides/CMS/security/owin-authentication/
- EPiServer CMS UI AspNetIdentity OWIN authentication: http://world.episerver.com/documentation/developer-guides/CMS/security/episerver-aspnetidentity/
- Configuring mixed-mode OWIN authentication: http://world.episerver.com/documentation/developer-guides/CMS/security/configuring-mixed-mode-owin-authentication/
- Configuring Active Directory membership provider: http://world.episerver.com/documentation/developer-guides/CMS/security/Configuring-Active-Directory-membership-provider/
- Integrate Azure AD using OpenID Connect: http://world.episerver.com/documentation/developer-guides/CMS/security/integrate-azure-ad-using-openid-connect/

Exercise A3 – Optional: Creating, editing, saving, and publishing content

In this exercise, you will get an understanding of how an editor works in the Episerver CMS. You will create a new page, add some links and images, and publish the page.

Prerequisites: complete Exercises A1 and A2.

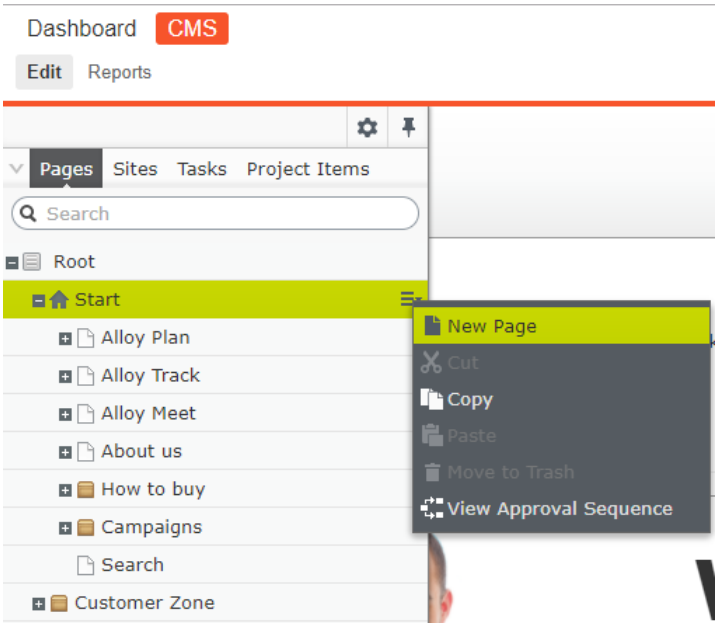
While working through these exercises, note the groups, users, access rights, and resources that were defined in Exercise A2, as summarized in the following table:

Groups	Users	Access rights	Resources
AccessToAdminView	Admin, Alice	All	All
AccessToAdminView, Developer	Dana	All	All
ContentCreators	Eve	Read, Create, Change, Delete	All content, except News & Events and Product pages
NewsEditors	Nick	Full	News & Events
Marketers	Michelle	Create, Publish	All content, except News & Events
		Create type	Product pages

Adding and publishing a new product page

1. Start the **AlloyDemo** website, log in as **Eve**, and note that she only has access to **CMS** | **Edit** and **CMS** | **Reports**, as shown in the following screenshot:

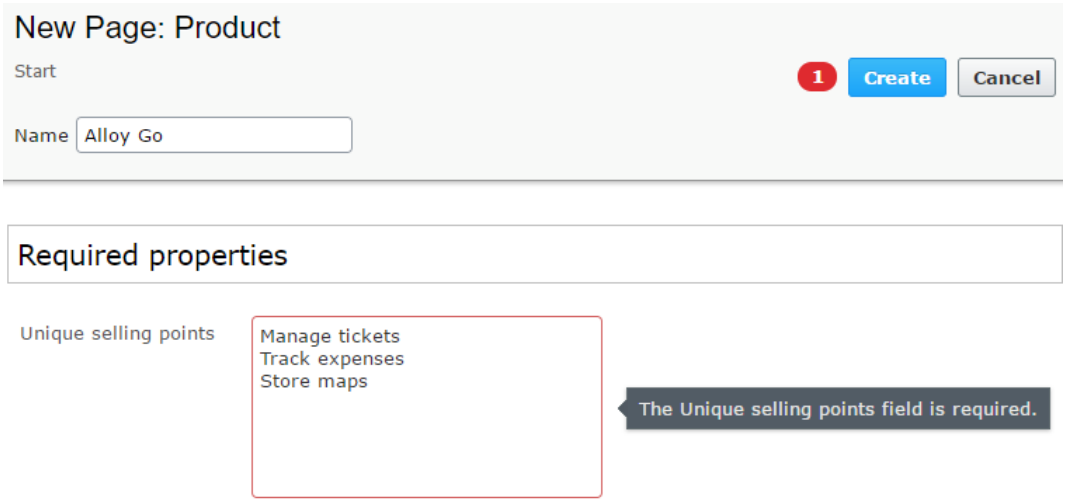
2. Under the **Start** page, add a new page, as shown in the following screenshot:



3. In the list of page types, note that Product is not available for Eve, because only members of Marketers have access rights to create product pages.

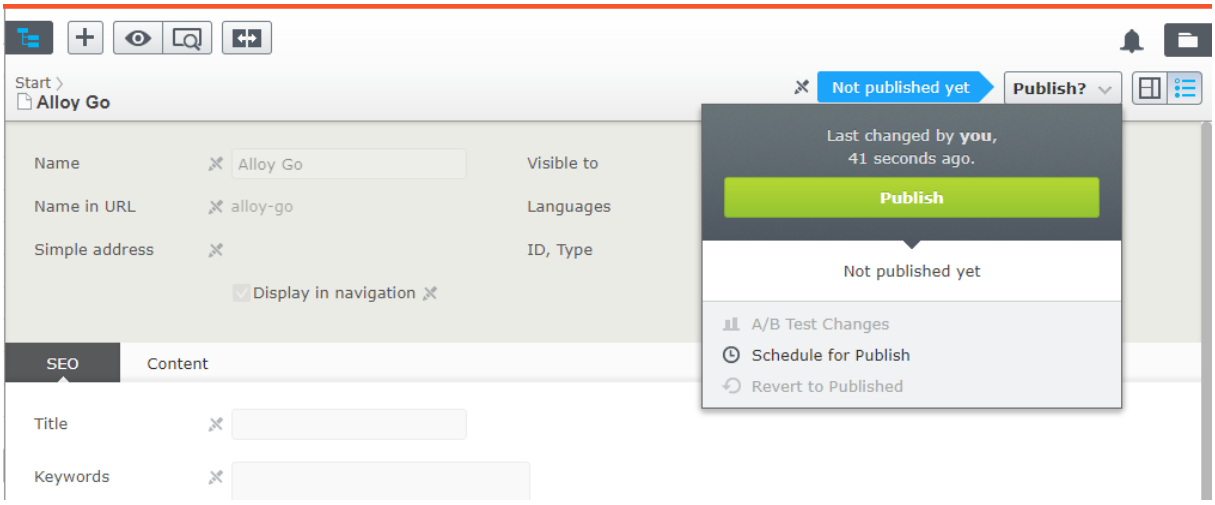
4. Log out **Eve**, and log in as **Michelle**, and note that she has access to **CMS** | **Edit**, **CMS** | **Reports**, and **CMS** | **Visitor Groups**, as shown in the following screenshot:

5. Add a new page under **Start**.
6. Click **Product**, enter the name **Alloy Go**, and note that you must enter some USPs, as shown in the following screenshot:



Alloy Go will be a travel management software app that manages employee expenses, tickets, and maps.

7. Enter some USPs, and click **Create**.
8. Note that Michelle can create a product page, and she can publish it, but she cannot edit it, as shown in the following screenshot:

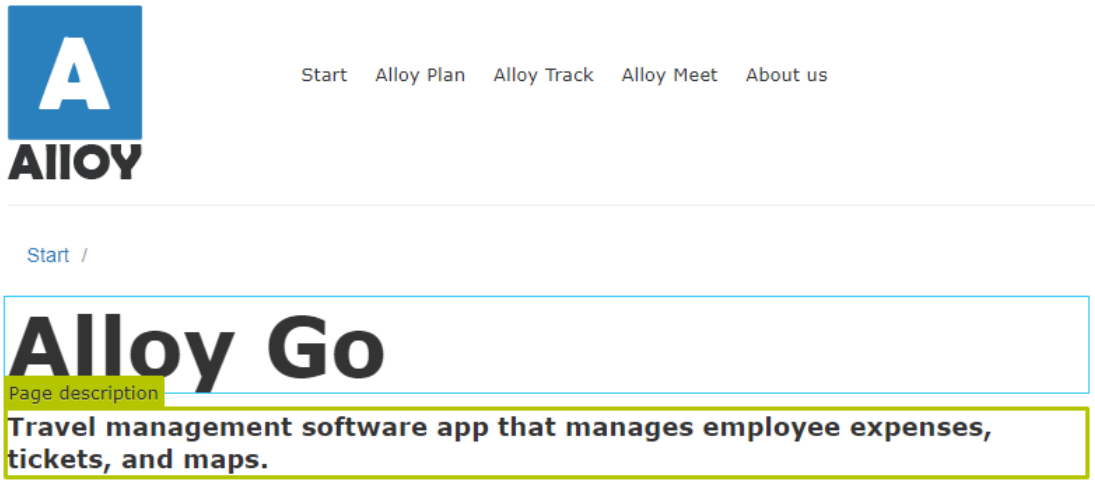


9. **Publish** Alloy Go.
10. Log out as **Michelle**.

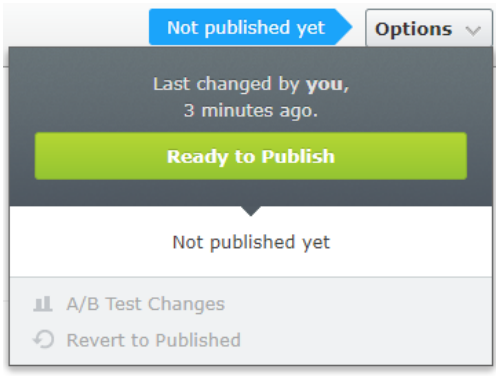
Editing the product page

1. Log in as **Eve**.
2. Edit **Alloy Go**.

3. Enter a page description, as shown in the following screenshot:



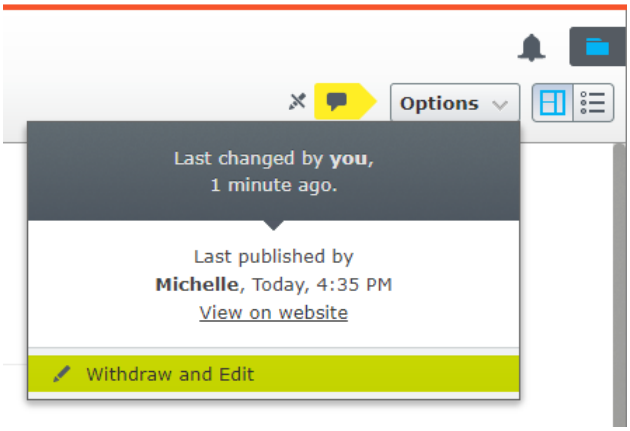
4. Enter some text for the main body. Be creative.
5. Click **Options** button, and note that Eve cannot publish.
6. Click **Ready to Publish**, as shown in the following screenshot:



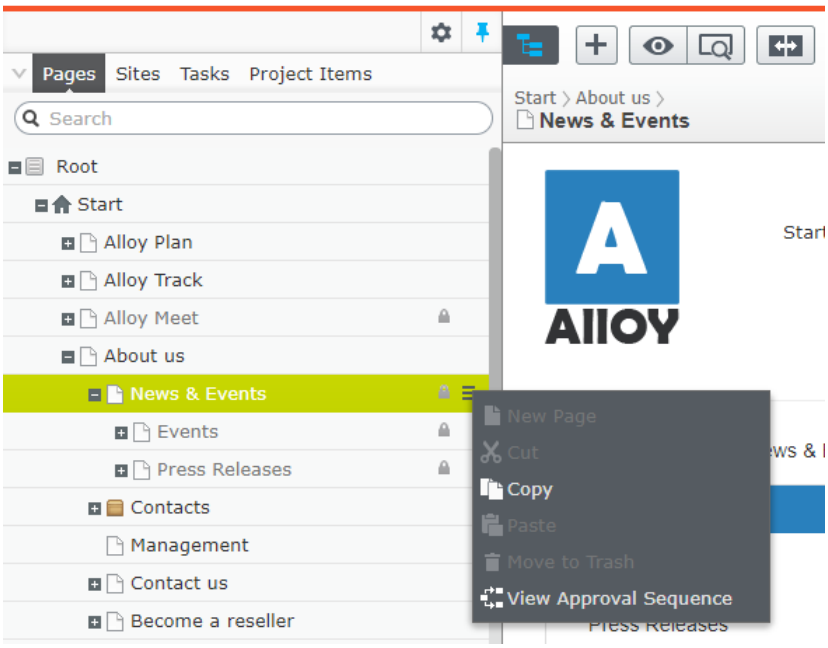
7. In **Assets** pane, select **Media** tab, and under the **For All Sites** folder, create a new folder named **Alloy Go**.
8. Use your favourite search engine to find some suitable images of travel related items, and upload them to the **Alloy Go** folder.
9. Drag and drop the image(s) into the main body, and explore the image editor feature.
10. Create a **Teaser** block in the **Alloy Go** folder.


You will only be able to set the teaser to be Ready to Publish at this point.

11. Open the **Alloy Go** page. You previously set Alloy Go page to **Ready to Publish**, so it is locked. Click **Options**, and select **Withdraw and Edit**, as shown in the following screenshot:



12. Drag and drop the teaser block you created into the MainContentArea on Alloy Go page.
13. Click **Options**, and select **Ready to Publish**.
14. In **Navigation** pane, try to add a new page under **About us** | **News & Events**, and note that section and its children are locked for Eve, as shown in the following screenshot:

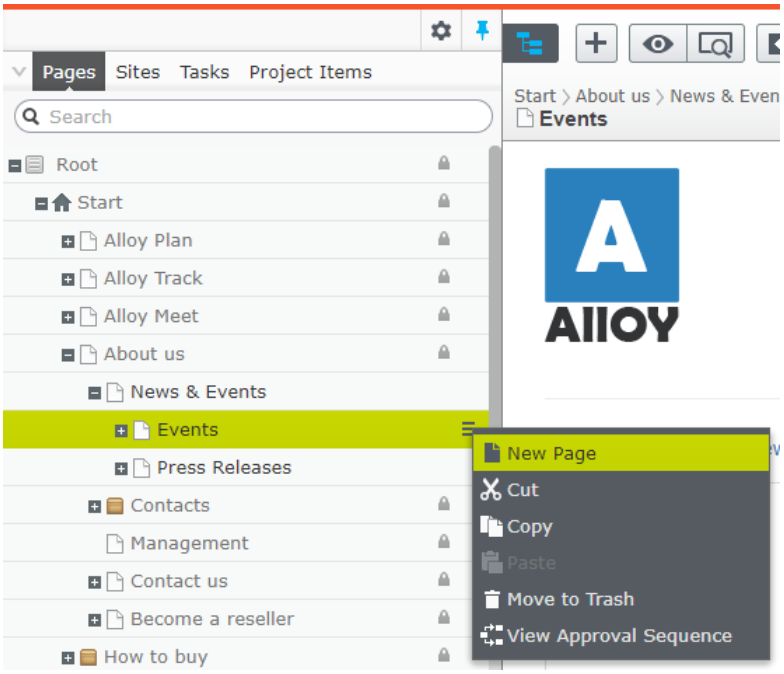


15. Log out as Eve.

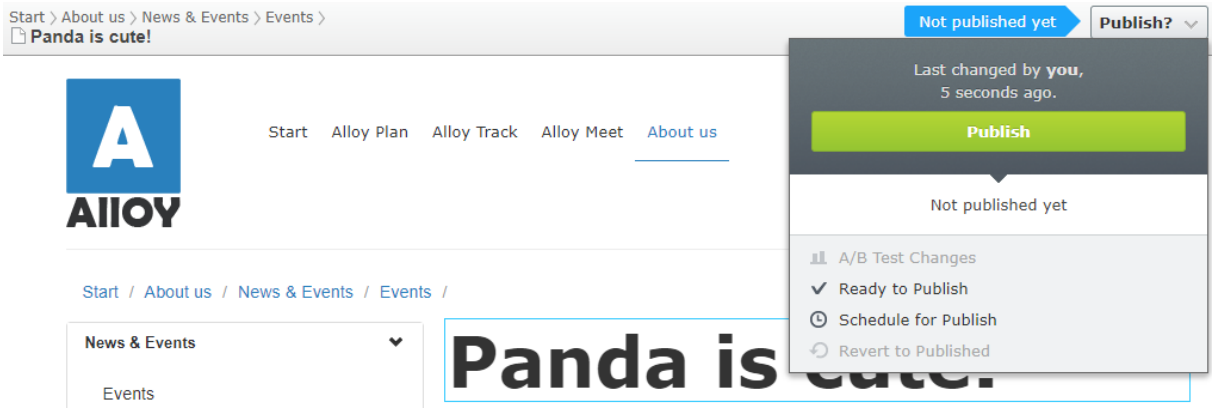
Adding and publishing a new article page

1. Log in as Nick.

2. In **Navigation** pane, try to add a new page under **About us** | **News & Events** | **Events**, and note that section and its children are available for Nick, but all other pages are locked, as shown in the following screenshot:



3. Create an **Article** page under **About us** | **News & Events** | **Events**, named **Panda is cute!**
4. **Publish** the new page, as shown in the following screenshot:

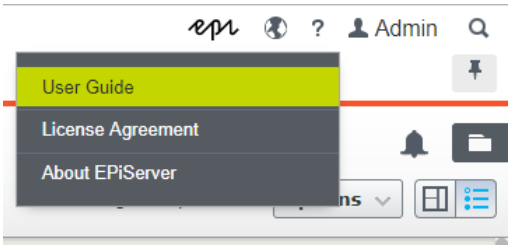


5. Log out as **Nick**.
6. Log in as **Larry**, and note that all content is locked for Larry.
7. Close the browser.

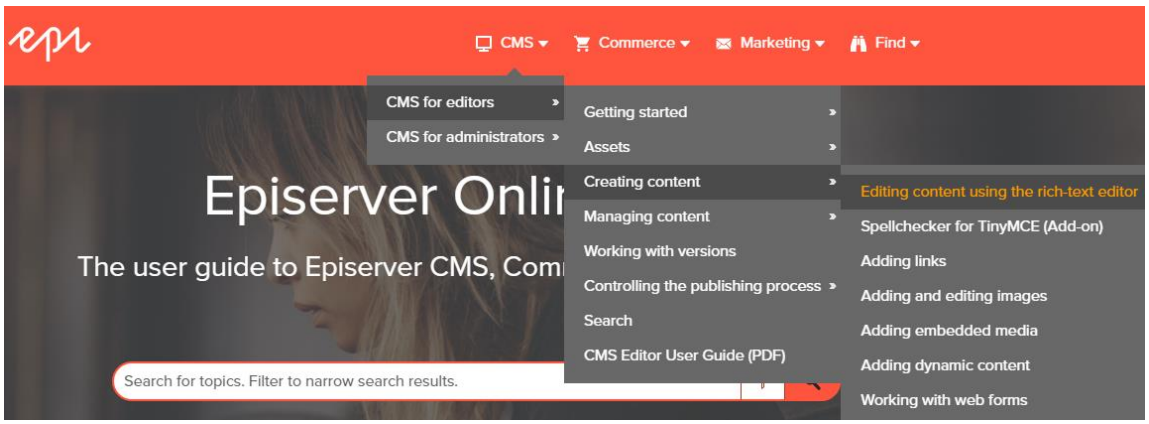
Reviewing the online user guides

1. Start the AlloyDemo website, and log in as anyone.

2. In the **Global** menu, click **Help** | **User Guide**, as shown in the following screenshot:



3. Navigate to **CMS** | **CMS for editors** | **Creating content**, and review some of the editing features like using the rich-text editor, adding and editing images, or Working with versions, as shown in the following screenshot:



Exercise A4 – Optional: Personalizing, approving, and A/B testing content

In this exercise, you will get an understanding of how personalization with visitor groups works, and how content approvals and A/B testing for marketing works. You will create a new page as one user, and then approve it as a sequence of other users. You will also perform A/B testing on some content.

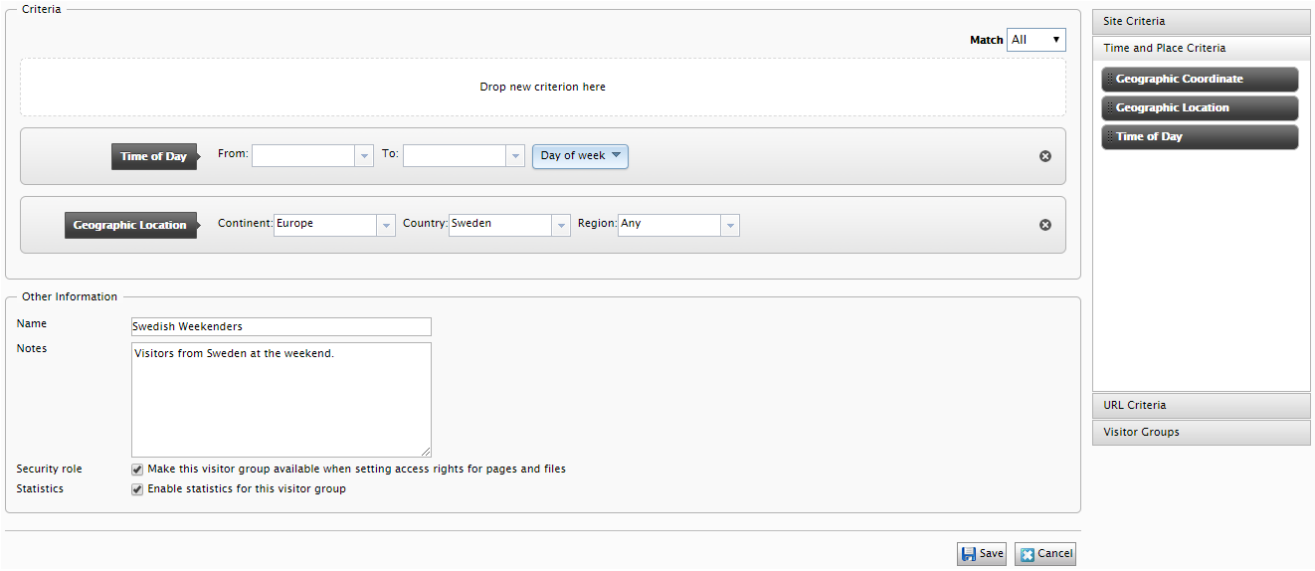
Prerequisites: complete Exercises A1 and A2.

While working through these exercises, remember the groups, users, access rights, and resources that you defined in Exercise A2, as summarized in the following table:

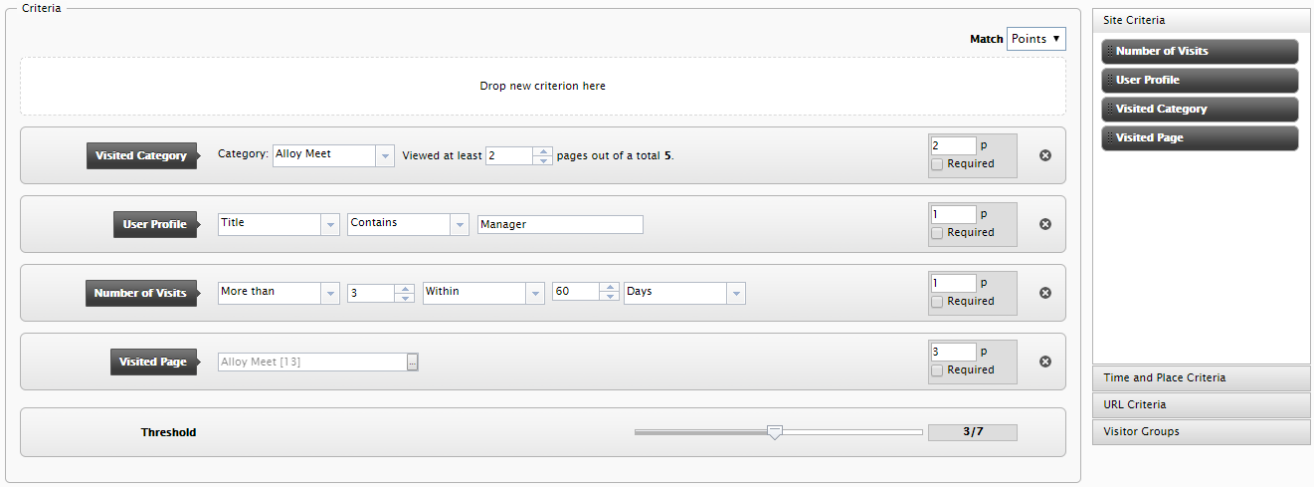
Groups	Users	Access rights	Resources
ContentCreators	Eve	Full, except Administer	All content, except News & Events
NewsEditors	Nick	Full	News & Events
Marketers	Michelle	Create, Publish	All content, except News & Events
		Create type	Product pages
AccessToAdminView	Alice	All	All
Lawyers	Larry	n/a	n/a
CLevelExecs	Carlos	n/a	n/a

Define some visitor groups for personalization

1. Open the **Training** solution with the **AlloyDemo** project.
2. Start the **AlloyDemo** site, and log in as **Alice**.
3. Navigate to **CMS** | **Visitor Groups**, and click **+ Create**.
4. Enter the following, as shown in the following screenshot:
    a. Name: Swedish Weekenders
    b. Notes: Visitors from Sweden at the weekend.
    c. Security role: selected
    d. Statistics: selected
5. Drag and drop from the **Time and Place Criteria** section, as shown in the following screenshot:
    a. **Geographic Location**: Europe, Sweden
    b. **Time of Day**: Day of week: Saturday and Sunday



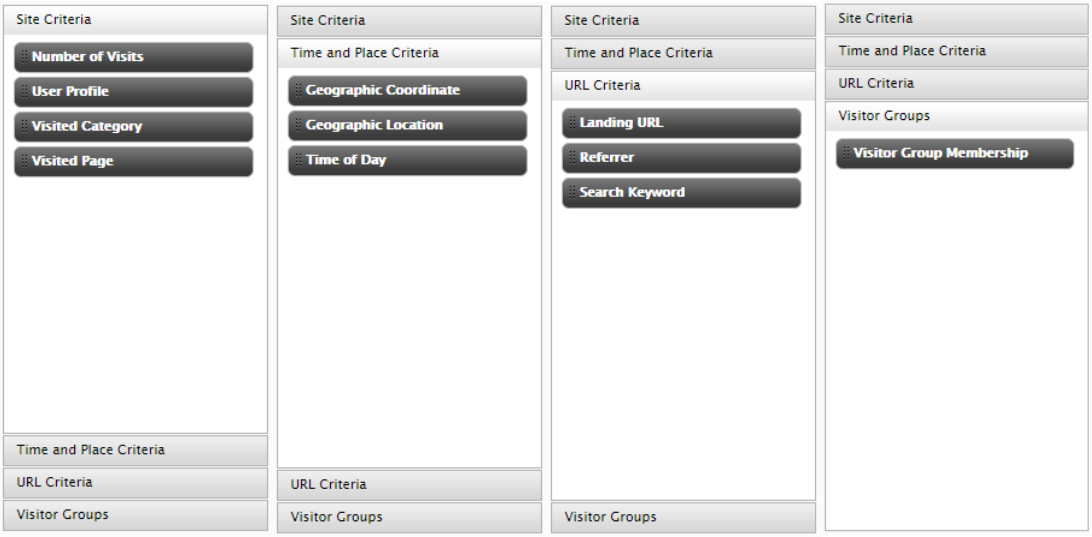
6. Click **Save**.
7. Click **+ Create**.
8. Enter the following, as shown in the following screenshot:
    a. Name: Alloy Meet Promotion
    b. Notes: Visitors who have expressed an interest in Alloy Meet.
    c. Security role: selected
    d. Statistics: selected
    e. Match: Points
9. Drag and drop from the **Site Criteria** section, as shown in the following screenshot:
    a. Visited Category: Alloy Meet, at least 2 pages; 2 points
    b. User Profile: Title contains Manager; 1 point
    c. Number of Visits: More than 3 within 60 days; 1 point
    d. Visited Page: Alloy Meet; 3 points



10. Set **Threshold** to 3 out of 7 points.
11. Cick **Save**.

Install some extra visitor groups criteria

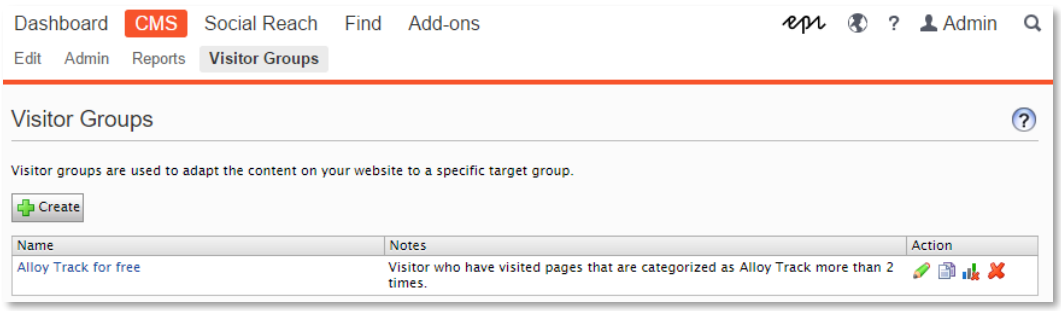
The default 11 visitor groups criteria are shown in the following screenshots:



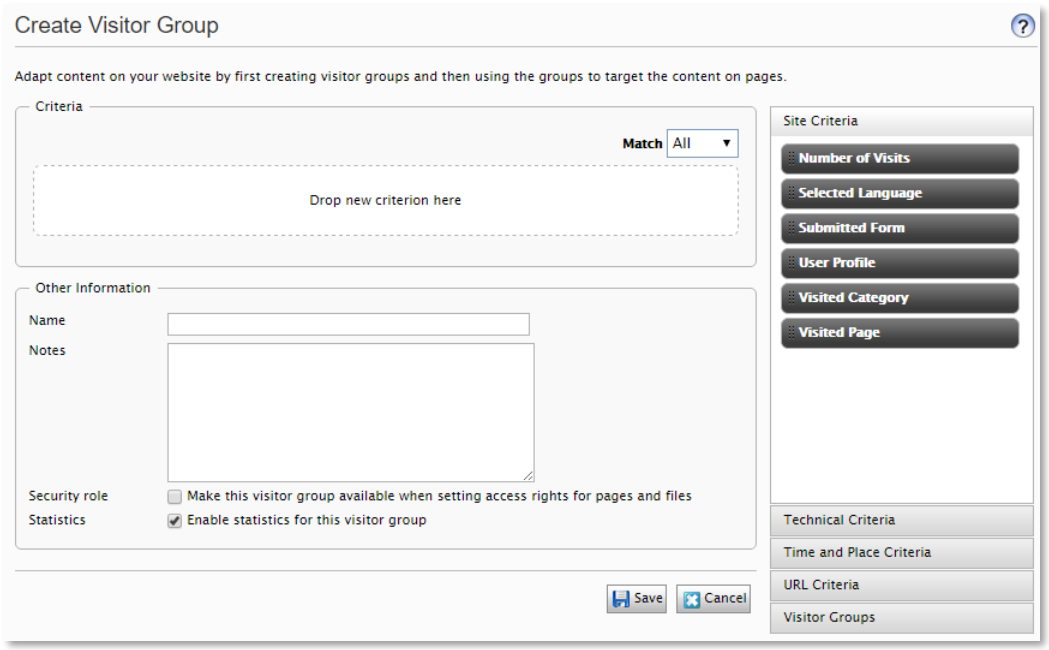
After installing the *Visitor Groups Criteria Pack* add-on, you will have 11 more.

1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. Enter the following command:

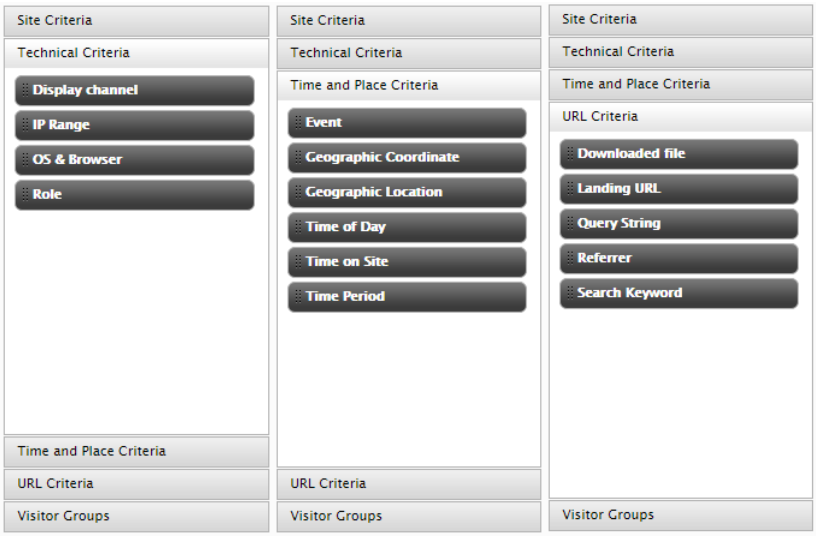
   ```
   Install-Package -ProjectName AlloyDemo EPiServer.VisitorGroupsCriteriaPack
   ```
3. Start the website and log in as **Admin**.
4. Navigate to **CMS** | **Visitor Groups**.



5. Click **Create**, and note the extra criteria, such as **Selected Language** and **Submitted Form**, as shown in the following screenshot:

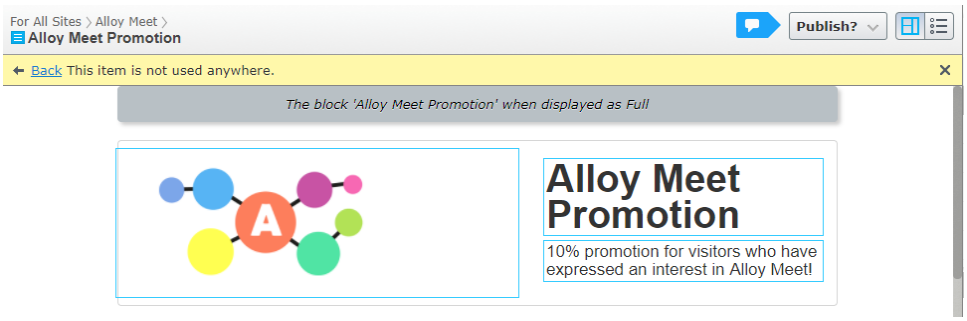


The Visitor Groups Criteria Pack adds 11 more criteria, as shown in the following screenshots:

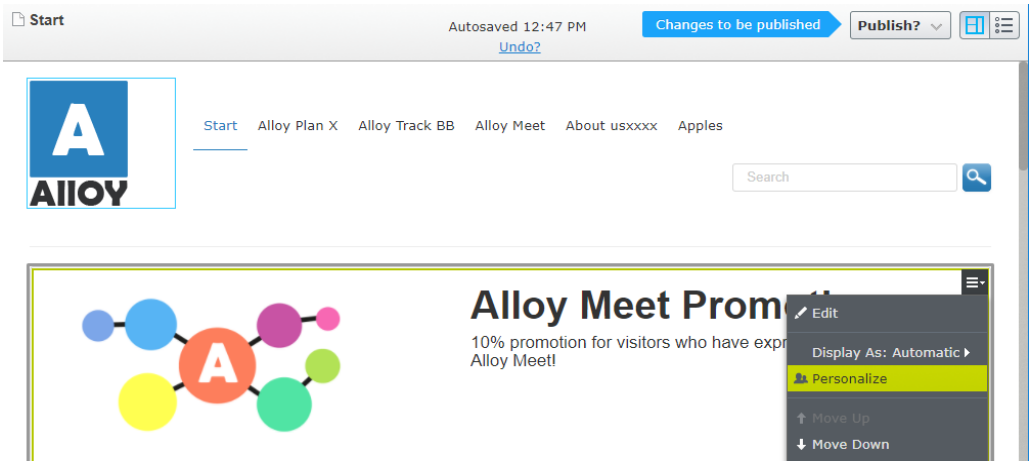


Create some personalized content

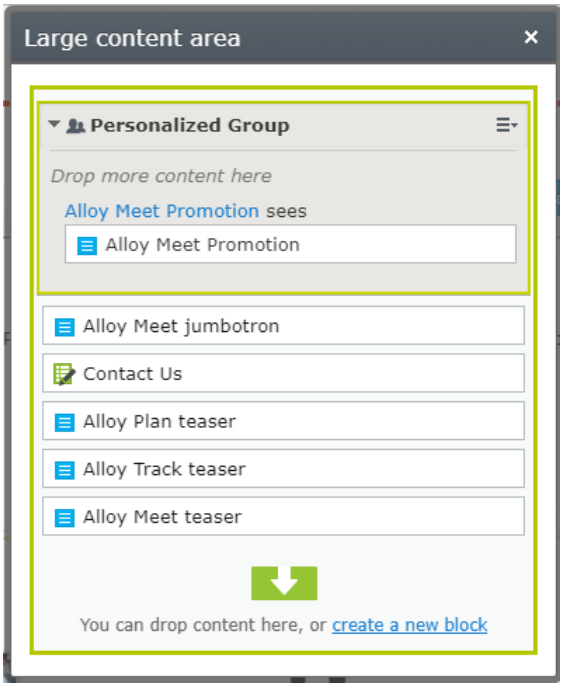
1. In Edit view, open **Assets** panes, on the **Blocks** tab, in the **Alloy Meet** folder, create a **Teaser** block named **Alloy Meet Promotion**, as shown in the following screenshot:



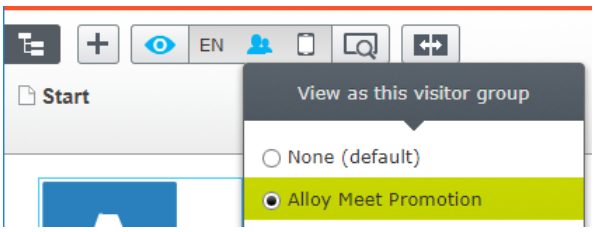
2. **Publish** the block.
3. Edit the **Start** page, and drag and drop **Alloy Meet Promotion** to the top of its main content area.
4. In the block's context menu, click **Personalize**, as shown in the following screenshot:



5. Select **Alloy Meet Promotion**, as shown in the following screenshot:



6. To preview the page as if you are a member of the visitor group, in the toolbar, toggle view settings, click **Alloy Meet Promotion**, as shown in the following sreeenshot:



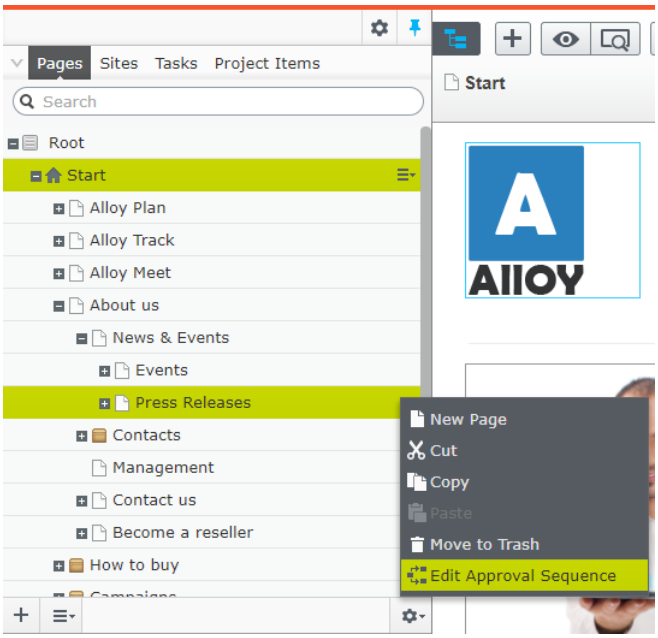
Explore content approvals

You can read the Episerver CMS *Editor User Guide* to learn how to use this feature.

http://webhelp.episerver.com/latest/cms-edit/content-approvals.htm

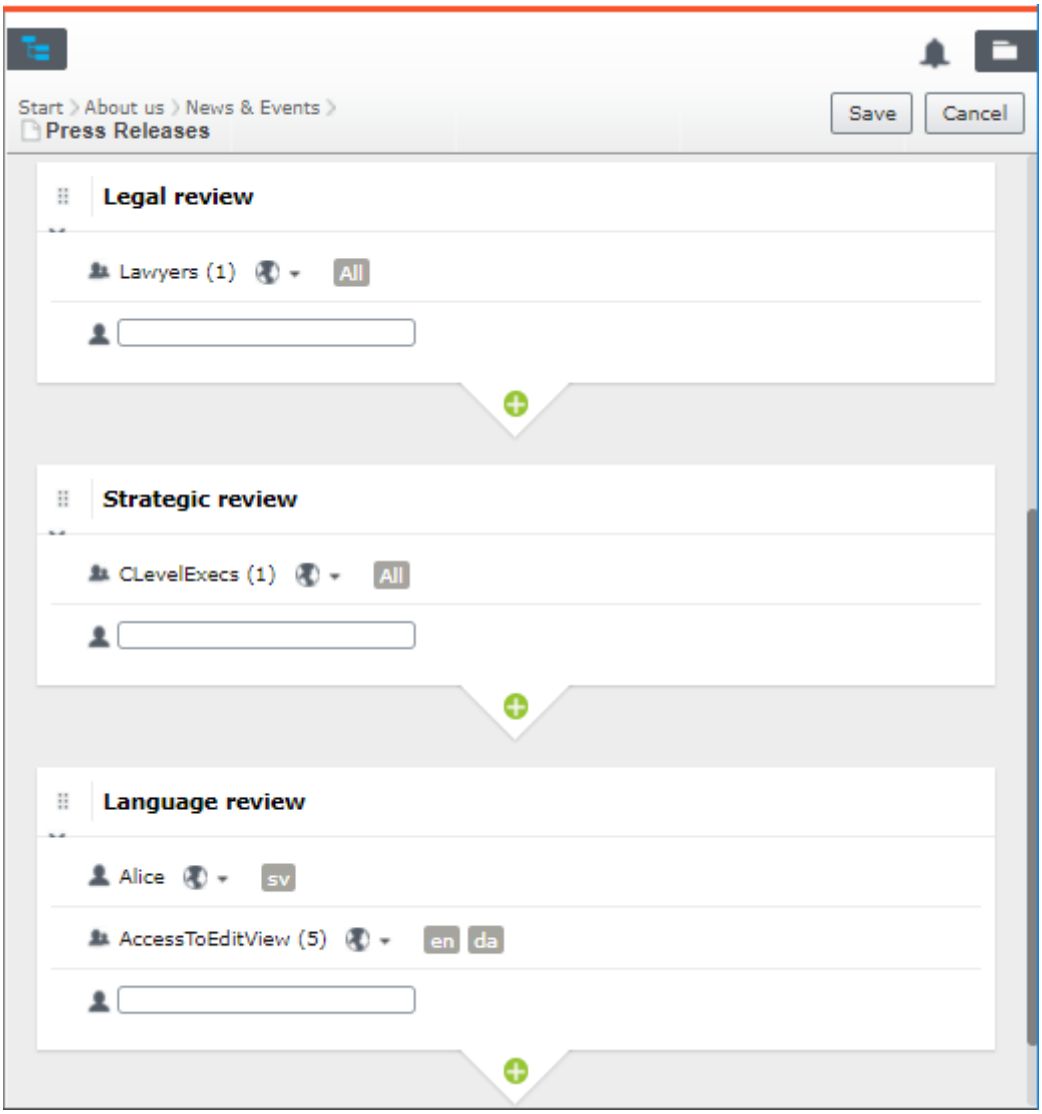
Episerver CMS 10.10 and later adds support for groups/roles in approval sequence steps, as well as users.

1. Open the **Training** solution with the **AlloyDemo** project.
2. Start the **AlloyDemo** site, and log in as **Alice**.
3. Navigate to **CMS** | **Edit**, and view the **Pages** in the **Navigation** pane.
4. Expand **About us** and **News & Events**.
5. Click the content menu for **Press Releases**, and choose **Edit Approval Sequence**, as shown in the following screenshot:



6. Set the approval sequence for the **Press Releases** page to **Enabled**.

7. Set **Require comment on Decline**.
8. Add three steps, and assign the groups that you created in Exercise A2, as shown in the following screenshot, and as listed in the following lettered bullets:
    a. Legal review by a lawyer
    b. Strategic review by a C-Level executive
    c. Review of Swedish content by Alice, who speaks Swedish fluently, and a review of other languages by anyone with access to Edit view



It is good practice to use groups with at least two members as reviewers in each step. Assigning Alice to the Language review step is bad practice, because she might not be available to approve or decline.

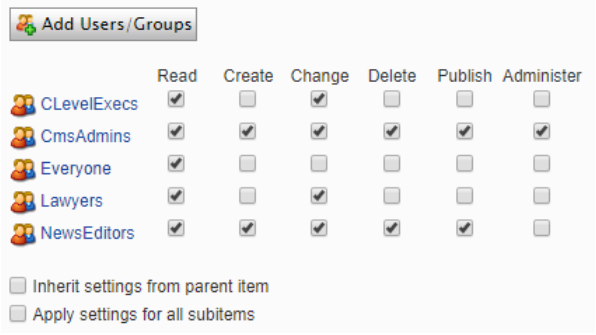
9. Save the sequence.

Enabling group members to approve

The members of the **Lawyers** and **CLevelExecs** groups have access to Edit view, but they have no other access rights. They must have at least **Change** access right to **Press Releases** to be able to approve content.

1. In Admin view, navigate to **Admin** | **Access Rights** | **Set Access Rights**.
2. In the content tree, expand **About us**, expand **News & Events**, and select **Press Releases**.

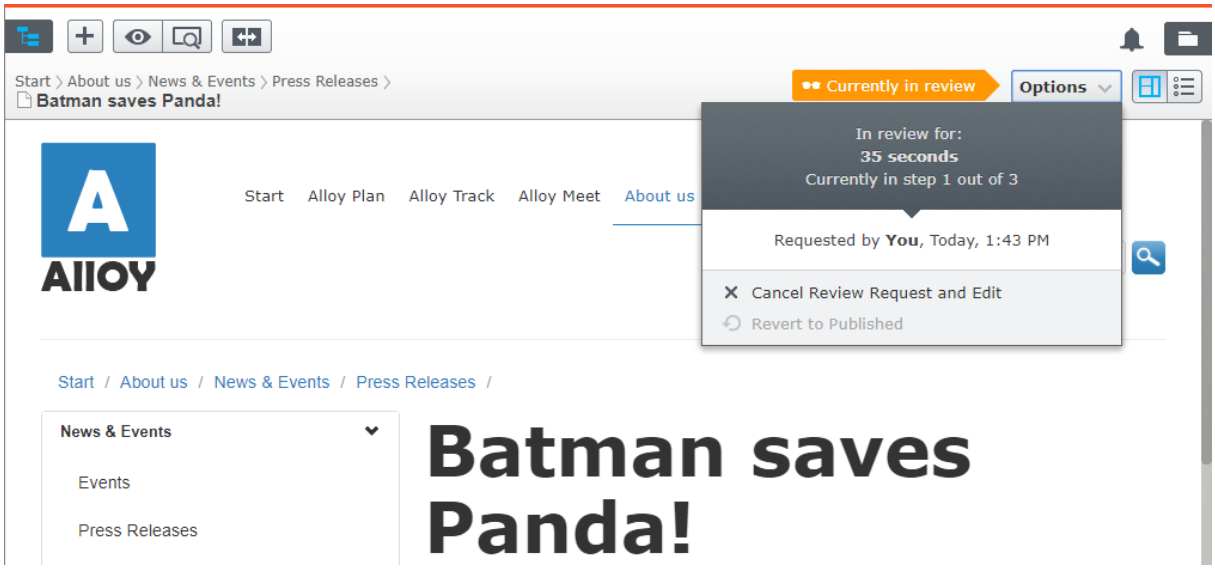
3. Clear the **Inherit settings from parent item** check box.
4. Click **Add/Users Groups**.
5. Search for, and add, **Lawyers** and **CLevelExecs**.
6. Give **Change** access rights to **Lawyers** and **CLevelExecs**, as shown in the following screenshot:



7. Click **Save**.

Test the approval sequence

1. Log in as **Nick**.
2. Add a new **Article** to the **Press Releases** named **Batman saves Panda!**
3. Mark the page as **Ready for Review**, and note that the page is currently in review in step 1 of 3, as shown in the following screenshot:



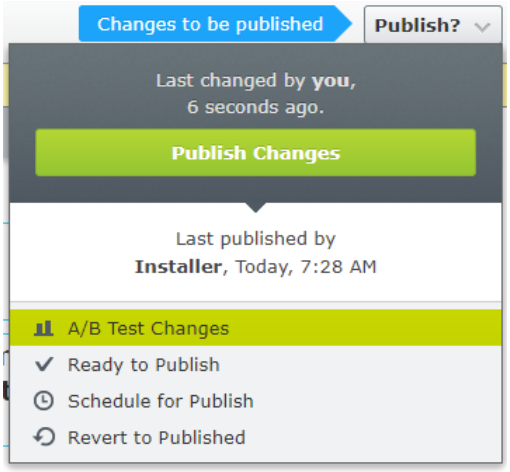
4. Log out, and log in again as **Larry**.
5. Click the notification bell, click the notification, and **Approve** the article.
6. Log out, and log in again as **Carlos**.
7. Click the notification bell, click the notification, and **Approve** the article.
8. Log out, and log in again as **Nick**.
9. Click the notification bell, click the notification, and **Approve** the article.
10. Publish the article.
11. Repeat the process for a new article, but see what happens when a lawyer declines approval.

Explore A/B testing

You can read the Episerver CMS *Editor User Guide* to learn how to use this feature if your instructor did not demonstrate it.

http://webhelp.episerver.com/latest/cms-edit/ab-testing.htm

Experiment with the A/B testing, as shown in the following screenshot:



Exercise A5 – Optional: Localizing content

In this exercise, you will localize some content into Swedish and Danish, including pages and blocks.

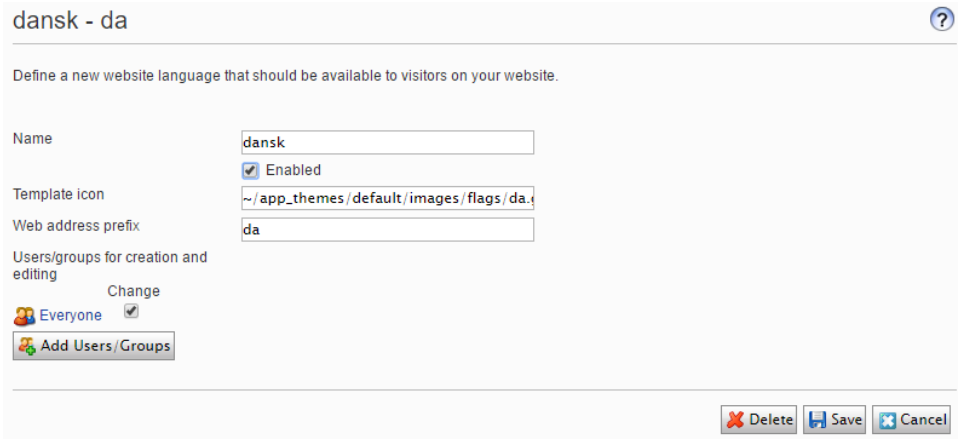
Prerequisites: complete Exercise A1.

Enabling Danish language for the website content

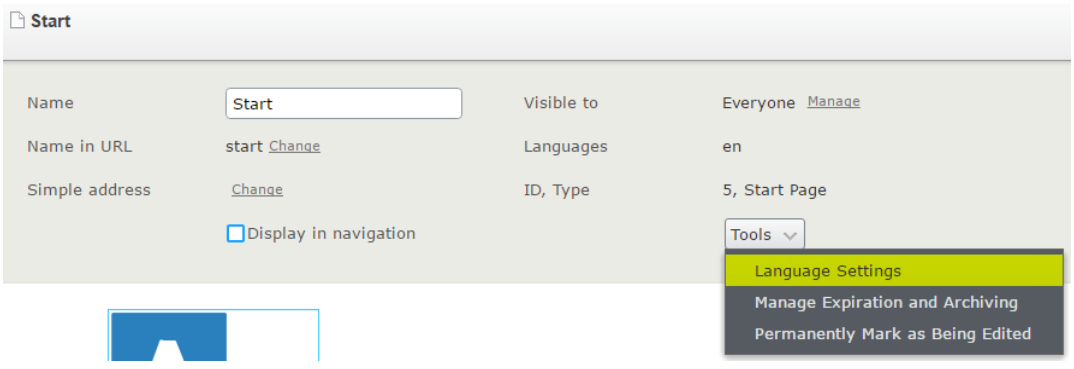
1. Open the **Training** solution with the **AlloyDemo** project.

Make sure you use **AlloyDemo**, so you have lots of sample content pages that you can translate.

2. Start the site, and log in as **Admin**.
3. Navigate to **CMS** | **Admin** | **Config** | **Manage Website Languages**, and note that English and Swedish are already enabled by the **Alloy (MVC)** project template.
4. Click Danish (**dansk**) language.
5. Check the **Enabled** box, click **Save**, as shown in the following screenshot:

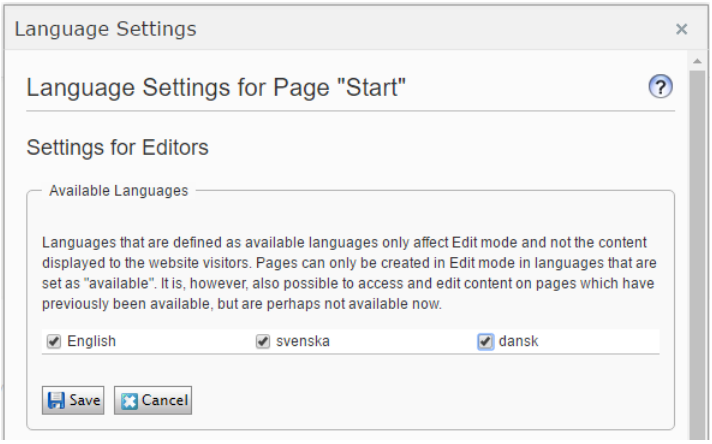


6. Navigate to **CMS** | **Edit** | **Pages** | **Start** page.
7. Click **Tools** | **Language Settings**, as shown in the following screenshot:



8. In **Settings for Editors**, click **Change**.

9. Check the **dansk** box, and click **Save**, as shown in the following screenshot:



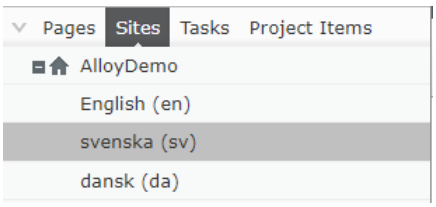
10. In **Settings for Site Visitors** | **Fallback Languages**, click **Change**.
11. Set **Swedish** to fallback to **Danish**, and then **English**.
12. Set **Danish** to fallback to **Swedish**, with no second fallback, then click **Save**, as shown in the following screenshot:

We could have given Danish a second fall back, but in the following steps, you will see what happens if you don't have one.

13. Close the **Language Settings** dialog box.

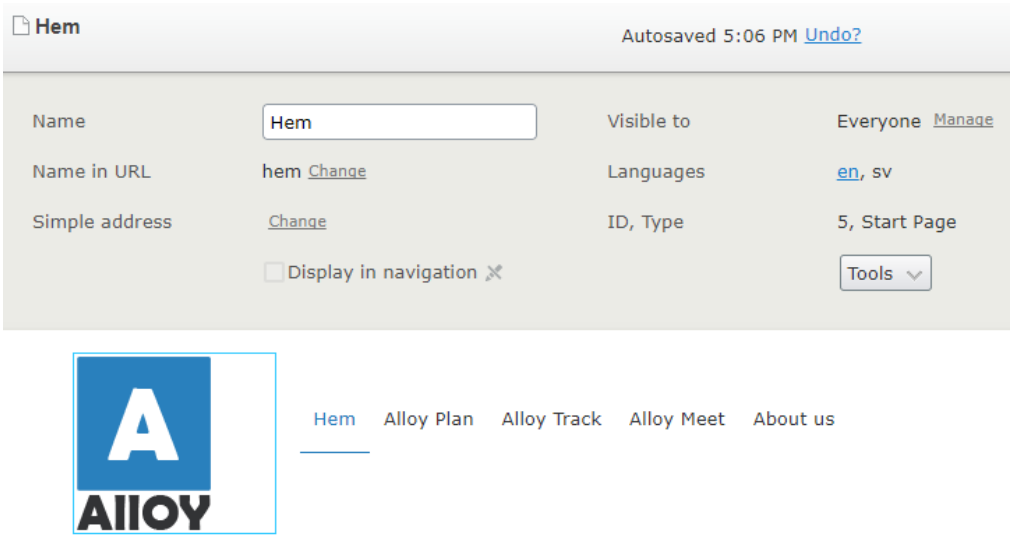
Translating content

1. In the **Navigation** pane, click **Sites**, and then switch to the **svenska** (Swedish) site, as shown in the following screenshot:



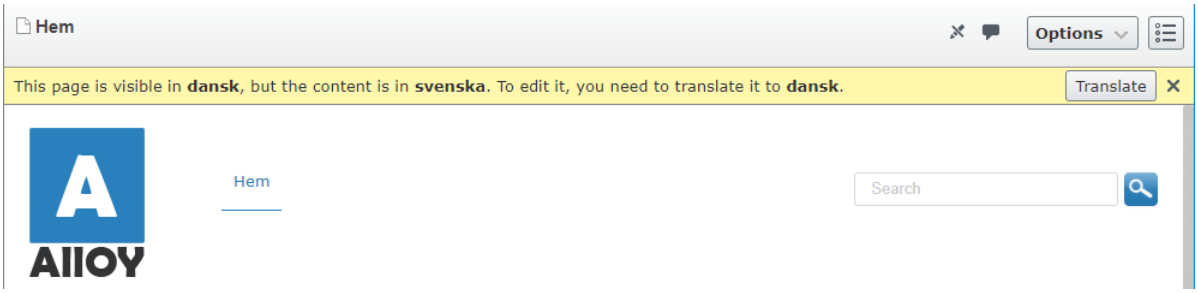
2. Note the **Start** page has already been translated into Swedish in the Alloy (MVC) project template.

3. Edit the name of the page to **Hem** (home in Swedish) and the **Name in URL** to **hem**, as shown in the following screenshot:

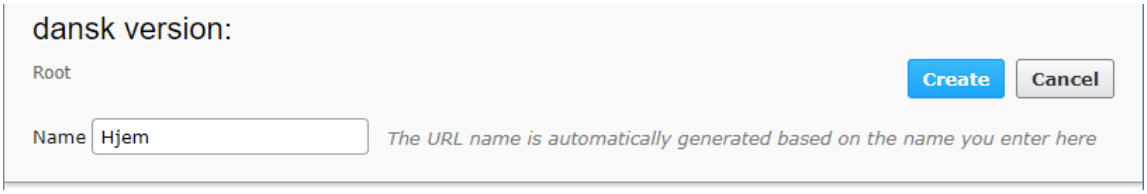


You can quickly toggle between languages for a page by clicking the language code links in the basic information area.

4. Publish the changes.
5. In **Navigation**, click **Sites**, and switch to the Danish language site, and note the page is visible to a visitor who asks for Danish (dansk) because it falls back to Swedish, as shown in the following screenshot:

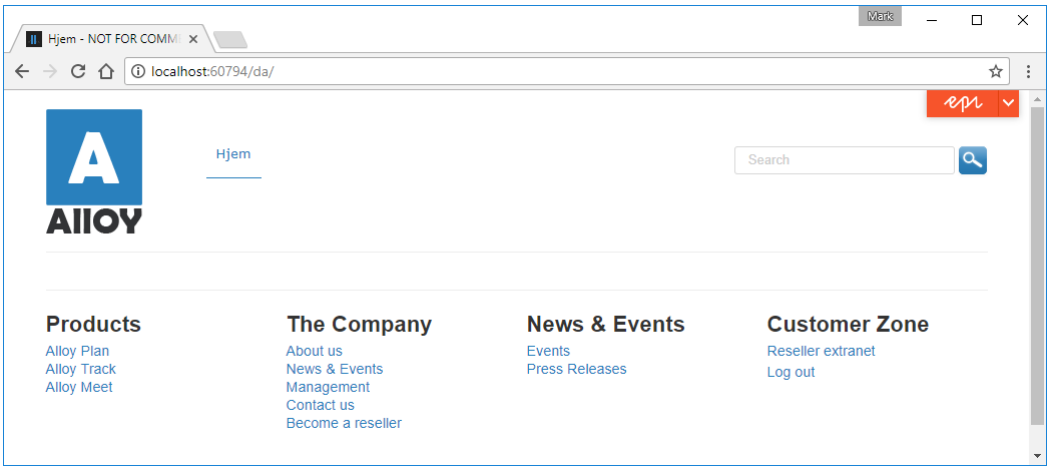


6. Click **Translate**. Note the page is NOT translated automatically for you. You must translate the text yourself (**Hjem** is Danish for home), and then click **Create**, as shown in the following screenshot:

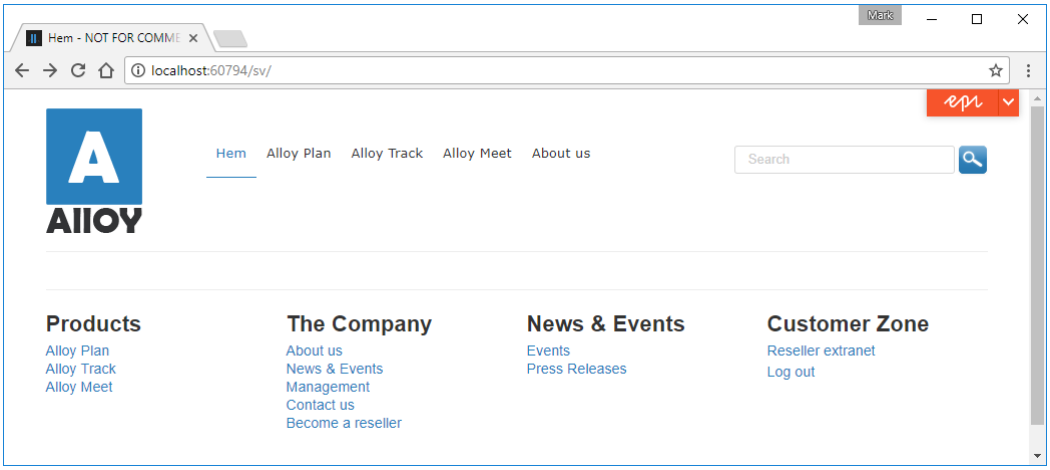


7. Publish the page.
8. View the site as a visitor, and enter /da/ at the end of the address box. Note that when viewed in Danish, the Danish Start page (which is empty) is displayed, and there are no links to other pages

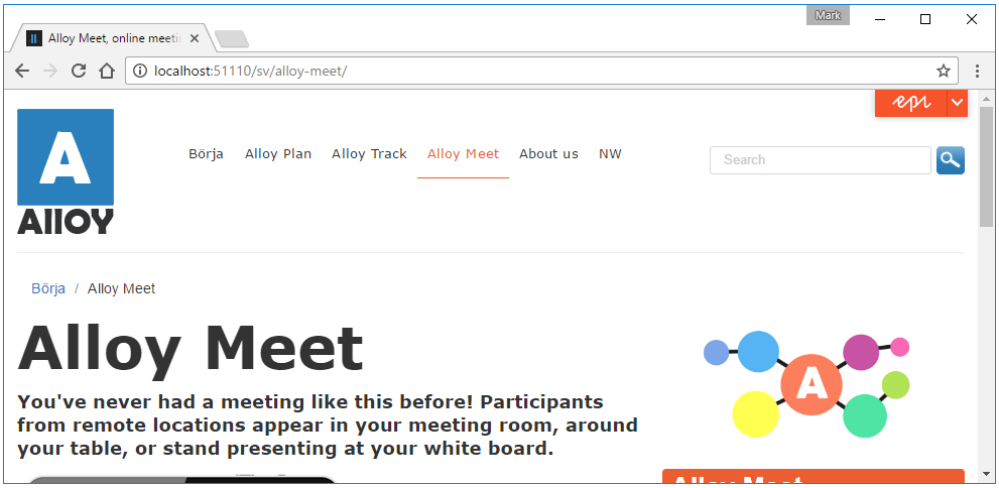
like products, because Danish can only fallback to Swedish pages, as shown in the following screenshot:



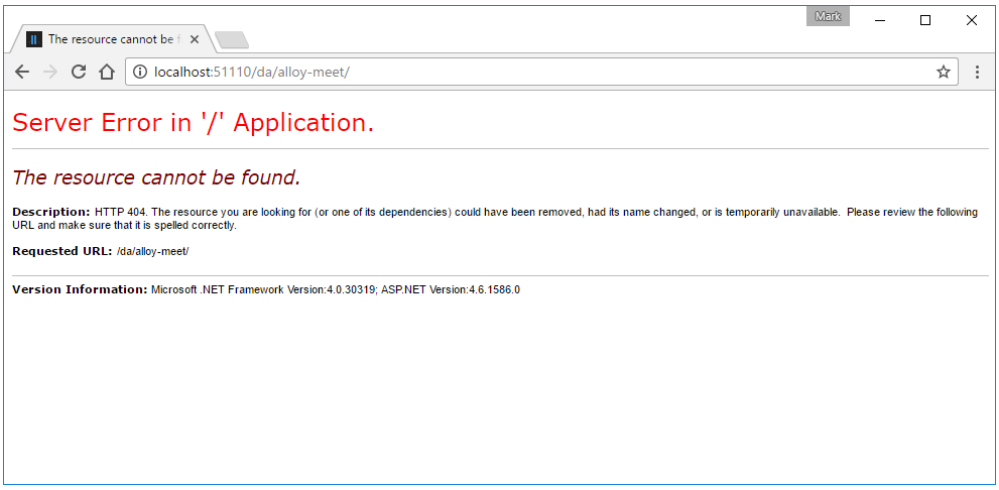
9. Change /da/ to /sv/ to request the Swedish start page, and note that due to fallback language settings, when content is not available in Swedish, it can fall back to English content, for example links to product pages, as shown in the following screenshot:



10. View the **Alloy Meet** page. It falls back to English, as shown in the following screenshot:



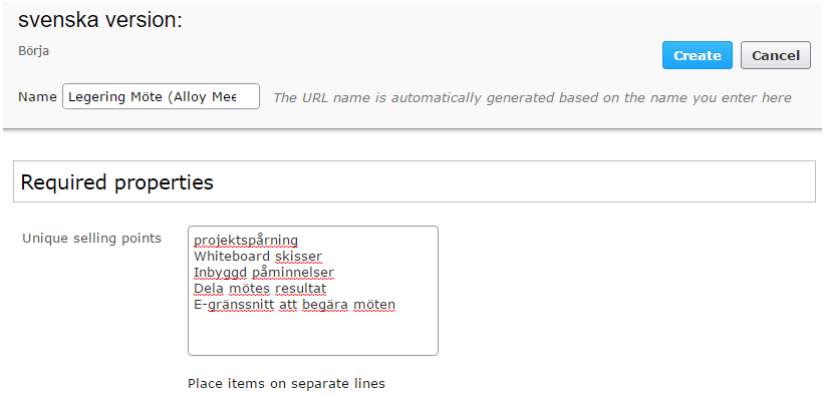
11. In the address bar, change **sv** to **da**, and note the 404 error due to strict language routing rules, as shown in the following screenshot:



12. Switch to **Edit** view.
13. Extract the folders and files in **cmsdevfun_exercisefiles.zip**.

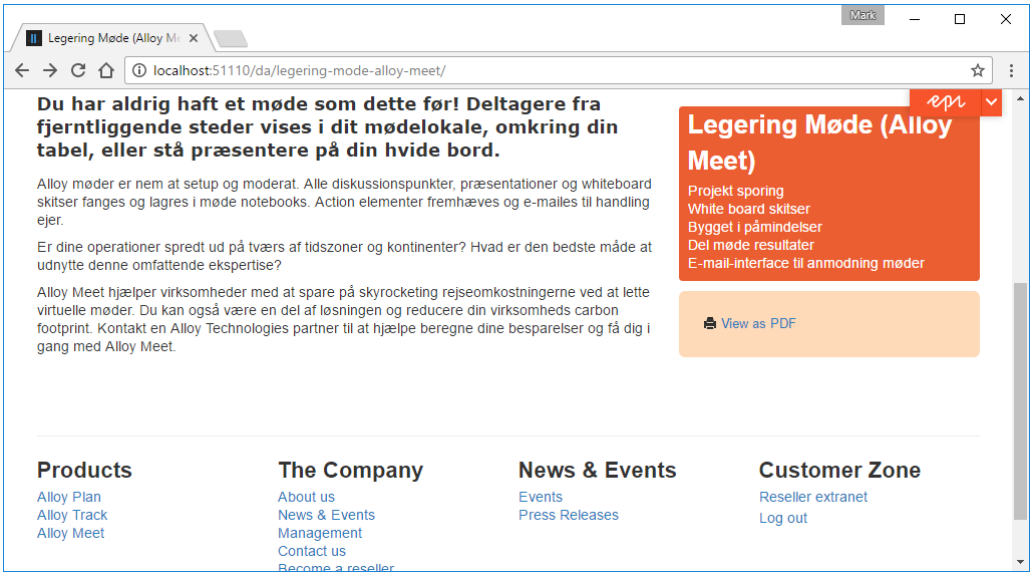
There is a file in **\cmsdevfun_exercisefiles\Module A\A5**, named **translations.pdf**, with translations for all the content in English, Swedish, and Danish.

14. Translate the **Alloy Meet** page into Swedish, as shown in the following screenshot:

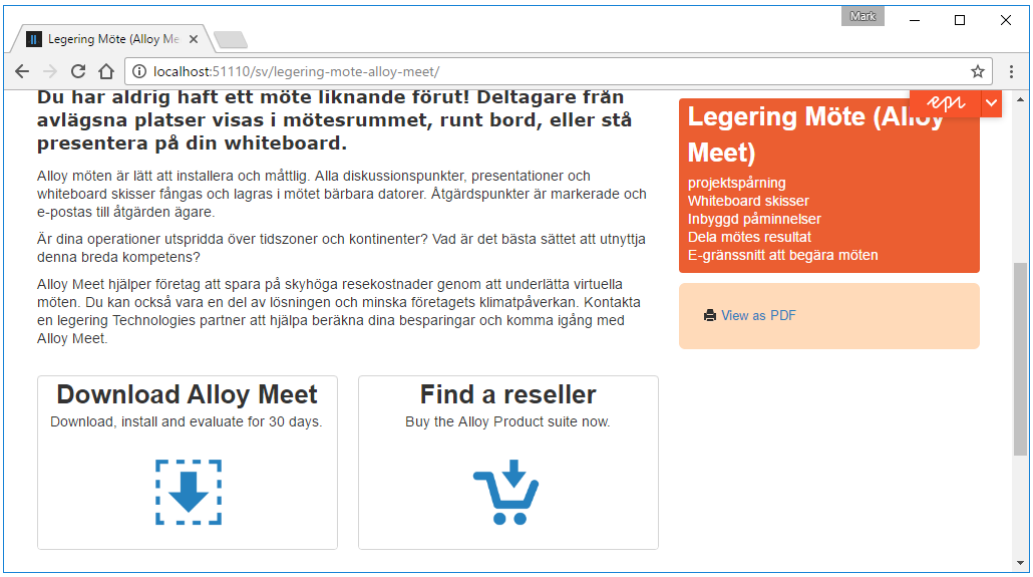


15. Translate the **Heading**, **Page description**, and **Main body** properties using the **translations.pdf** file, as shown in the following screenshot:

16. Publish the Swedish version of the Alloy Meet page.
17. Switch to the Danish site, translate and publish the Danish version of the Alloy Meet page.
18. View the site as a visitor. The Danish Alloy Meet page does not show blocks, because the blocks only exist in English, not Danish or Swedish, as shown in the following screenshot:



19. The Swedish version of the Alloy Meet page does show blocks (in English), as shown in the following screenshot:



Localizing content areas and blocks

ProductPage inherits from **StandardPage**, which has a **MainContentArea** that is not localized (because the property does not have `[CultureSpecific]` applied), so block references are shared by all language branches.

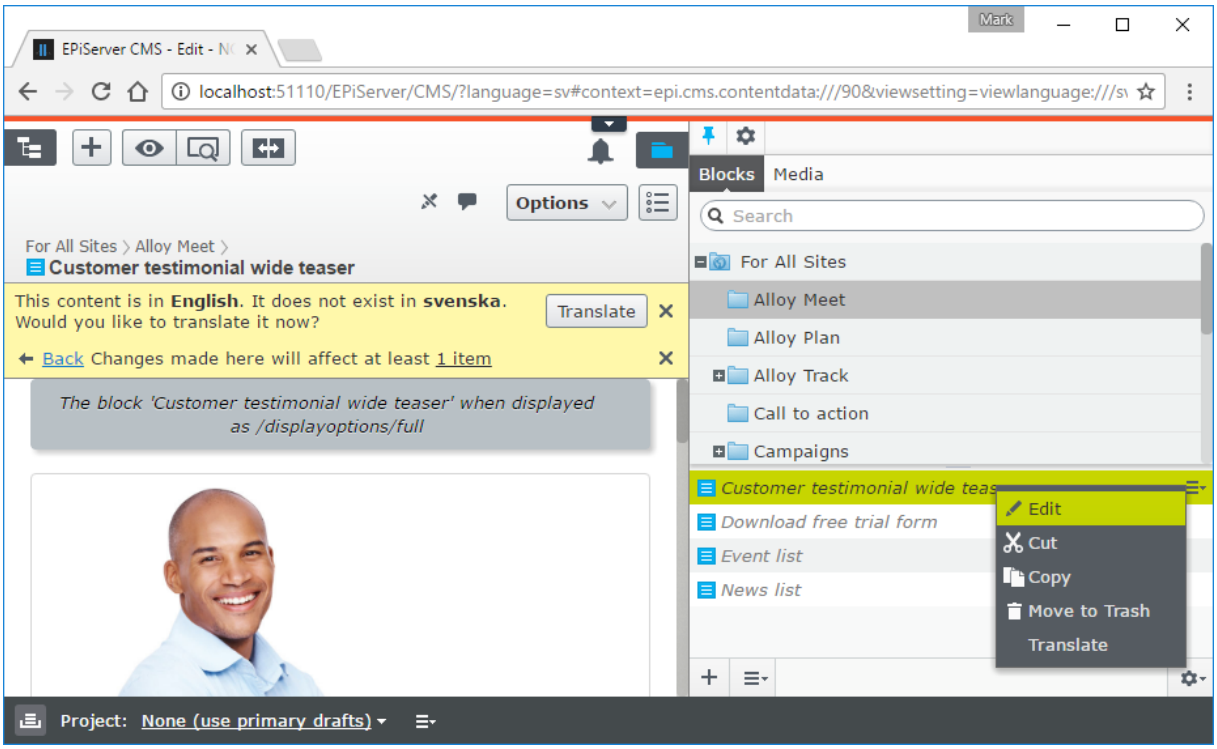
1. Open **~/Models/Pages/StandardPage.cs**. Note the MainContentArea is not culture specific, as shown in the following code snippet:

```
public class StandardPage : SitePageData
{
    [Display(
        GroupName = SystemTabNames.Content,
        Order = 310)]
    [CultureSpecific]
    public virtual XhtmlString MainBody { get; set; }

    [Display(
        GroupName = SystemTabNames.Content,
        Order = 320)]
    public virtual ContentArea MainContentArea { get; set; }
}
```

2. Open **~/Models/Blocks/TeaserBlock.cs**. Note the **Heading**, **Text**, and **Image** properties are `[CultureSpecific]`, as shown in the following code snippet:

```
[CultureSpecific]
[Required(AllowEmptyStrings = false)]
[Display(
    GroupName = SystemTabNames.Content,
    Order = 1)]
public virtual string Heading { get; set; }
```

3. Edit the site and switch to the Swedish version of the Alloy Meet page.
4. In the **Assets** pane, view **Blocks**, open the **Alloy Meet** folder, and edit the **Customer testimonial wide teaser** block, as shown in the following screenshot:

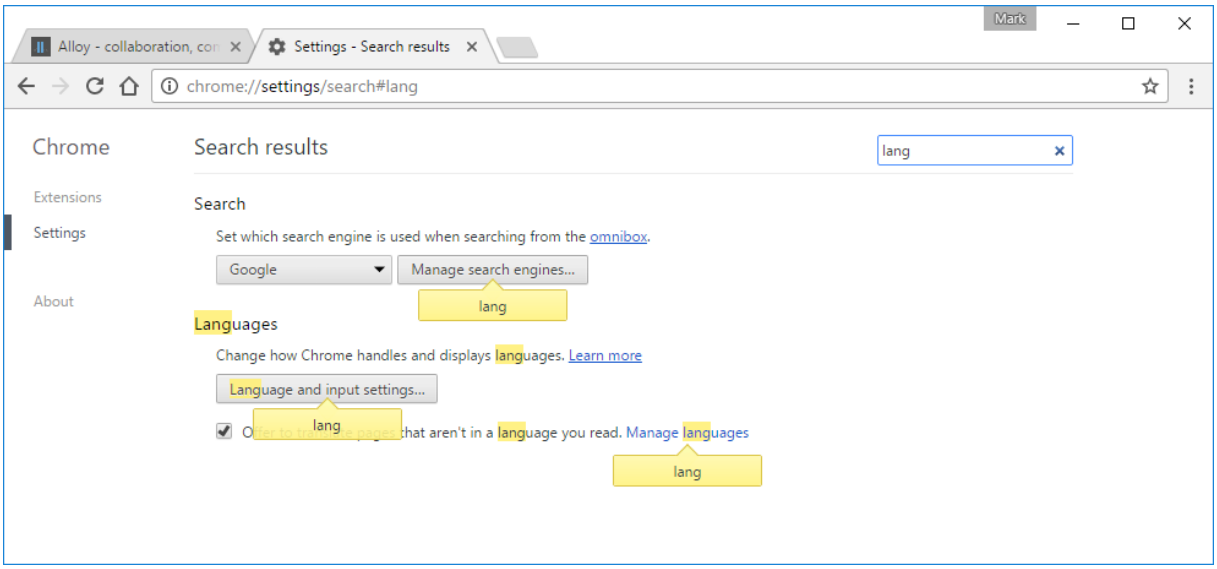


5. Translate the block into Swedish, as shown in the following screenshot:

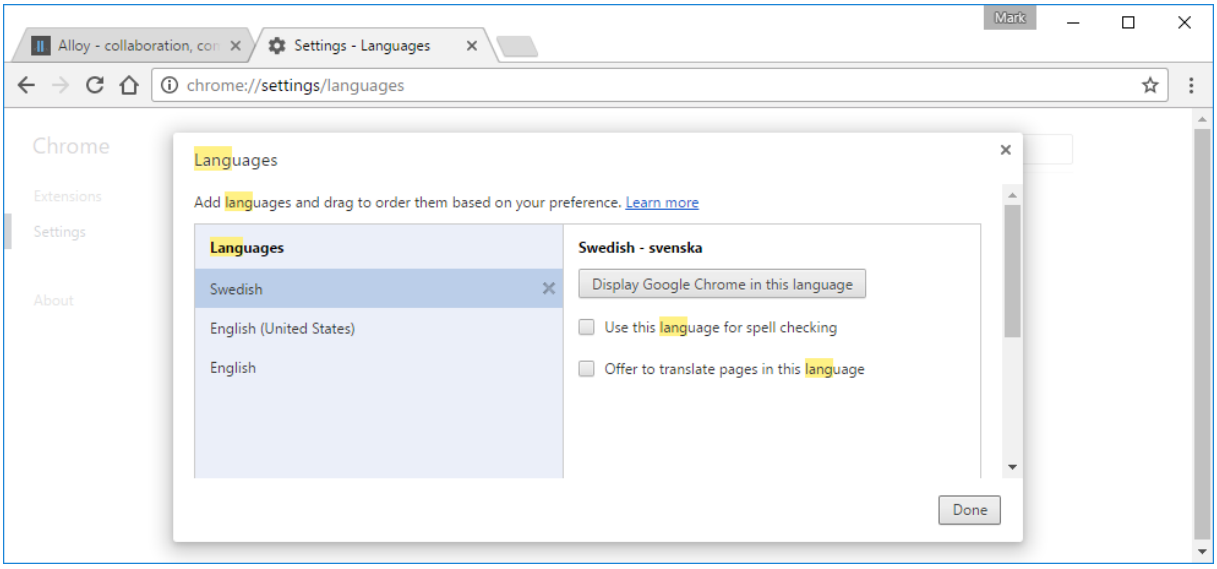
6. Publish the Swedish version of the block.
7. View the site as a visitor. Note the block is translated for Swedish page.

Configuring the site to detect language preference from the browser

1. In **Google Chrome**, choose **Settings**, and search for **lang**, as shown in the following screenshot:

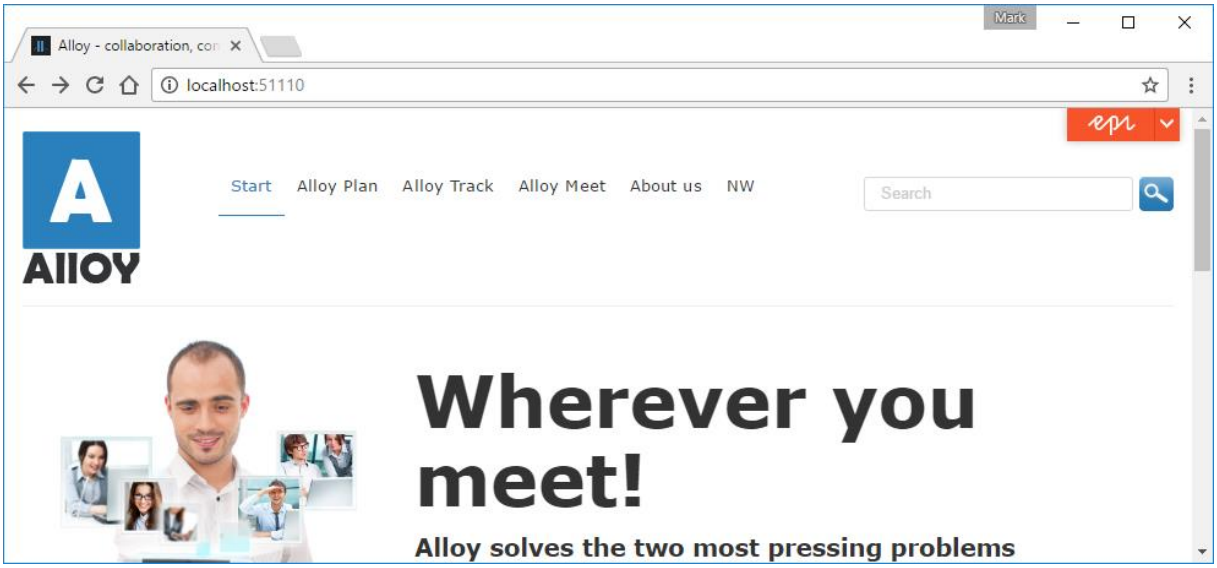


2. Click **Language and input settings...**, add **Swedish** and drag and drop it to the top of the list, as shown in the following screenshot:

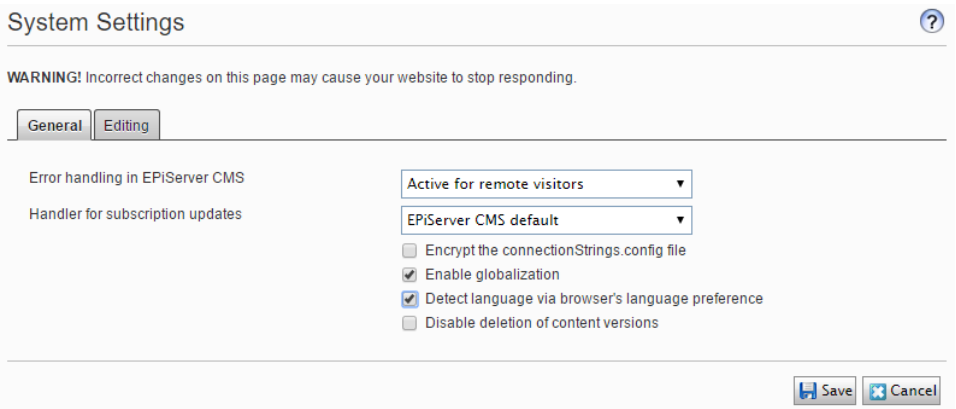


3. Click **Done**, and close the **Settings** tab.

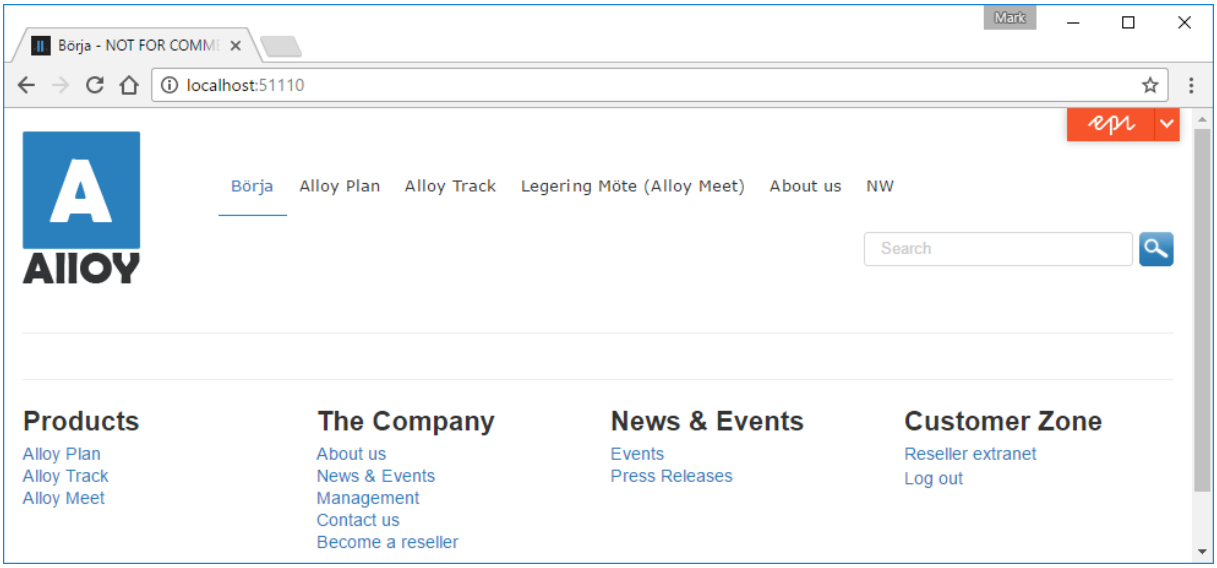
4. View the Start page as a visitor without a language code in the address bar, and the visitor sees the English version of the page, even though the browser is asking for Swedish, as shown in the following screenshot:



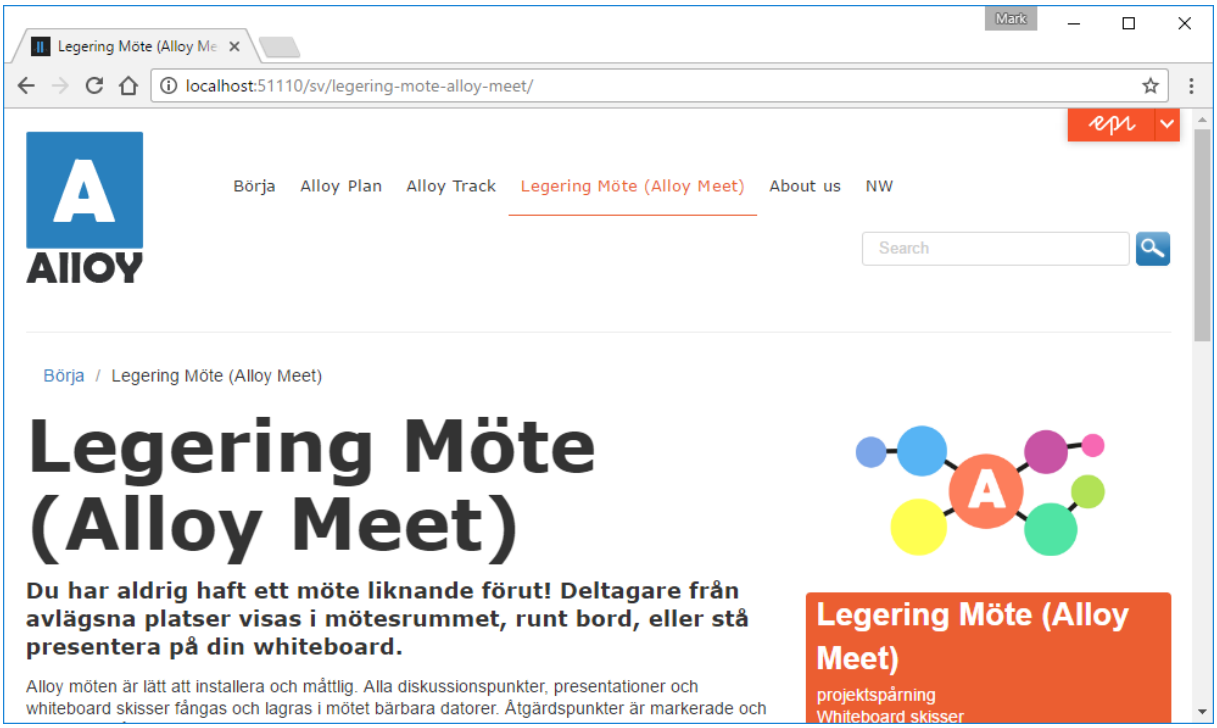
5. Log in and navigate to **CMS** | **Admin** | **Config** | **System Settings**.
6. Check the **Detect language via browser's language preference** box, and click **Save**, as shown in the following screenshot:



7. View the site as a visitor again, this time you see Swedish by default, as shown in the following screenshot:



8. All links from the Start page will include **/sv/** to request Swedish pages, as shown in the following screenshot:



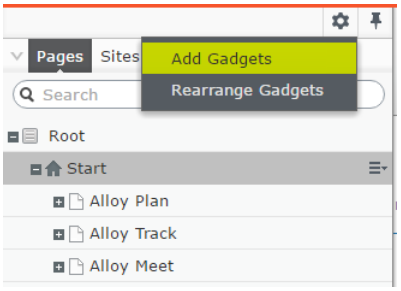
Automatically translating content using Episerver Languages

Episerver Languages add-on provides easy access to a single interface for managing multiple languages and translations of content, with a built-in feature for automated translation. No additional license fee is required for the add-on, and Microsoft Azure Translator Text API can translate 2 million characters per month for free. It is an example of an add-on that registers itself as a gadget that can then be added to either the **Navigation** or the **Assets** panes.

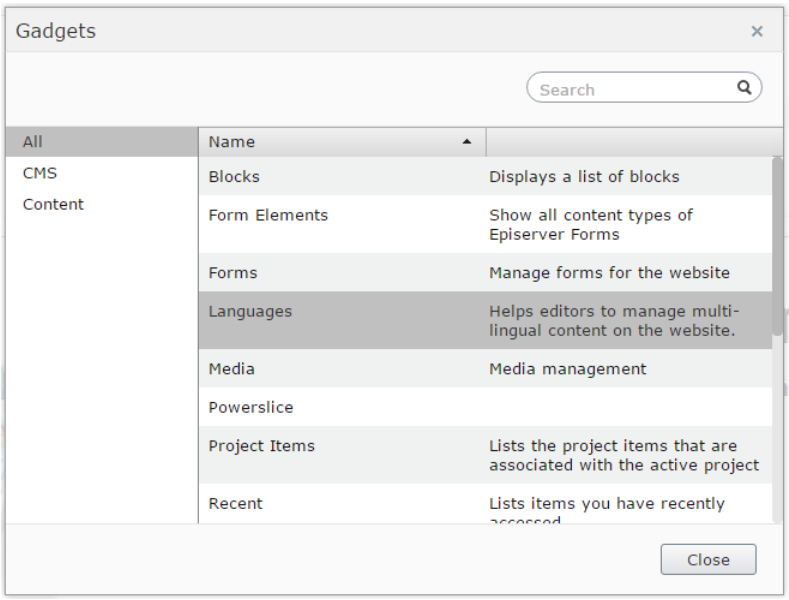
1. Open the solution with the **AlloyDemo** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
3. Enter the following command:

   ```
   Install-Package -ProjectName AlloyDemo EPiServer.Labs.LanguageManager
   ```

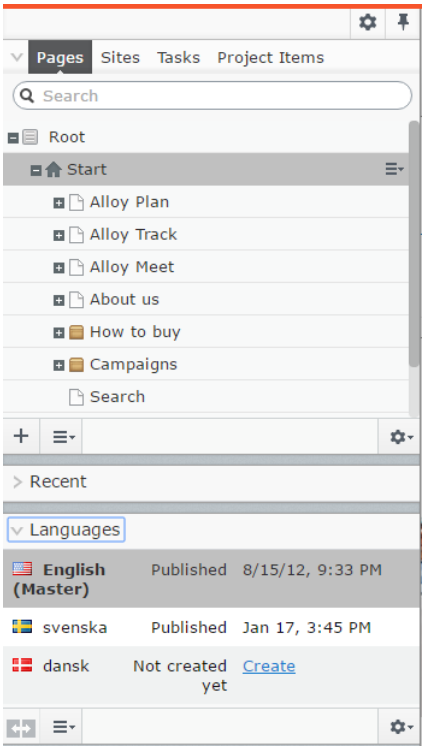
4. Start the site, and log in as **Admin**.
5. Navigate to **CMS** | **Edit**.
6. In the **Navigation** pane, on the **Settings** menu, click **Add Gadgets**, as shown in the following screenshot:



7. In the **Gadgets** picker, click **Languages**, as shown in the following screenshot:

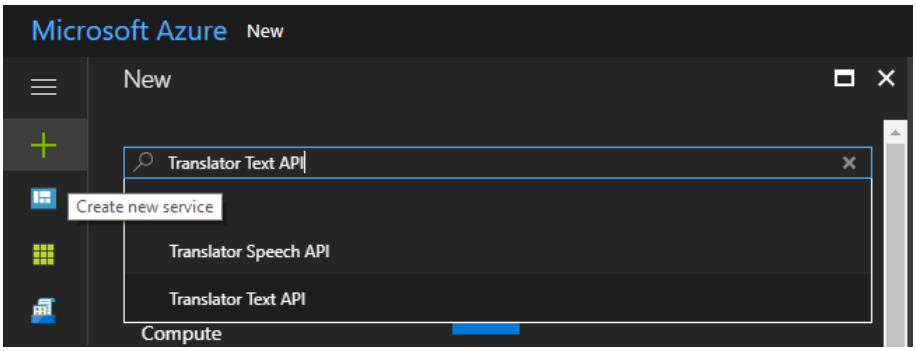


8. The **Languages** gadget will be added to the bottom of the **Navigation** pane, as shown in the following screenshot:

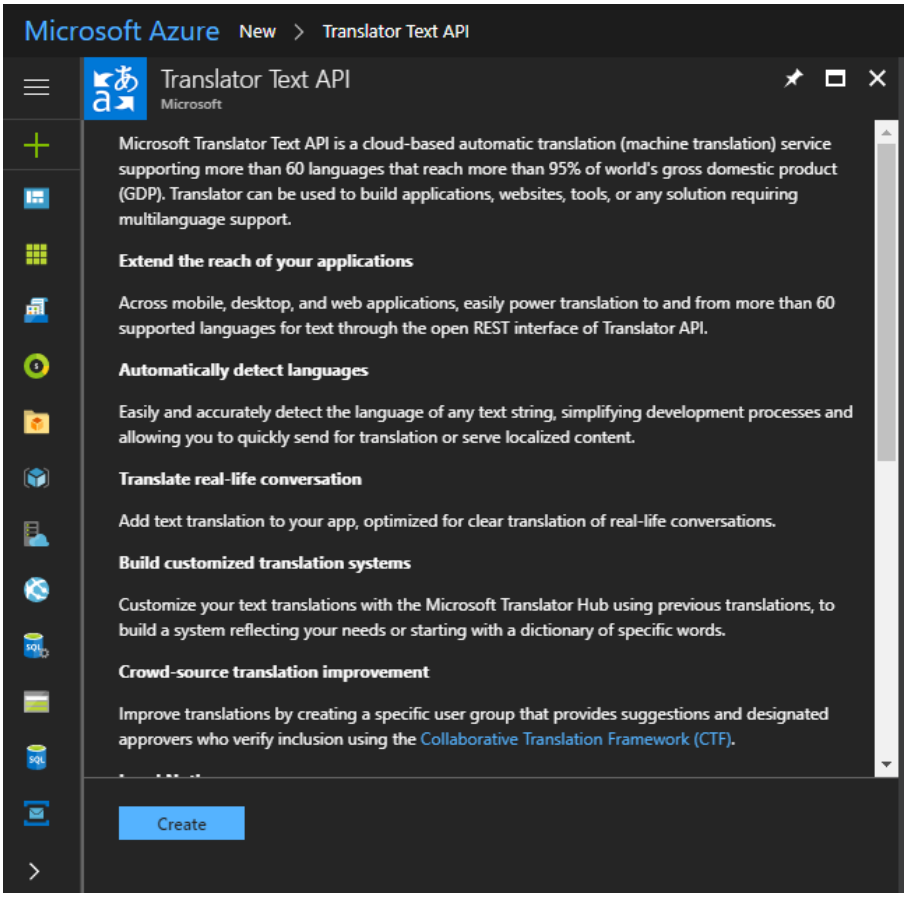


9. Sign in to the Azure portal with a Microsoft Azure account: `http://portal.azure.com`

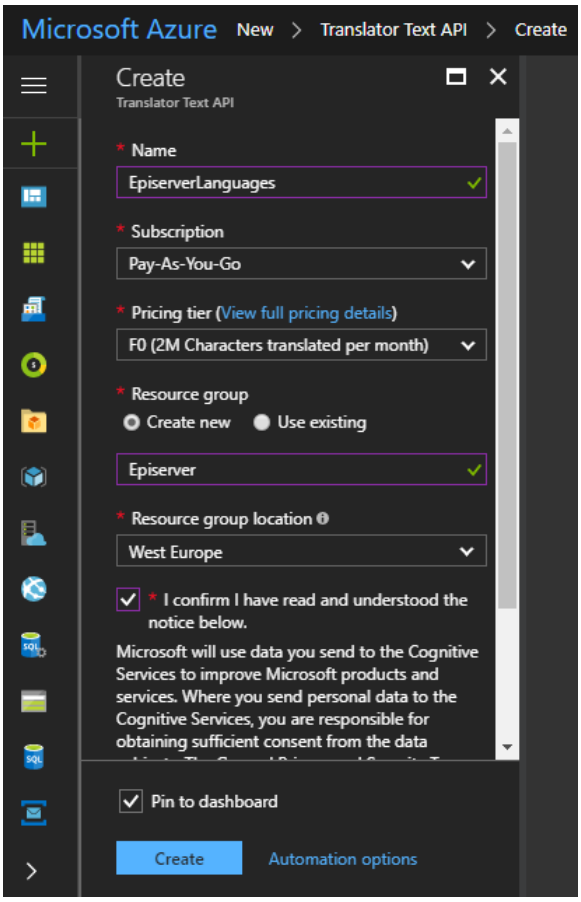
10. Click + to create a new service, and search for **Translator Text API**, as shown in the following screenshot:



11. Read the description, and then click Create, as shown in the following screenshot:

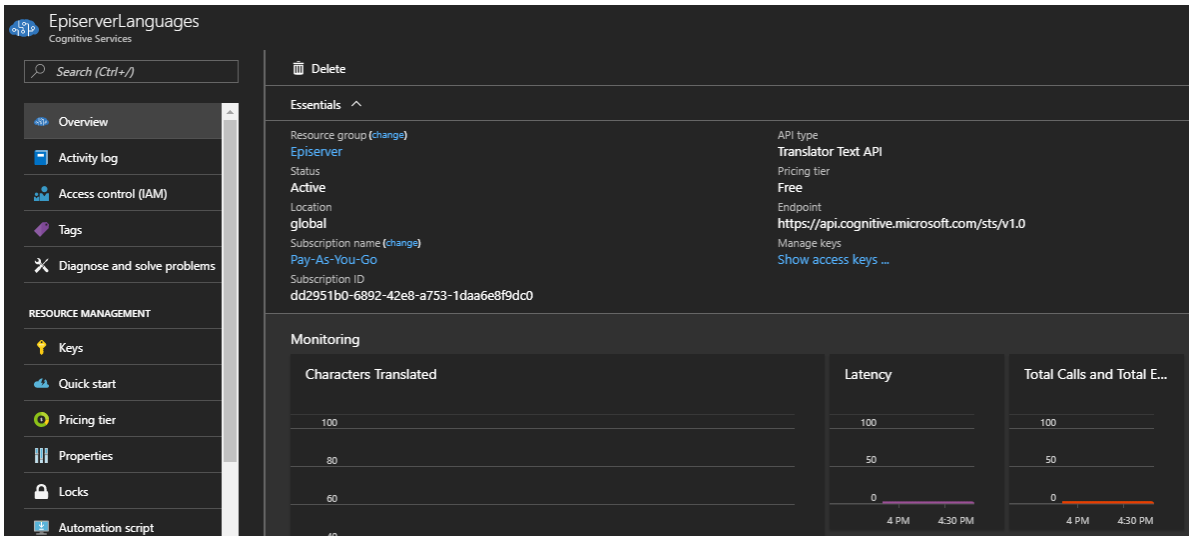


12. Set the following options:
    - Name: EpiserverLanguages
    - Pricing tier: F0 (2M Characters translated per month)



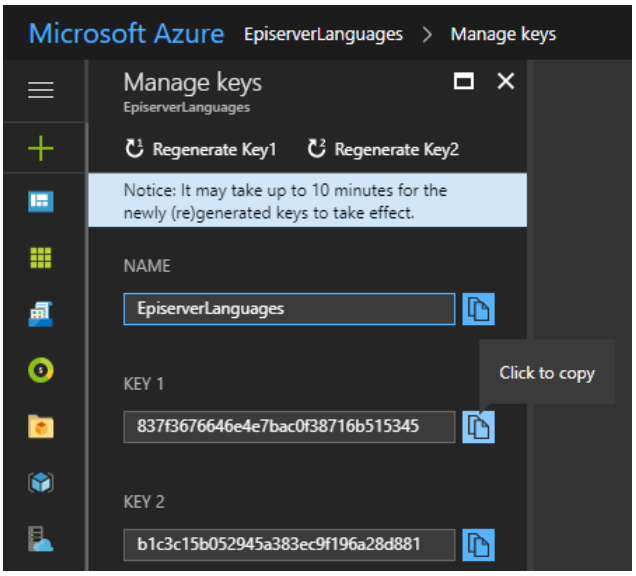
F0 is the free tier.

13. Click **Create**.
14. Navigate to the **EpiserverLanguages** service, as shown in the following screenshot:

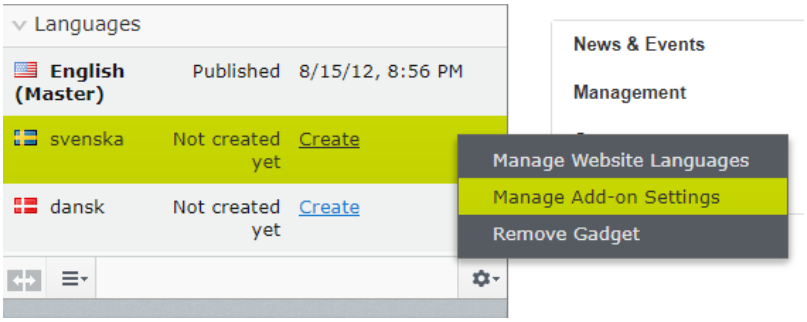


15. Click **Show access keys...**

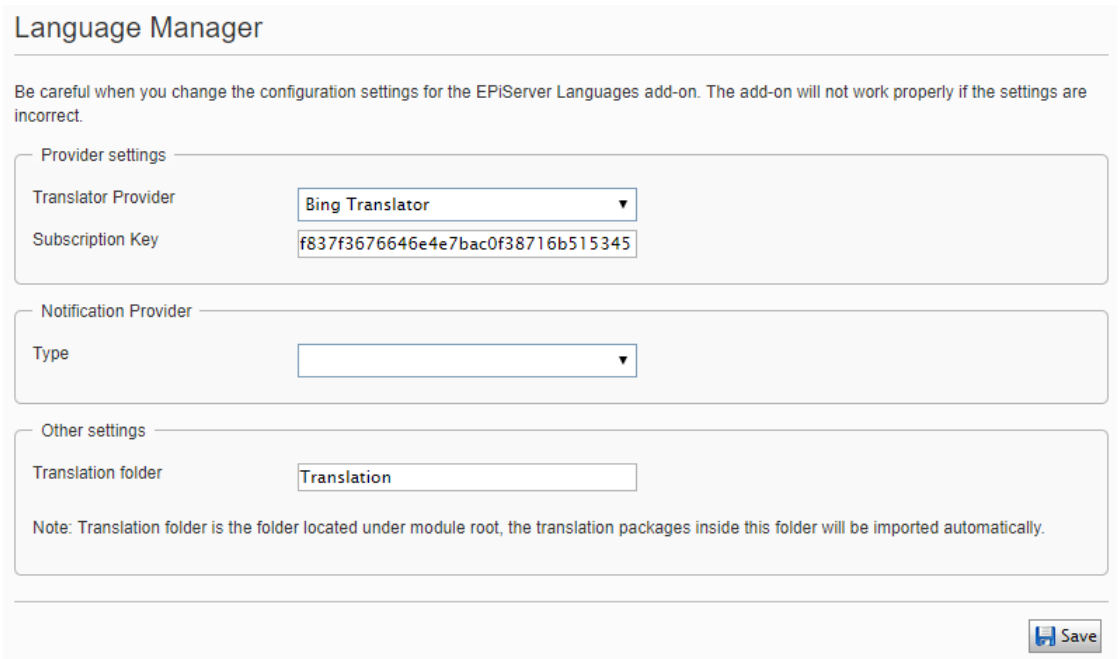
16. In the **Manage keys** blade, click the copy button next to one of the two keys, as shown in the following screenshot:



17. In the **Languages** gadget, click the Settings menu, and click **Manage Add-on Settings**, as shown in the following screenshot:

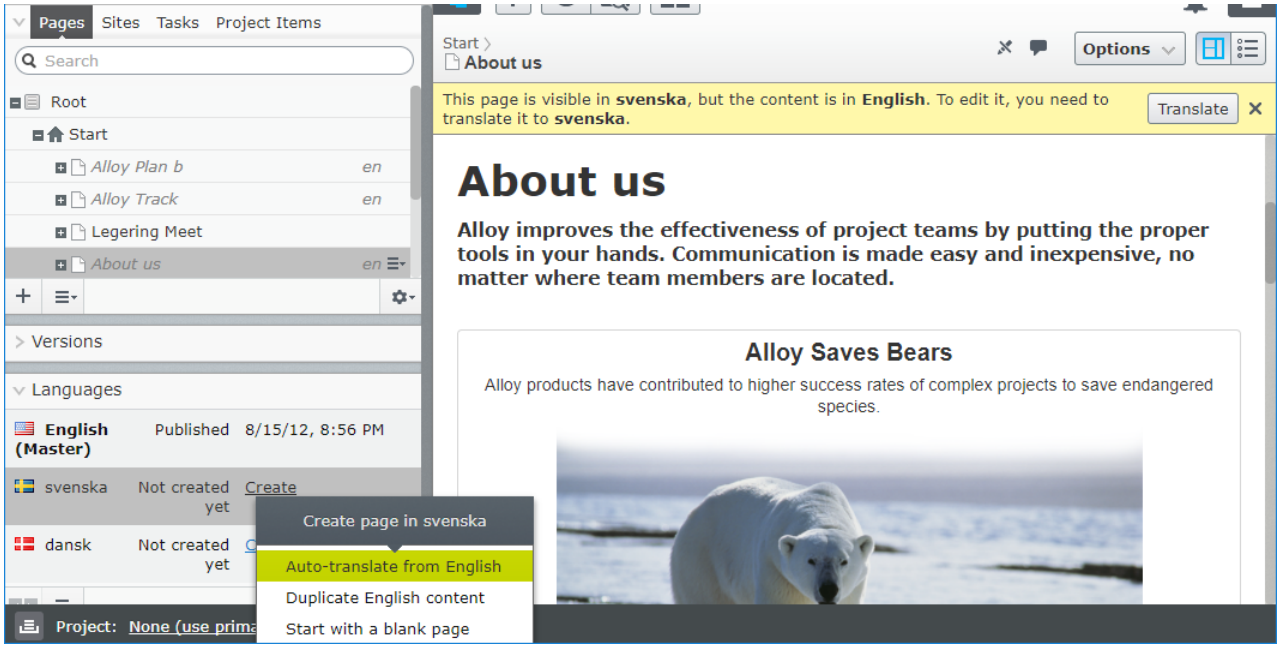


18. In Admin view, in **Language Manager**, set **Translator Provider** to **Bing Translate**, and paste the key for the **Subscription Key**, as shown in the following screenshot:

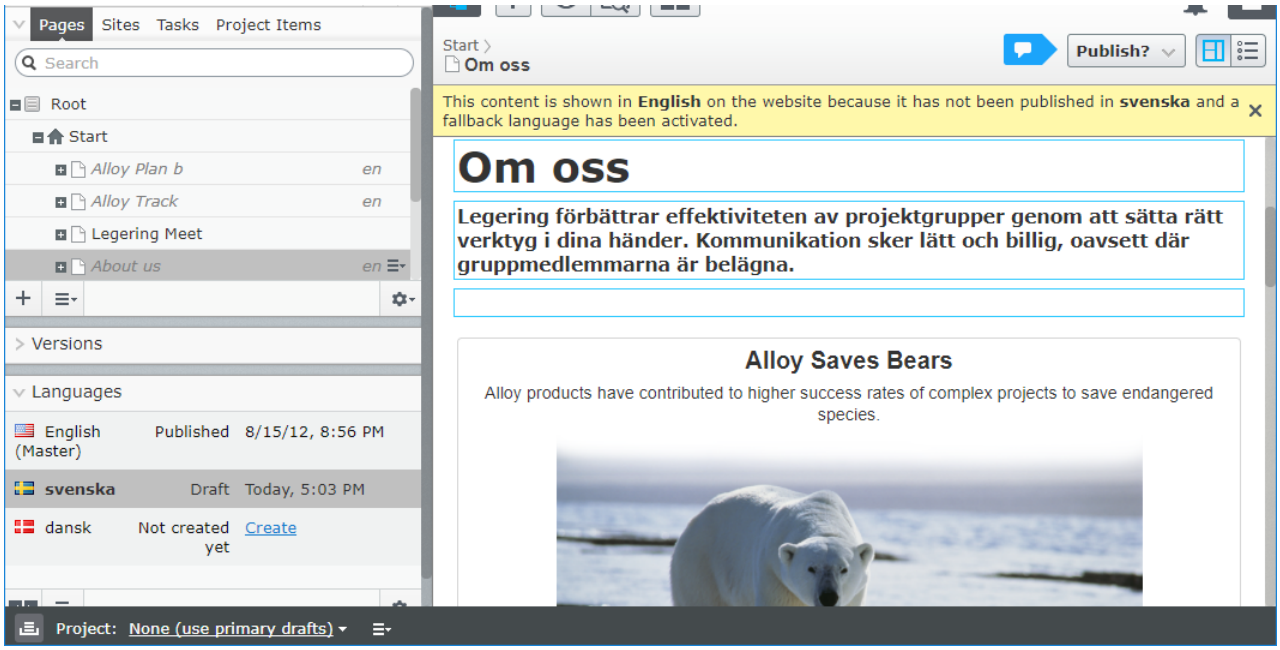


19. Click **Save**.

20. Navigate to Edit view, and select **About us** in the **Pages** tree.
21. In **Languages**, for **svenska**, click **Create**, and then **Auto-translate from English**, as shown in the following screenshot:

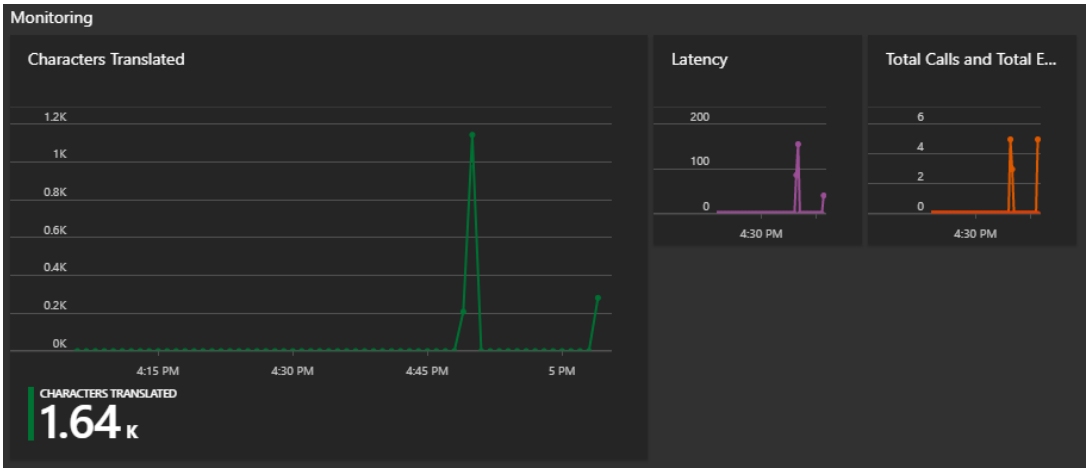


22. After a few seconds, any properties that are culture specific will be translated automatically, as shown in the following screenshot:



23. Publish the Swedish page.

24. In Azure portal, note the **Monitoring** section, as shown in the following screenshot:

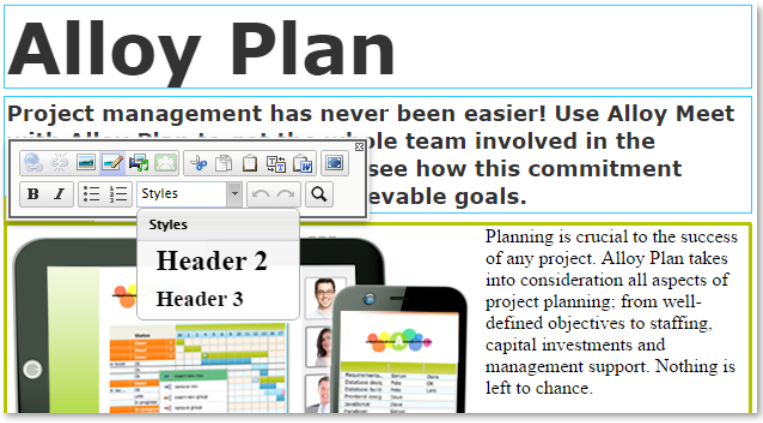


25. Review the **Episerver Languages User Guide** online:

    `http://webhelp.episerver.com/latest/addons/languages.htm`

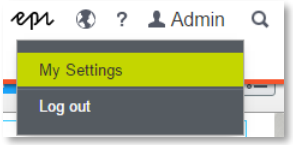
Localizing the TinyMCE toolbar styles drop-down list

1. Open the solution with the **AlloyDemo** project.
2. Start the site, and log in as **Admin**.
3. Edit the **Alloy Plan** product page, and click inside the **Main body** property to see the TinyMCE toolbar.
4. Click the **Styles** drop-down list box, as shown in the following screenshot:



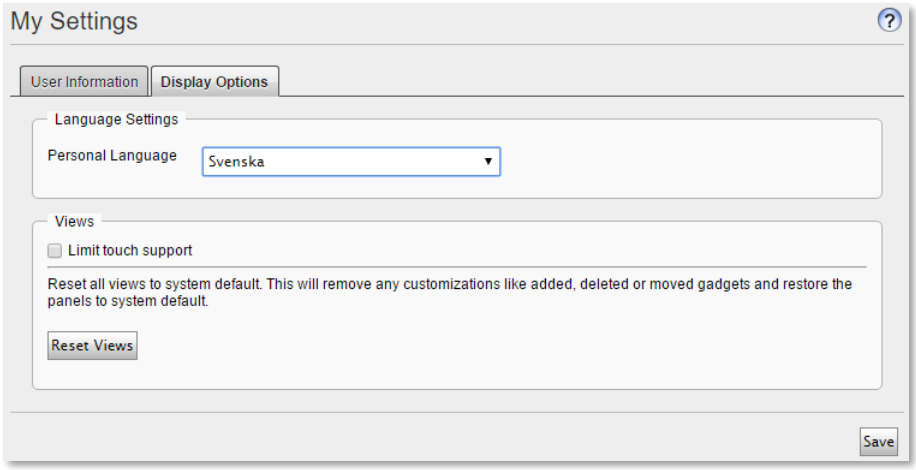
Switching to Swedish

1. In the **Global** menu, click **Admin**, and then **My Settings**, as shown:

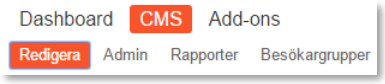


2. Click the **Display Options** tab. The Admin user has their **Personal Language** set to **Use system language**, which on my laptop, is English.

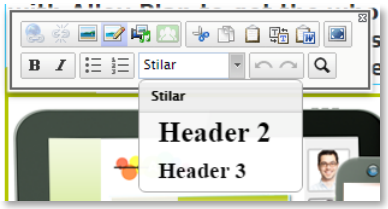
3. Change the language to **Svenska** and click **Save**, as shown in the following screenshot:



4. After a few seconds, the page refreshes, and shows all labels in Swedish.
5. Switch to Edit view by showing the Global menu, and clicking **CMS** | **Redigera**.



6. Click in the **Main body** again and note that the label **Styles** has been localized into Swedish as **Stilar**, but the two choices, **Heading 2** and **Heading 3**, have not.



7. Close the browser.

Localizing styles

1. In **Solution Explorer**, open **~/Static/css/editor.css**. This is the file that is used by TinyMCE to configure its options. The path to this file can be set through the Admin view or by editing the Web.config `<episerver><applicationSettings uiEditorCssPaths="..." />` attribute.
2. Add two additional menu names, **Introduction** and **Alert Box**, and add a menu title, **Headings**, as shown in the following CSS:

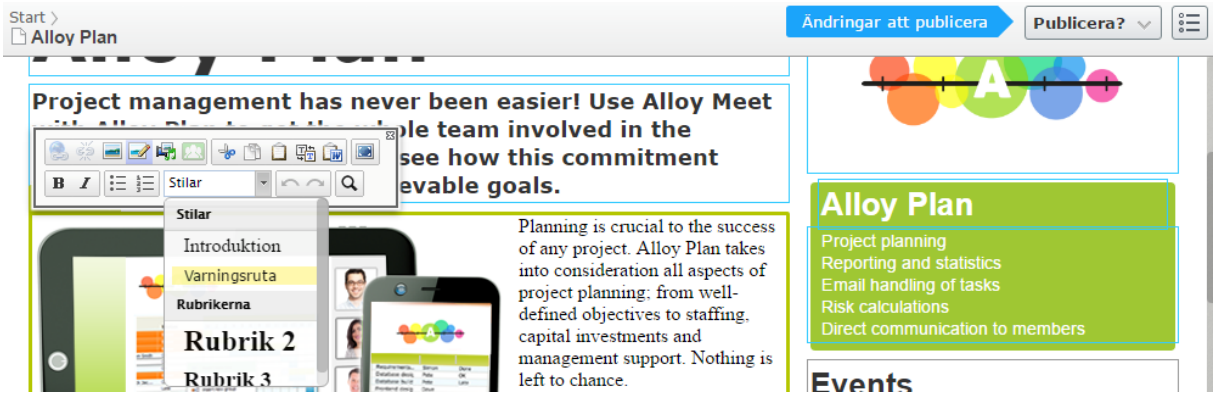
```
p.introduction {
    ChangeElementType: true; /* allows change of element, i.e. h1 to p*/
    EditMenuName: Introduction;
}

.alert-info {
    EditMenuName: Alert Box;
}

h2 {
    EditMenuTitle: Headings;
    EditMenuName: Header 2;
}

h3 {
    EditMenuName: Header 3;
```

```
}

.alert-info {
    background-color: #FFF8AA;
    border-color: #858585;
    color: #000000;
    font-family: Verdana;
    font-size: 12px;
}

.header.dim {
    margin: 2% 0;
    opacity: 0.3;
}
```

3. In **~/Resources/LanguageFiles**, add a new XML file named **TinyMCE.xml**.
4. Modify its content as shown in the following markup, and note that where an edit menu name had a space in the CSS, it must be replaced with an underscore in the matching XML element name:

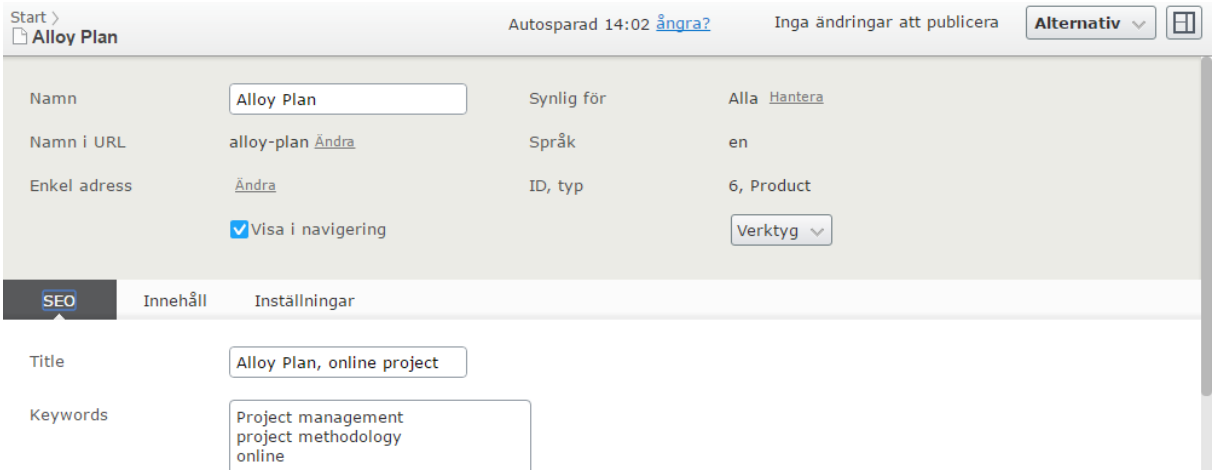
```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <editorstyleoptions>
      <introduction>Introduction</introduction>
      <alert_box>Alert Box</alert_box>
      <headings>Headings</headings>
      <header_2>Heading 2</header_2>
      <header_3>Heading 3</header_3>
    </editorstyleoptions>
  </language>
  <language name="Svenska" id="sv">
    <editorstyleoptions>
      <introduction>Introduktion</introduction>
      <alert_box>Varningsruta</alert_box>
      <headings>Rubrikerna</headings>
      <header_2>Rubrik 2</header_2>
      <header_3>Rubrik 3</header_3>
    </editorstyleoptions>
  </language>
</languages>
```

5. Start the website, and log in as **Admin**.
6. Edit **Alloy Plan**, and click the styles drop-down, as shown in the following screenshot:



Although the names of the styles in the drop-down list, the Publish button and the blue information message have been localized into Swedish, the content types and their properties have not.

If the editor has chosen Swedish as their personal language and they edit the product page in **All Properties** view, Episerver localizes as much of the user experience labelling as possible, but it cannot automatically localize things like the page type name, custom group (tab) names like **SEO**, and custom property names like **Title** and **Keywords**, as shown in the following screenshot:



Defining the localization text values

1. In **~/Resources/LanguageFiles**, copy and paste **ContentTypeNames.xml** by pressing *Ctrl* + *C*, then *Ctrl* + *V*.
2. Rename the copied file to **ContentTypesNames-sv.xml**.
3. Find the **<language>** element and modify it as follows:

```
<language name="Svenska" id="sv">
```

4. Find the **<productpage>** element and modify it as follows:

```
<productpage>
  <name>Produkt</name>
  <description>Används för att presentera en specifik produkt</description>
</productpage>
```

5. In **~/Resources/LanguageFiles**, copy and paste **PropertyNames.xml**.
6. Rename the copy to **PropertyNames-sv.xml**.
7. Find the **<language>** element and modify it as follows:

```
<language name="Svenska" id="sv">
```

8. Find the **<meta...>** elements and modify them as follows:

```
<metadescription>
  <caption>Sidbeskrivning</caption>
  <help>Används som meta beskrivning och allmänt som en ingress</help>
</metadescription>
<metakeywords>
  <caption>Nyckelord</caption>
</metakeywords>
<metatitle>
  <caption>Titel</caption>
</metatitle>
```

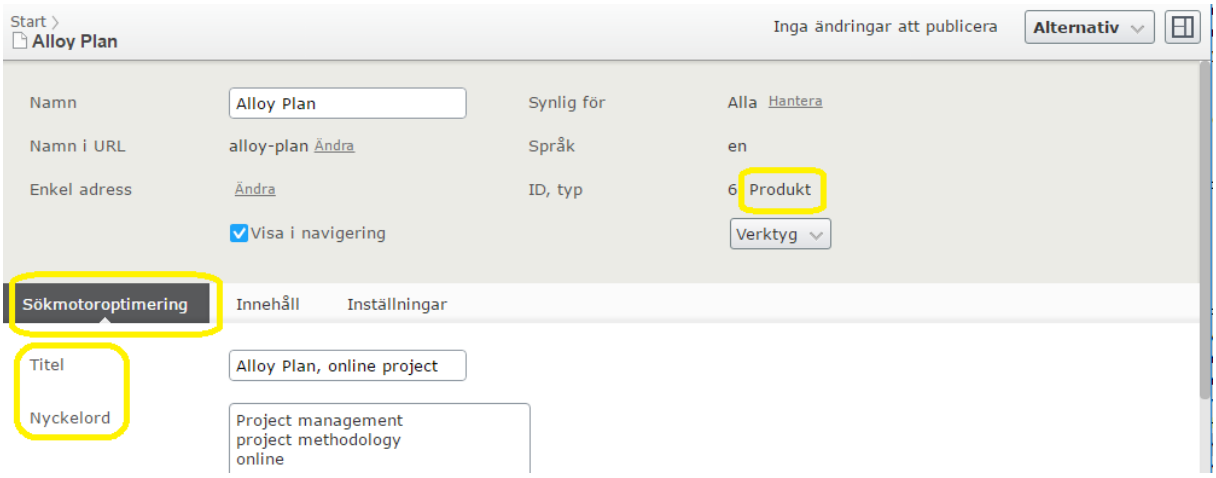
9. In **~/Resources/LanguageFiles**, copy and paste **GroupNames.xml**.
10. Rename the copy to **GroupNames-sv.xml**. Modify it as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
```

```
    <language name="Svenska" id="sv">
      <headings>
        <heading name="Contact">
          <description>Kontakta</description>
        </heading>
        <heading name="Default">
          <description>Standard</description>
        </heading>
        <heading name="News">
          <description>Nyheter</description>
        </heading>
        <heading name="Products">
          <description>Produkter</description>
        </heading>
        <heading name="MetaData">
          <description>Sökmotoroptimering</description>
        </heading>
        <heading name="SiteSettings">
          <description>Webbplatsinställningar</description>
        </heading>
        <heading name="Specialized">
          <description>Specialiserad</description>
        </heading>
      </headings>
    </language>
  </languages>
```

Viewing the localizations

1. Start the site, and log in as **Admin**.
2. Edit the **Alloy Plan** page, and switch to **All Properties** view.
3. Note the parts of the user interface that are now localized into Swedish that weren't before: the page type, the group name/tab, and the property names, as highlighted in the following screenshot:



Optional task: Localizing all content types and property names

Use Google (or Bing) translation to localize more of the content types and their property names.

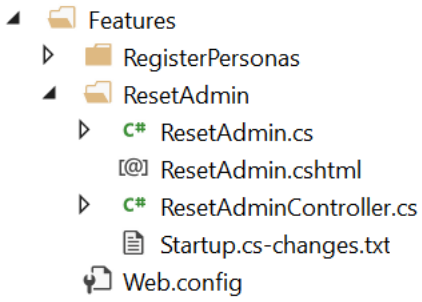
Exercise A6 – Optional: Resetting the Admin account

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

Prerequisites: complete Exercise A1.

Adding the reset admin feature

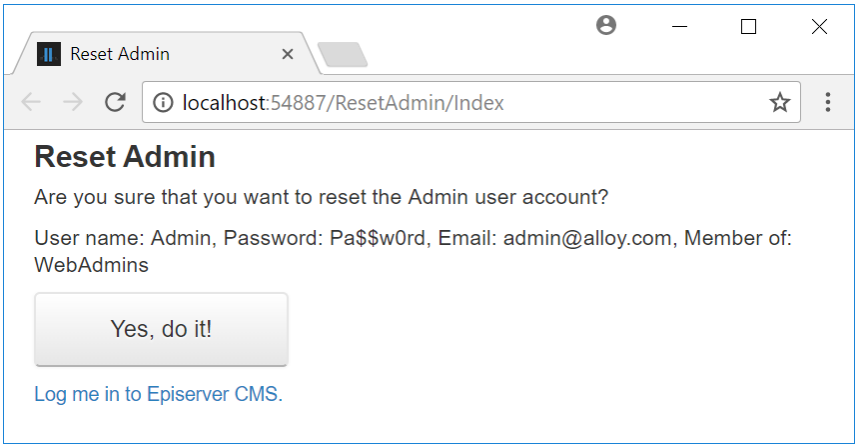
1. If you haven't done so already, extract the folders and files in **cmsdevfun_exercisefiles.zip**.
2. Drag and drop the **\cmsdevfun-exercisefiles\Module A\A6\Features** folder into the **AlloyDemo** project.
3. Expand the **Features** folder and review the files included, as shown in the following screenshot:
    a. **ResetAdmin.cs**: a static class to enable the feature.
    b. **ResetAdmin.cshtml**: a Razor file for the user interface of the feature.
    c. **ResetAdminController.cs**: a controller that performs the work of the feature.



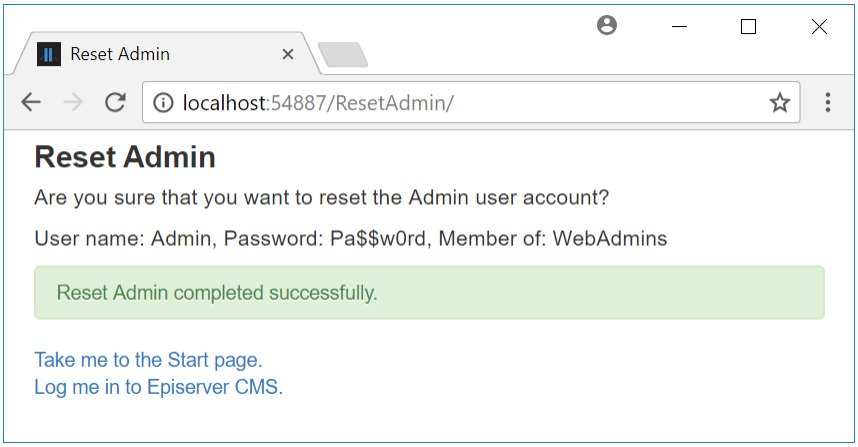
4. Open **ResetAdminController.cs** file and modify the username and password if you want.
5. Open the **Startup.cs-changes.txt** file and copy and paste the statements into the appropriate place in the **Startup.cs** file.

Resetting the admin account

1. Start the **AlloyDemo** 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 B – Defining Content Types

Goal

The overall goal of the exercises in this module is to implement typical examples of content types with templates and layouts while following good practice. You will:

1. Set up an **Empty** Episerver website and define a **Start** page type with a page template.
2. Define media types to handle generic files, image files, and SVG files.
3. Create shared layouts for page templates, follow good practice to define a base page type and a base page controller, define a page view model and use it in layouts, views, and controllers.
4. Define a **Standard** page with a **MainBody** property and an alternative layout, a **Product** page with **UniqueSellingPoints** property and a nested layout, and add a navigation menu.

Exercise B1 – Setting up the AlloyTraining website

In this exercise, you will set up an Empty Episerver website and update it to use the latest version of Episerver CMS. You will:

- Add Bootstrap files.
- Add helper classes used in later exercises to save you time.
- Define a Start page type.
- Create a page template for the Start page type.
- Optional: localize the Start page type for editors who speak Swedish but not English

Prerequisites: Microsoft Visual Studio 2015 or later with Episerver CMS Visual Studio Extension.

Creating AlloyTraining from the Empty Episerver Web Site project template

You MUST name your project **AlloyTraining**. The reason is that the solution classes have been written assuming that project name and so all the namespaces will use it, e.g., **AlloyTraining.Models.Pages.StartPage**, and so on.

1. In Visual Studio, navigate to **File** | **Add** | **New Project...**

You can use a new solution if you prefer.

2. In the left section, navigate to **Installed** | **Templates** | **Visual C#** | **Episerver**.
3. In the middle section, select **Episerver Web Site** project, and enter the following options:
   - Target: **.NET Framework 4.6.1** or later compatible version.
   - Name: **AlloyTraining**
4. Click **OK**.
5. Choose the **Empty** template and click **OK**.
6. In **Solution Explorer**, right-click **Solution 'Training' (2 projects)** and click **Set StartUp Projects**.
7. Click **Current selection**, and then click **OK**.

You will now be able to quickly switch between running the two websites just by selecting a project in Solution Explorer and pressing *Ctrl + F5*.

Installing some add-ons and updating the Episerver NuGet packages and database

1. In Visual Studio, navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. In **Package Manager Console**, set these two options:
    a. Package source: **All**
    b. Default project: **AlloyTraining**
3. Enter the following command to install Episerver Search indexing service:

```
Install-Package -ProjectName AlloyTraining EPiServer.Search
```

It is worth installing **Episerver Search** now so that you can use the Global search and search boxes in the **Navigation** and **Assets** panes as an editor and administrator.

4. Enter the following command to install Episerver Search integration with CMS:

```
Install-Package -ProjectName AlloyTraining EPiServer.Search.Cms
```

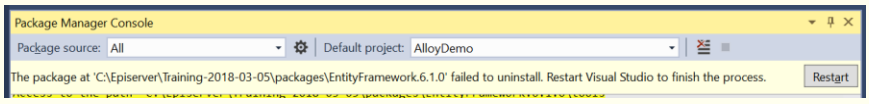
5. Enter the following command to install Episerver Forms:

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

6. Enter the following command to update all packages using restrictions defined in packages.config:

```
Update-Package -ProjectName AlloyTraining -ToHighestMinor
```

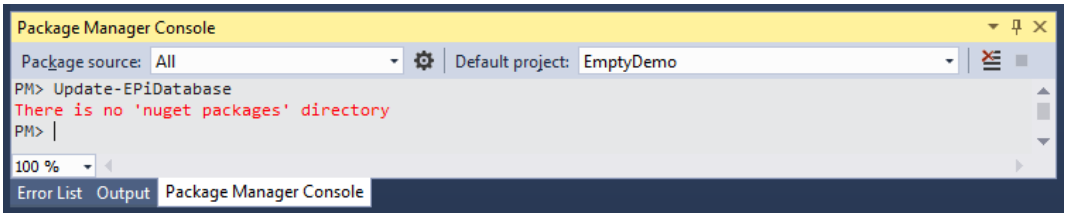
You might need to restart Visual Studio a couple of times to update Entity Framework, as shown in the following screenshot:



7. In **Package Manager Console**, set these two options:
    a. Package source: **All**
    b. Default project: **AlloyTraining**
8. Enter the following command to update the CMS database:

```
Update-EPiDatabase
```

9. If you see an error message, as shown in the following screenshot, then close Visual Studio, restart, reopen the solution, select the correct **Default project**, and try again:



Reviewing authentication and authorization in an Episerver CMS Empty site

1. In the **AlloyTraining** project, open **~/Web.config**.
2. Find the `<authentication>` element, as shown in the following configuration:

```
<authentication mode="Forms">
  <forms name=".EPiServerLogin"
         loginUrl="Util/login.aspx"
         timeout="120"
```

```
            defaultUrl="~/" />
    </authentication>
```

In an empty Episerver site, authentication uses **Forms**, aka **ASP.NET Membership**.

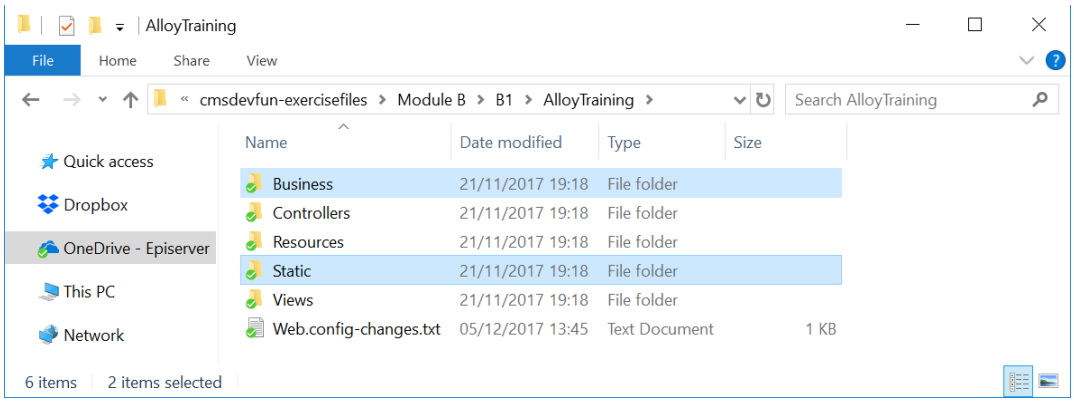
3. Find the `<membership>` and `<rolemanager>` elements, as partially shown in the following configuration:

```
<membership defaultProvider="MultiplexingMembershipProvider" ...>
  <providers>
    <clear />
    <add name="MultiplexingMembershipProvider" ...
         provider1="SqlServerMembershipProvider"
         provider2="WindowsMembershipProvider" />
    <add name="WindowsMembershipProvider" ... />
    <add name="SqlServerMembershipProvider" ...
         connectionStringName="EPiServerDB" ...
         minRequiredPasswordLength="6"
         minRequiredNonalphanumericCharacters="0" ... />
  </providers>
</membership>
<roleManager enabled="true"
             defaultProvider="MultiplexingRoleProvider"
             cacheRolesInCookie="true">
  <providers>
    <clear />
    <add name="MultiplexingRoleProvider" ...
         provider1="SqlServerRoleProvider"
         provider2="WindowsRoleProvider" ... />
    <add name="WindowsRoleProvider" ... />
    <add name="SqlServerRoleProvider" ...
         connectionStringName="EPiServerDB" applicationName="/" />
  </providers>
</roleManager>
```

In the Empty Episerver site project template, membership and role providers are configured to use either the Windows or SQL Server providers, with a multiplexing provider configured to use SQL Server first. This allows new roles and users to be created. You can remove these and add alternative providers, for example, Microsoft Azure Active Directory.

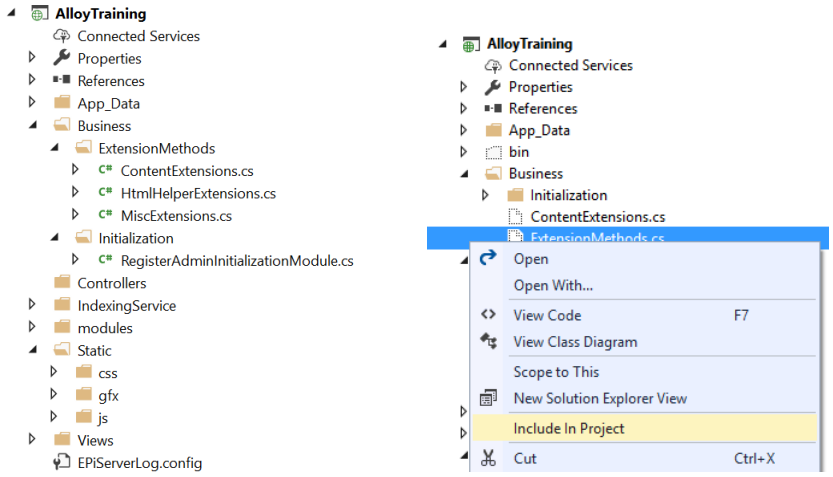
Adding Bootstrap and Business logic

1. If you haven't already, then extract the folders and files in **cmsdevfun_exercisefiles.zip**.
2. Drag and drop or copy the **Business** and **Static** folders from **\cmsdevfun-exercisefiles\Module B\B1** to the root of the **AlloyTraining** project, as shown in the following screenshot:



Make sure you drag and drop to the **AlloyTraining** project, NOT AlloyDemo!

3. In **Solution Explorer**, expand the **Business** and **Static** folders, as shown in the following screenshot:



4. Note the following folders and files were added:
   - **Extension methods** for use later in the exercises to generate menus and so on.
   - **Initialization** module for registering an Admin user account if using SQL Server provider.
   - **Static** application files for stylesheets, JavaScript libraries, and graphics.

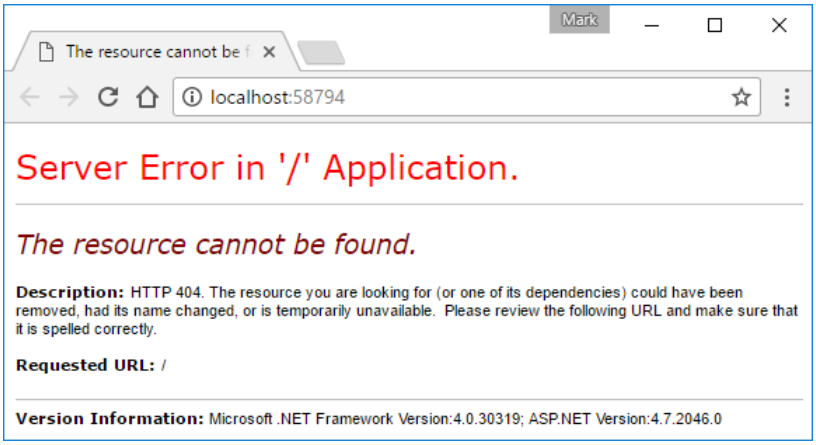
If you are using Visual Studio 2015, ensure that the added files are part of the project by right-clicking the files and selecting **Include In Project**, as shown in the preceding screenshot (on the right), otherwise you will get compile errors due to missing classes.

5. Click **Build** | **Build Solution** to ensure the website compiles.

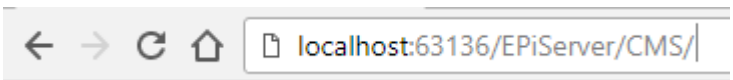
Testing the Empty Episerver website

1. In **Solution Explorer**, click **AlloyTraining**.
2. Start the website by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.
3. You will see a **HTTP 404** error message, as shown in the following screenshot:

This is expected, because the site is empty, and does not yet have a start page.



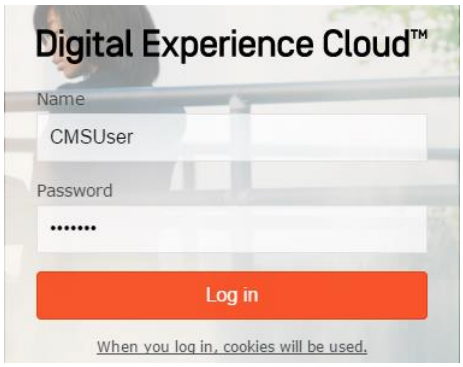
4. Enter **/EPiServer/CMS/** at the end of the address box, as shown in the following screenshot:



The **Episerver Web Site - Empty** project template configures ASP.NET Membership providers to allow a member of the Windows group named **Administrators** to be recognized as an administrator of the CMS.

5. Log in using a *local* Windows account that is a member of the *local* Windows group named **Administrators**. If you don't have a local Windows account, move on to the next task.

In an Episerver training room, you can enter the following details for a local Windows account, as shown in the following screenshot: Name: **CMSUser**, Password: **CMSUser**

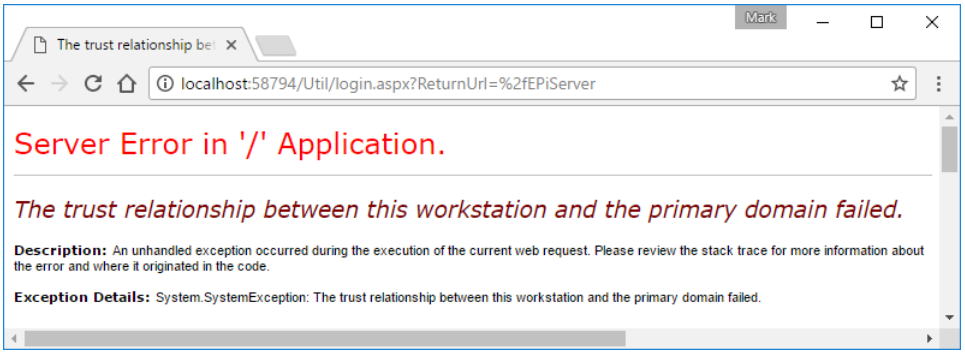


Yes, even an on-premise deployed website is part of Episerver's Digital Experience Cloud™. 😉

If you can't authenticate with a Windows account

You do NOT need to do this task if you logged in with **CMSUser** or another Windows adminstrator account! Skip ahead to task **Defining a class of string constants for grouping content types and a Start page type** on page **68**.

If you do not have a *local* Windows account that is a member of the *local* Windows group named **Administrators** or if you see the following error message, then you can add a setting to Web.config to create a SQL-stored account instead:



WARNING! The password of a SQL-stored account cannot be reset, so following these steps will **delete** an existing SQL-stored account named **Admin** (if it already exists) and recreate it with a known password. It will create the **WebAdmins** group (if does not already exist) and assign full access rights for the role to the Root page.

1. In **AlloyTraining**, open **Web.config**.
2. Add an entry to `<appSettings>`, as shown in the following markup:

```
<add key="alloy:RegisterAdmin" value="true" />
```

3. Find the `<membership>` element and set the **defaultProvider** to **SqlServerMembershipProvider**, as shown in the following markup:

```
<membership defaultProvider="SqlServerMembershipProvider" ...
```

4. Find the <roleManager> element and set the **defaultProvider** to **SqlServerRoleProvider**, as shown in the following markup:

   ```
   <roleManager enabled="true" defaultProvider="SqlServerRoleProvider" ...
   ```

5. Save and close **Web.config**.
6. In **AlloyTraining**, open **~\Business\Initialization\RegisterAdminInitializationModule.cs**.
7. Review the file to understand what it does, as shown in the following code:

```
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.Security;
using System.Configuration;
using System.Web.Security;

namespace AlloyTraining.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class RegisterAdminInitializationModule : IInitializableModule
    {
        private const string roleName = "WebAdmins";
        private const string userName = "Admin";
        private const string password = "Pa$$w0rd";
        private const string email = "admin@alloy.com";

        public void Initialize(InitializationEngine context)
        {
            string enabledString =
                ConfigurationManager.AppSettings["alloy:RegisterAdmin"];
            bool enabled;
            if (bool.TryParse(enabledString, out enabled))
            {
                if (enabled)
                {
                    #region Use ASP.NET Membership classes to create the role and user

                    // if the role does not exist, create it
                    if (!Roles.RoleExists(roleName))
                    {
                        Roles.CreateRole(roleName);
                    }

                    // if the user already exists, delete it
                    MembershipUser user = Membership.GetUser(userName);
                    if (user != null)
                    {
                        Membership.DeleteUser(userName);
                    }

                    // create the user with password and add it to role
                    Membership.CreateUser(userName, password, email);
                    Roles.AddUserToRole(userName, roleName);

                    #endregion

                    #region Use EPiServer classes to give full access to root of
content tree

                    var security = context.Locate.Advanced
```

```
                    .GetInstance<IContentSecurityRepository>();

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

                permissions.AddEntry(new AccessControlEntry(
                    roleName, AccessLevel.FullAccess));

                security.Save(ContentReference.RootPage,
                    permissions, SecuritySaveType.Replace);

                permissions = security
                    .Get(ContentReference.WasteBasket)
                    .CreateWritableClone() as IContentSecurityDescriptor;

                permissions.AddEntry(new AccessControlEntry(
                    roleName, AccessLevel.FullAccess));

                security.Save(ContentReference.WasteBasket,
                    permissions, SecuritySaveType.Replace);

                #endregion
            }
        }
    }

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

8. Start the site, enter **/EPiServer/CMS/**, and log in as a CMS admin.
9. Close the browser.
10. In **AlloyTraining**, open **Web.config**.
11. Modify the entry in `<appSettings>` to disable the registration of Admin, as shown in the following markup:

    ```
    <add key="alloy:RegisterAdmin" value="false" />
    ```

12. Save and close Web.config.

You now have full access to CMS administrator features without needing a Windows account.

Defining a class of string constants for grouping content types and a Start page type

Content types can be grouped when shown as a list for the editors. You will start by defining a static class with string constants for the group names you will use in your site.

1. In **Solution Explorer**, right-click **AlloyTraining** project, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SiteGroupNames.cs** for the name, and click **Add**.
3. Modify the file, as shown in the following code:

    ```
    namespace AlloyTraining
    {
        public static class SiteGroupNames
        {
            public const string Specialized = "Specialized";
            public const string Common = "Common";
    ```

```
            public const string News = "News";
        }
    }
```

Remember that you can copy and paste or drag and drop solution files into your project to save time.

4. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **StartPage.cs** for the name, and click **Add**.
6. In the `[ContentType]` attribute, change the **DisplayName** to **Start**.
7. In the `[ContentType]` attribute, set the **GroupName** to **SiteGroupNames.Specialized** and the sort index (i.e. **Order**) within that group to **10**.
8. In the `[ContentType]` attribute, set the **Description** of "The home page for a website with an area for blocks and partial pages."
9. In the body of the class, uncomment the **MainBody** property with all its attributes and change its **Order** to **20**.
10. Add a string **Heading** property with appropriate attribute values. **Order** values are used to sort properties within a tab group. It is good practice to use multiples of ten so that future properties can be added between existing ones.

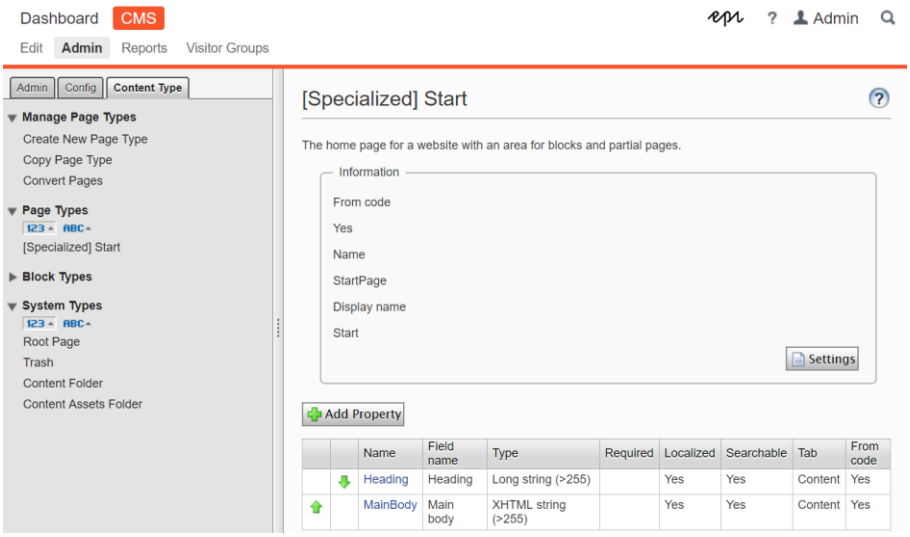
Your complete page type class should look something like the following code:

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

namespace AlloyTraining.Models.Pages
{
    [ContentType(DisplayName = "Start",
        // your code will have a random GUID here
        GroupName = SiteGroupNames.Specialized, Order = 10,
        Description = "The home page for a website with an area for blocks and partial
pages.")]
    public class StartPage : PageData
    {
        [CultureSpecific]
        [Display(Name = "Heading", Description =
            "If the Heading is not set, the page falls back to showing the Name.",
            GroupName = SystemTabNames.Content, Order = 10)]
        public virtual string Heading { get; set; }

        [CultureSpecific]
        [Display(Name = "Main body",
            Description = "The main body will be shown in the main content area of the
page, using the XHTML-editor you can insert for example text, images and tables.",
            GroupName = SystemTabNames.Content, Order = 20)]
        public virtual XhtmlString MainBody { get; set; }
    }
}
```

SystemTabNames.Content is the default so setting the **GroupName** attribute to this value is optional. Code in this exercise book, and in **cmsdevfun_exercisefiles.zip**, will not have GUIDs set for content types so that you can use the solution code and it will match based on namespace and class name instead of GUID values.

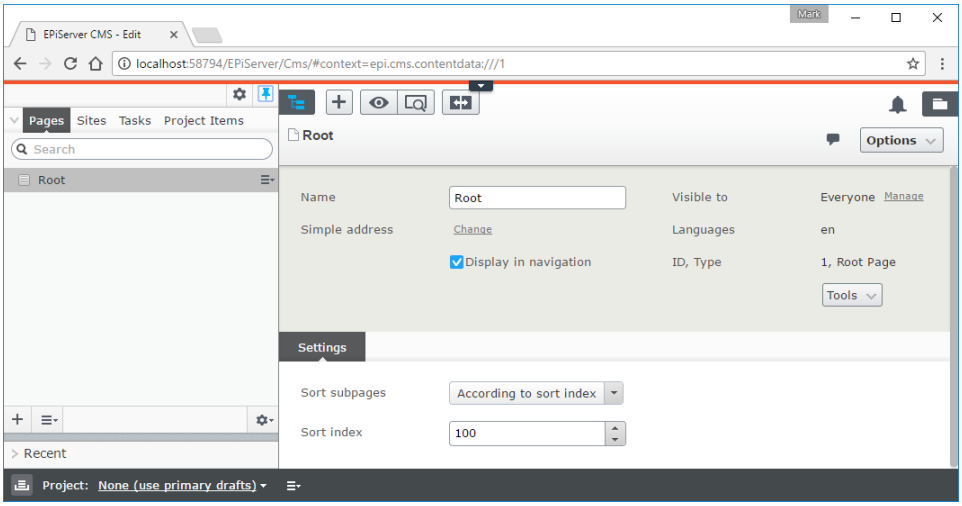
Creating a Start page instance

1. Start the website by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.
2. Enter **/EPiServer/CMS/** at the end of the address box.
3. Log in as a CMS admin, for example, **CMSUser** or **Admin**.
4. Navigate to **CMS** | **Admin** | **Content Type**, click **Start**, and note that the synchronization process has registered your page type, including its properties, as shown in the following screenshot:

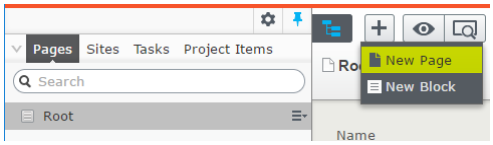


Members of the virtual role **CmsAdmins** can modify many of the values that are initially set by your code attributes by clicking **Settings**. These changes will override your code, even if you subsequently change your code. **CmsAdmins** can use the **Revert to Default** button to reset to the code attribute values.

5. Navigate to **CMS** | **Edit**.
6. **Navigation** pane shows only a **Root** page, as shown in the following screenshot:

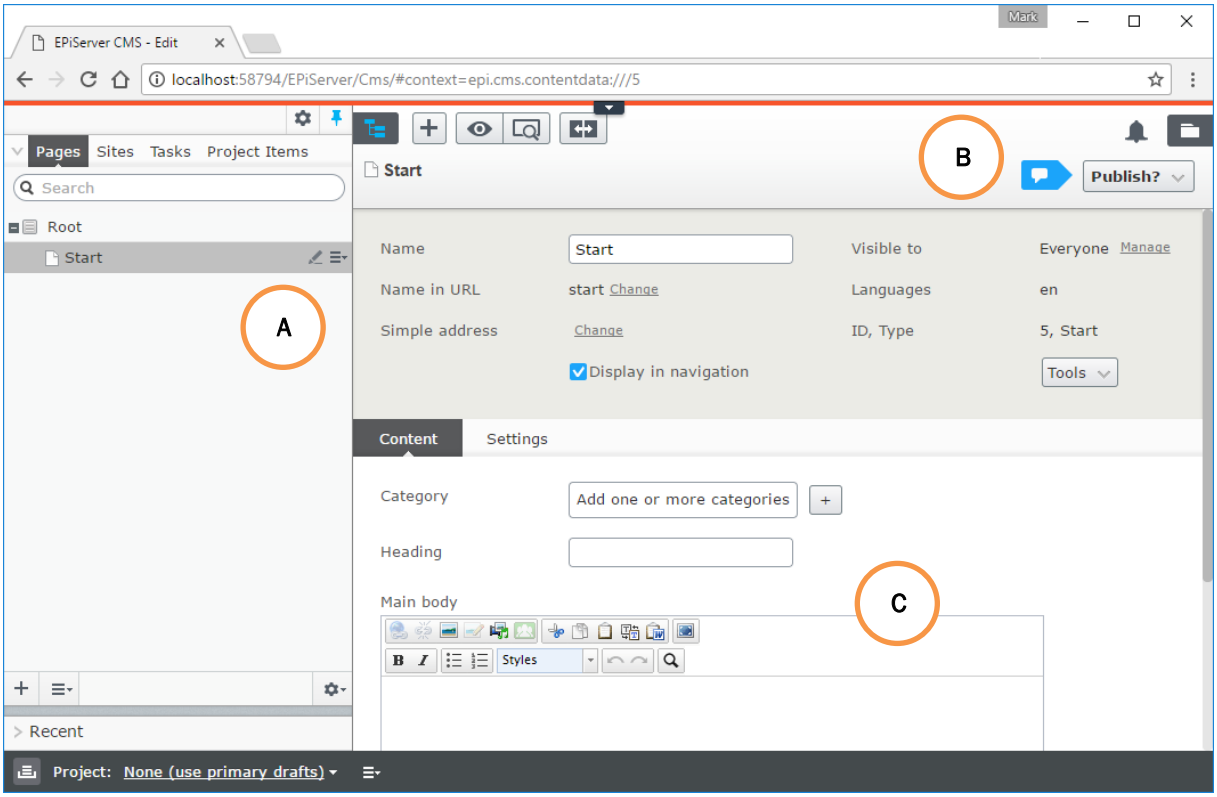


7. Click **+**, and then click **New Page**, as shown in the following screenshot:



8. Enter the name **Start**, and then click **OK**.

9. Note the following:
    a. Pencil icon indicates a saved draft, so the page is not published yet, and it won't be visible to visitors.
    b. Blue information icon also warns that you must publish this page.
    c. The two custom properties, **Heading** and **MainBody**, are grouped and sorted under the **Content** tab in **All Properties** view, as shown in the following screenshot:

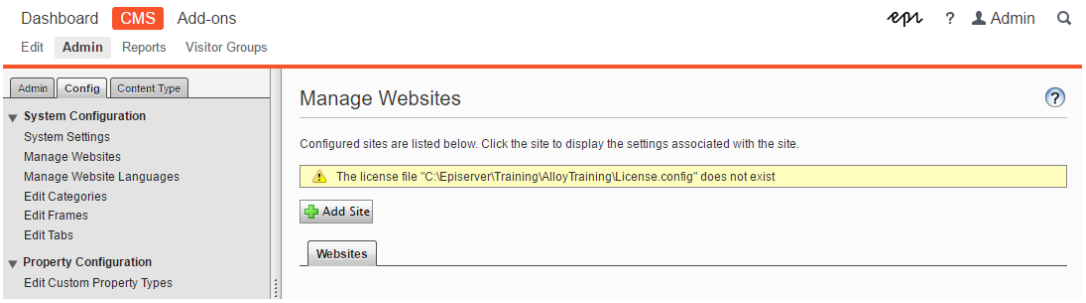


There is no On-Page Editing (OPE) yet because you have not created a page template.

10. Enter values for **Heading** and **Main body**. Be creative!

Every page has an inherited property named **Category**. If you do not want to use it, you can hide it. *Episerver CMS Advanced Development* training course shows you how.

11. Click **Publish**, and then click **Publish**.
12. Navigate to **CMS** | **Admin** | **Config** | **Manage Websites**, and click **Add Site**, as shown in the following screenshot:

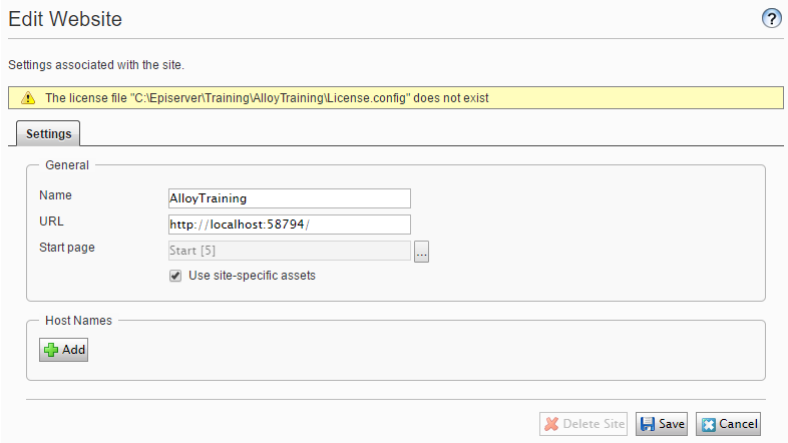


13. Enter the following details, as shown in the following screenshot:
    a. Name: AlloyTraining
    b. URL: [copy from browser address box]

c. Start page: click [...] to select the Start page you just created.

d. Use site-specific assets: selected

Use site-specific assets enables the **For This Site** folder in the **Assets** pane.

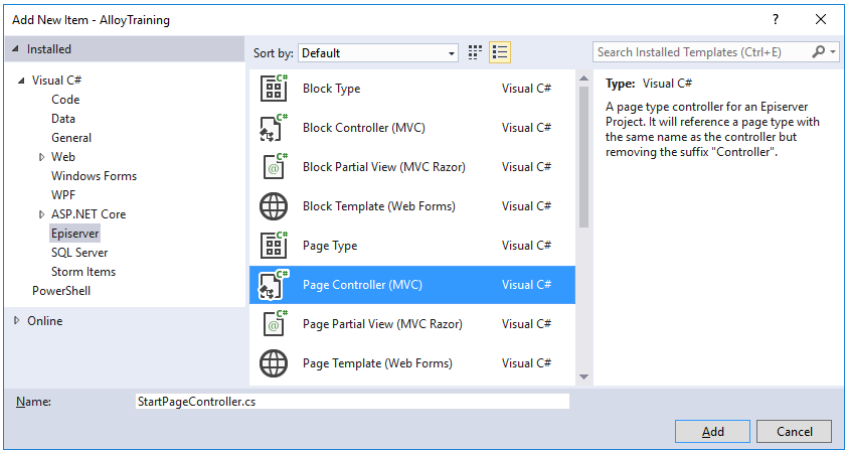


14. Click **Save**.
15. Switch to Edit view and note that the Start page now shows a “house” icon and the Global menu has a “globe” to switch to Live view.
16. Switch to Live view and note that the Start page still returns a 404 because it does not yet have a template.
17. Close the browser.

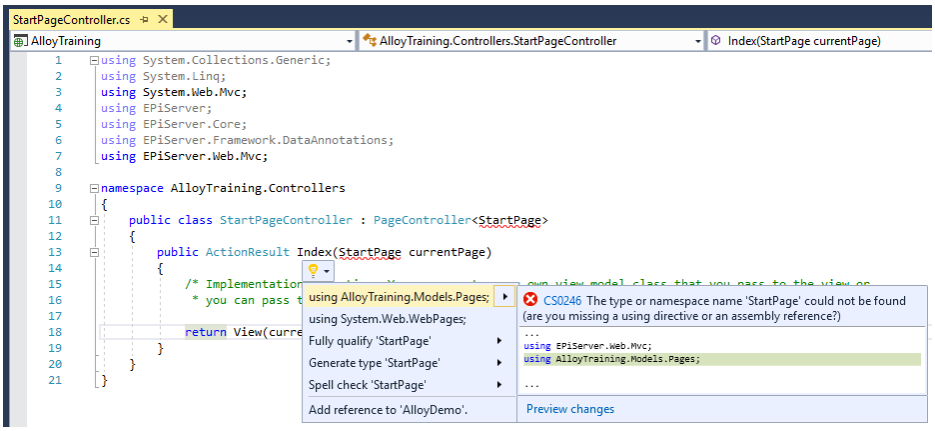
Creating a Start page template

A page template in MVC is a combination of a controller and a view.

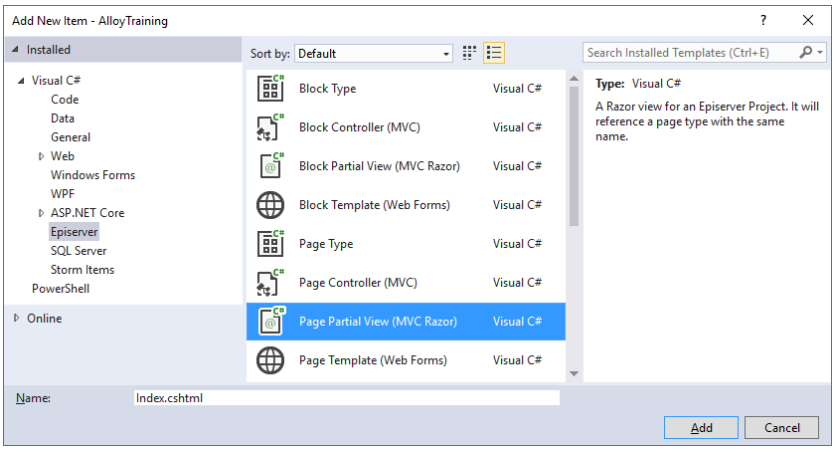
1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **StartPageController.cs** for the name, and click **Add**, as shown in the following screenshot:



3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace, as shown in the following screenshot:



4. In **AlloyTraining**, right-click **Views**, and add a new folder named **StartPage**.
5. Right-click **StartPage**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
6. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the name, and click **Add**, as shown in the following screenshot:



Most of the project and item templates provided by **Episerver CMS Visual Studio Extension** are well designed. But the name of this one is misleading, partly because there is no actual difference between a View and a Partial View; all .cshtml files are the same.

7. Change the **@model** to use the correct page type, as shown in the following code:

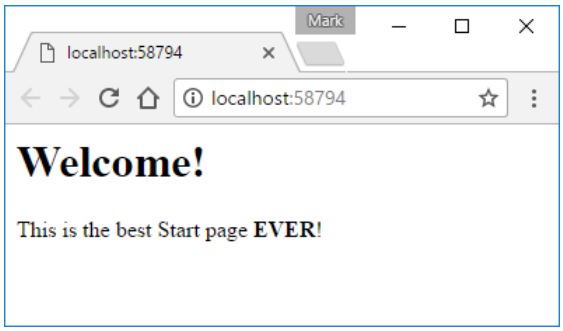
```
@model AlloyTraining.Models.Pages.StartPage
```

The existing code uses **PropertyFor** to output the **MainBody** which allows it to be edited on-page.

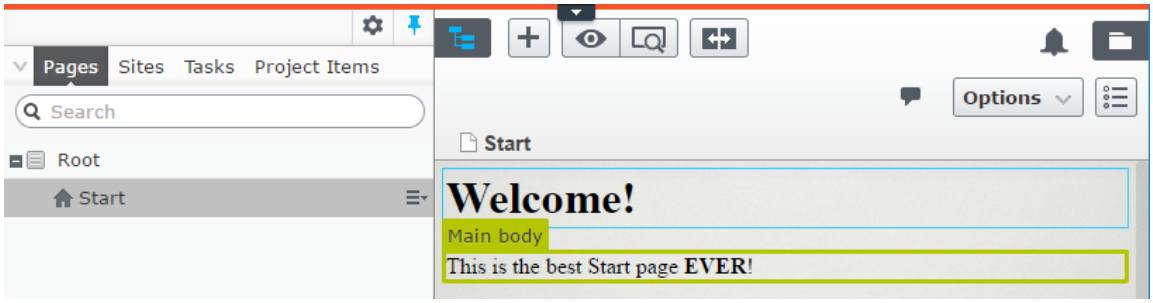
8. Add an `<h1>` element to output the **Heading** if it exists, otherwise fall back to the **Name**, as shown in the following markup:

```
<h1 @Html.EditAttributes(m => m.Heading)>
    @(Model.Heading ?? Model.Name)
</h1>
```

9. Start the website, and note the page template is used to render the **Start** page, as shown in the following screenshot:



10. Enter **/EPiServer/CMS/** at the end of the address box, and log in as a CMS admin.
11. Navigate to **CMS** | **Edit**., and note the On-Page Editing (OPE) experience due to the page template and use of **PropertyFor** and **EditAttributes** in the view, as shown in the following screenshot:



12. Verify that the on-page edit logic for the `<h1>` tag works, i.e. that when you edit the text it is the **Heading** property value that is updated, and that when a value exists for **Heading** it is that value that is rendered to the visitor, otherwise the **Name** is rendered.
13. Empty the **Heading** property value and publish the change to verify that the fallback works, i.e. that the page name is displayed to the visitor and to the editor.
14. Close the browser.

Localizing to the Swedish language for the Start page and its properties

1. Drag and drop or copy the **Resources** folder from **\cmsdevfun-exercisefiles\Module B\B1** to the root of the **AlloyTraining** project.

> Make sure you use **AlloyTraining**, because the demo project already has language files.

2. Expand the folder named **~\Resources\LanguageFiles** and open the file named **ContentTypes.xml**, and note its contents, as shown in the following markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <contenttypes>
      <startpage>
        <name>Start</name>
        <description>The home page for a website with an area for blocks and partial
pages.</description>
        <properties>
          <heading>
            <caption>Heading</caption>
            <help>If the Heading is not set, the page falls back to showing the
Name.</help>
          </heading>
          <mainbody>
```

```
            <caption>Main body</caption>
            <help>The main body will be shown in the main content area of the page,
using the XHTML-editor you can insert for example text, images and tables.</help>
          </mainbody>
        </properties>
      </startpage>
    </contenttypes>
  </language>
  <language name="Svenska" id="sv">
    <contenttypes>
      <startpage>
        <name>Start</name>
        <description>Hemsidan för en webbplats med ett område för block och
sidor.</description>
        <properties>
          <heading>
            <caption>Rubrik</caption>
            <help>Om rubriken inte är inställd, faller sidan tillbaka för att visa
namnet.</help>
          </heading>
          <mainbody>
            <caption>Huvudkroppen</caption>
            <help>Huvudkroppen kommer att visas i huvudinnehållet på sidan med hjälp
av XHTML-redigeraren som du kan infoga tex, bilder och tabeller.</help>
          </mainbody>
        </properties>
      </startpage>
    </contenttypes>
  </language>
</languages>
```

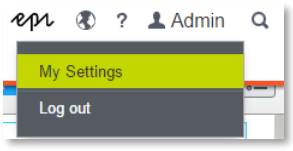
3. Open **Web.config**.
4. In the **<episerver.framework>** element, add the following markup:

Copy and paste this element from **\cmsdevfun-exercisefiles\Module B\B1\Web.config-changes.txt**.

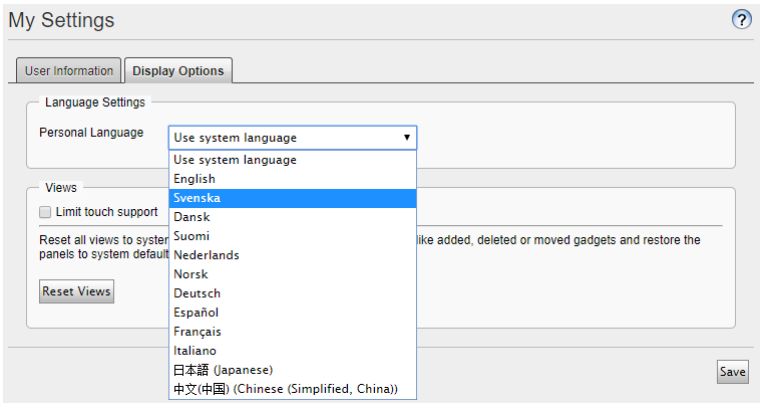
```
<localization fallbackBehavior="Echo, MissingMessage, FallbackCulture"
fallbackCulture="en">
  <providers>
    <add virtualPath="~/Resources/LanguageFiles" name="languageFiles"
         type="EPiServer.Framework.Localization.XmlResources
               .FileXmlLocalizationProvider, EPiServer.Framework.AspNet" />
  </providers>
</localization>
```

Breaking change in CMS 11
Assembly name has changed to EPiServer.Framework.AspNet. For CMS 10, you would have used just EPiServer.Framework.

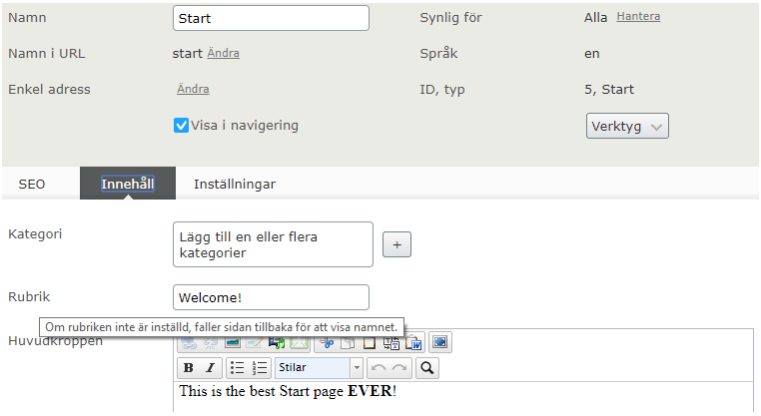
5. Start **AlloyTraining** by navigating to **Debug** | **Start Without Debugging** or press *Ctrl + F5*.
6. Enter **/EPiServer/CMS/** at the end of the address box, and log in.
7. In the **Global** menu, click your user name menu, and then **My Settings**, as shown in the following screenshot:



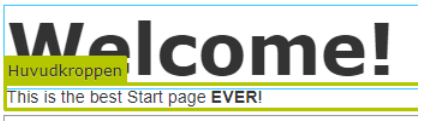
8. Click the **Display Options** tab. The Admin user has their **Personal Language** set to **Use system language**, which on my laptop, is English. Change the language to **Svenska** and click **Save**, as shown in the following screenshot:



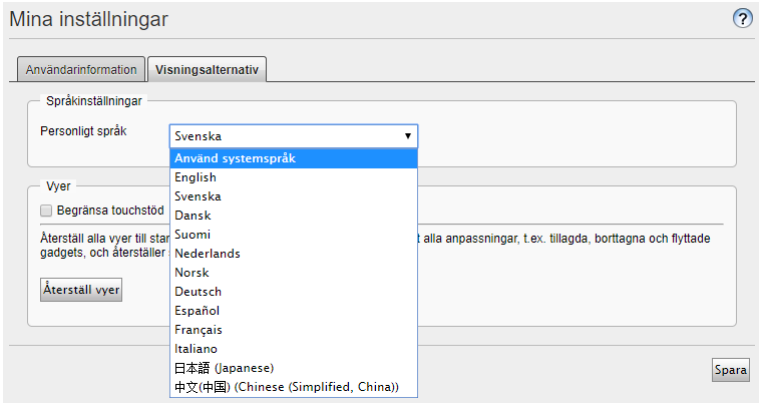
9. Edit the **Start** page, switch to **All Properties** view, and note the property names and tool tips of your custom **Heading/Rubrik** and **Main body/Huvudkroppen** have been localized into Swedish, as shown in the following screenshot:



10. Switch to on-page editing and note the same:



11. Pull down the Global menu, click your user name menu, click **Mina installningar**, click **Visningsalternativ**, and switch back to **Anvand systemsprak**, as shown in the following screenshot:



12. Close the browser.

Exercise B2 – Managing media assets

In this exercise, you will define some media types to enable files to be uploaded.

Prerequisites: complete Exercise B1.

Enabling upload of any file by defining a media type

Before uploading files, a little code is needed for the system to be able to recognize media.

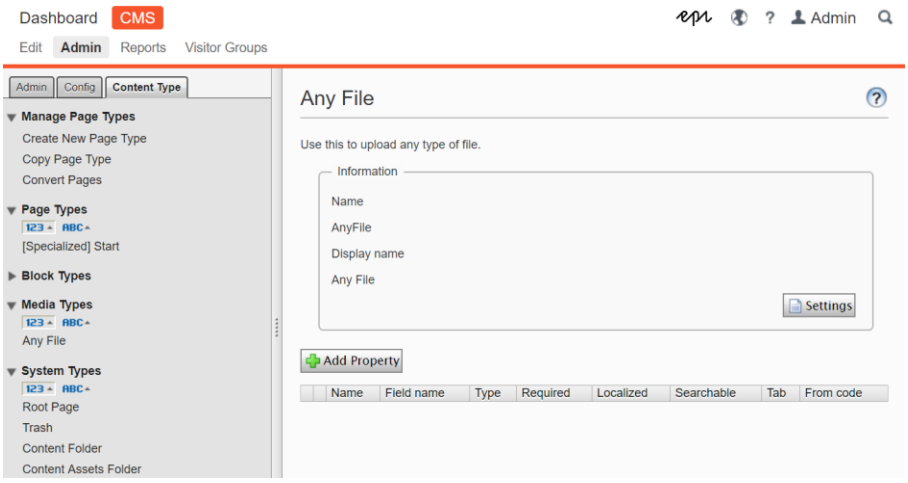
1. In **AlloyTraining**, expand **Models**, right-click **Media**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Media Type**, enter **AnyFile.cs** for the name, and click **Add**.
3. Change the **DisplayName** to **Any File**.
4. Add a **Description** of "Use this to upload any type of file."
5. Delete the commented example property and the commented **MediaDescriptor** attribute.

Your code should look something like the following:

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

namespace AlloyTraining.Models.Media
{
    [ContentType(DisplayName = "Any File",
        // your code will have a GUID
        Description = "Use this to upload any type of file.")]
    public class AnyFile : MediaData
    {
    }
}
```

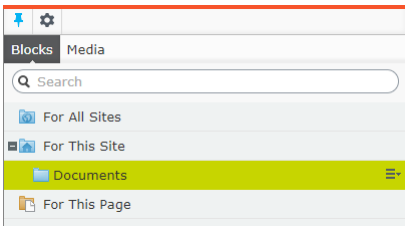
Uploading files

1. Start the **AlloyTraining** website, and log in as a CMS Admin.
2. Navigate to **CMS** | **Admin** | **Content Type**, and select **Any File**, as shown in the following screenshot:



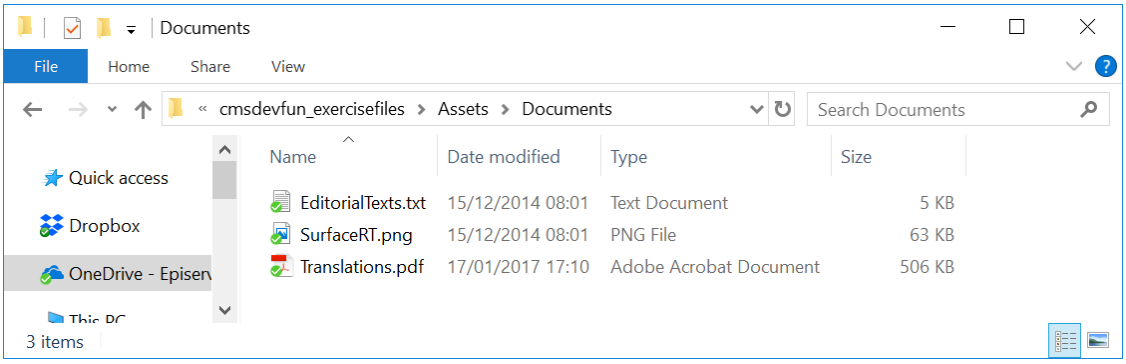
3. Navigate to **CMS** | **Edit**.

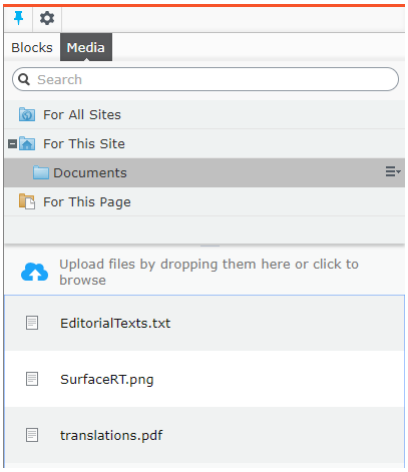
4. In **Assets** pane, create a folder named **Documents** in **For This Site**, as shown in the following screenshot:



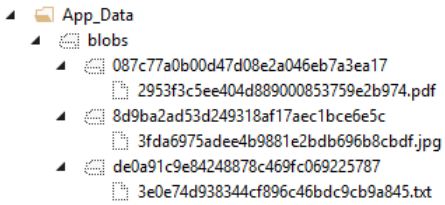
It does not matter if **Blocks** or **Media** is currently selected; they both share the same folder structure.

5. Click **Media** and select the **Documents** folder.
6. Start **File Explorer**.
7. From the folder **\cmsdevfun-exercisefiles\Assets\Documents**, drag and drop **SurfaceRT.png**, **EditorialTexts.txt**, and **Translations.pdf** into the **Documents** folder, as shown in the following screenshots:



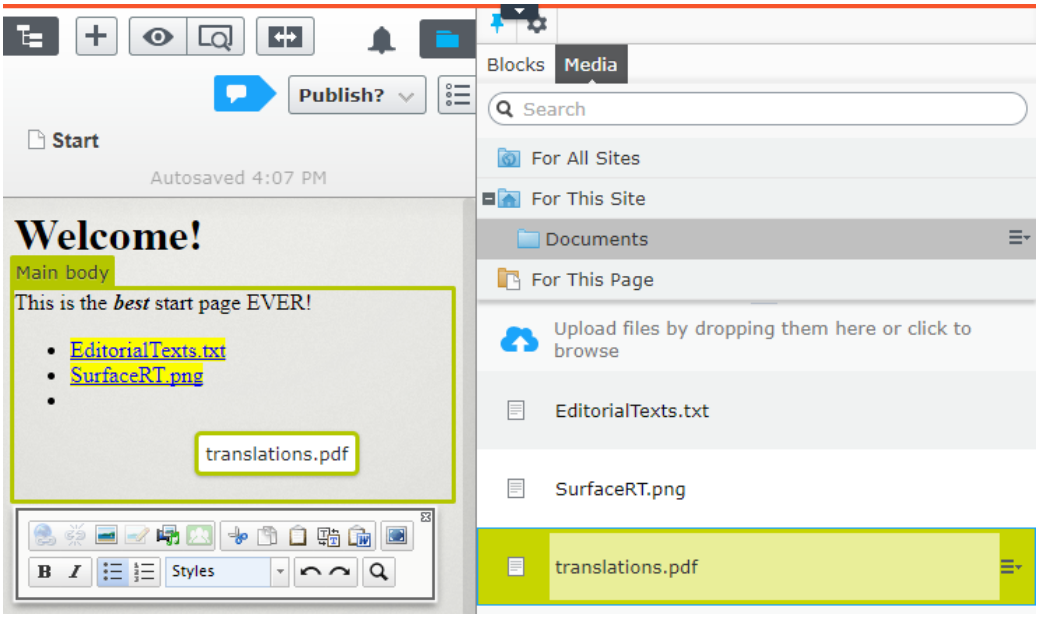


8. Leave the browser running, and in Visual Studio, in **Solution Explorer**, expand the **App_Data** folder, toggle **Show All Files**, refresh to show the **blobs** folder, and note that each uploaded file has its own folder that matches a content GUID in the CMS database, and a file for a version, as shown in the following screenshot:

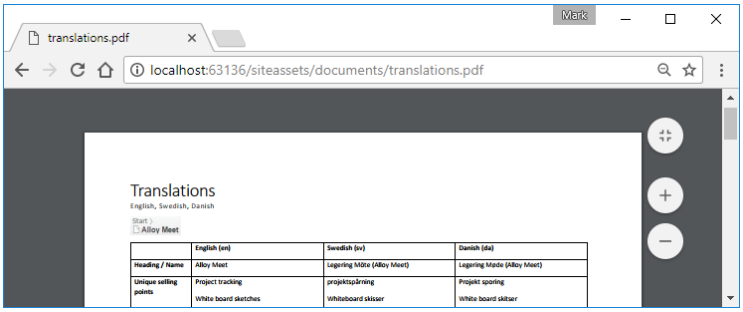


The default BLOB provider uses the filesystem. You can configure alternative BLOB providers, for example, to use Microsoft Azure Blob Storage or Amazon Web Services S3.

9. Back in the browser, edit the **Start** page.
10. In the **Main body**, create a bulleted list, drag and drop the three files, and note that links to the media files are created, and there is no special handling for images, as shown in the following screenshot:



11. **Publish** the Start page.
12. Switch to **Live** view, and click on each link to see the effect for visitors, as shown in the following screenshot:



Customize handling of images by defining an image type

Before uploading image files, a little code is needed for the system to be able to recognize images as a special type of media and therefore customize how they are handled when dragged and dropped into a rich text property.

The previously uploaded image, SurfaceRT.png, will remain registered as an “Any File” content type. To give it the special image handling, it would have to be deleted and uploaded again. There is a built-in Admin view feature to convert pages, but it cannot be used to convert other content types.

1. In **AlloyTraining**, expand **Models**, right-click **Media**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Media Type**, enter **ImageFile.cs** for the name, and click **Add**.
3. Modify the class to inherit from **ImageData**.

4. Change the **DisplayName** to **Image File**.
5. Add a **Description** of "Use this to upload image files."
6. Uncomment the **MediaDescriptor** attribute and set the file extensions to: **png,gif,jpg,jpeg**
7. Uncomment the **Description** property and delete its **Display** attribute.

Your code should look something like the following:

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

namespace AlloyTraining.Models.Media
{
    [ContentType(DisplayName = "Image File",
        // your code will have a GUID
        Description = "Use this to upload image files.")]
    [MediaDescriptor(ExtensionString = "png,gif,jpg,jpeg")]
    public class ImageFile : ImageData
    {
        [CultureSpecific]
        [Editable(true)]
        public virtual string Description { get; set; }
    }
}
```


Make sure you inherit from `ImageData`, not `MediaData`!

Enabling upload of SVG files by defining a media type

`ImageData` does not recognise Scalable Vector Graphics (SVG) files so we must define a separate content type to handle them.

1. In **AlloyTraining**, expand **Models**, right-click **Media**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Media Type**, enter **SvgFile.cs** for the name, and click **Add**.
3. Modify the class to inherit from **ImageData**.
4. Change the **DisplayName** to **SVG File**.
5. Add a **Description** of "Use this to upload Scalable Vector Graphic (SVG) images."
6. Uncomment the **MediaDescriptor** attribute and set the file extensions to: **svg**
7. Delete the **Description** property and its **Display** attribute.
8. Override the **Thumbnail** property to return the **BinaryData** property.

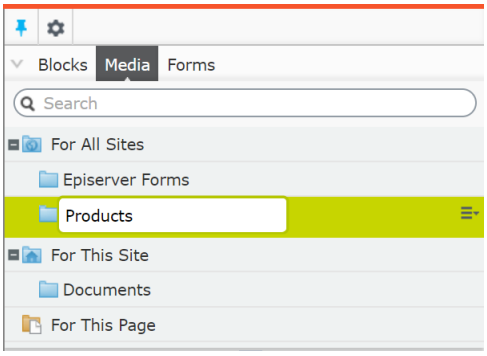
Your code should look something like the following:

```
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;
using EPiServer.Framework.Blobs;
using EPiServer.Framework.DataAnnotations;

namespace AlloyDemo.Models.Media
{
    [ContentType(DisplayName = "SVG File",
        // your code will have a GUID
        Description = "Use this to upload Scalable Vector Graphic (SVG) images.")]
```

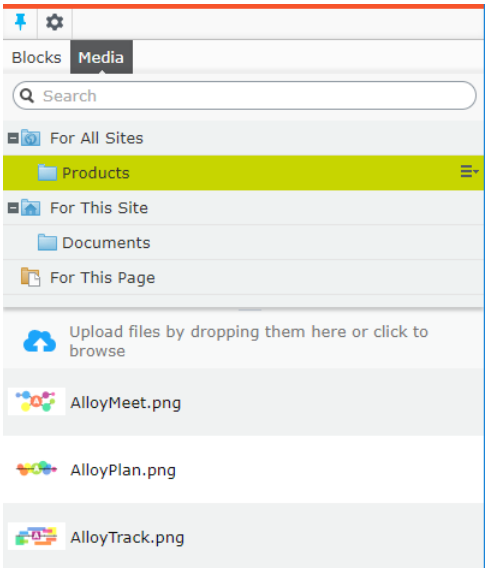
```
    [MediaDescriptor(ExtensionString = "svg")]
    public class SvgFile : ImageData
    {
        // instead of generating a smaller bitmap file for thumbnail,
        // use the same binary vector image for thumbnail
        public override Blob Thumbnail { get => base.BinaryData; }
    }
}
```

Uploading images

1. Start the **AlloyTraining** site, and log in as a CMS Admin.
2. Navigate to **CMS** | **Edit**.
3. In **Assets** pane, click **Media**, and create a folder named **Products** in **For All Sites**, as shown in the following screenshot:

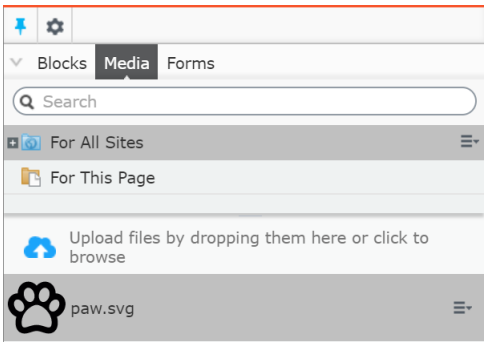


4. Upload images of the three products from **cmsdevfun_exercisefiles.zip** in the folder **\Assets\Products** into the **Products** folder, as shown in the following screenshot:

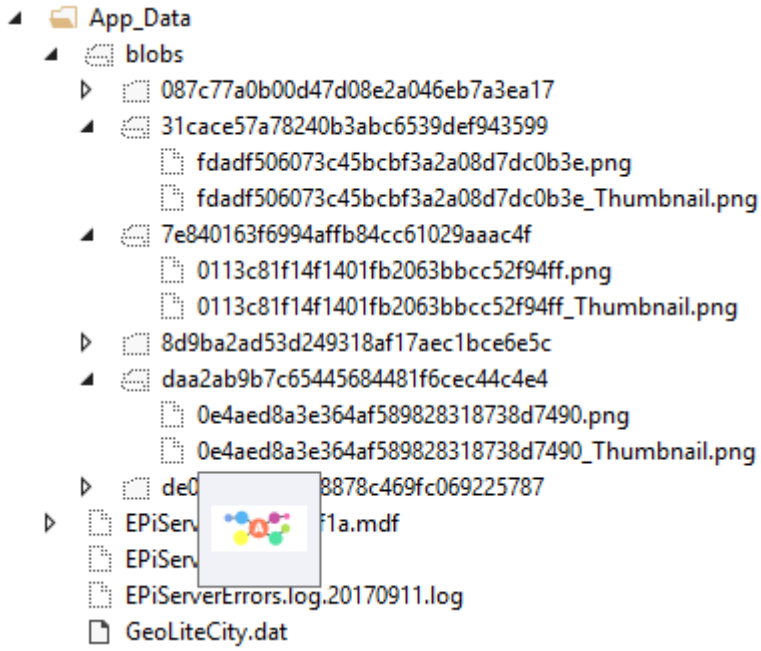


One feature of inheriting from `ImageData` instead of `MediaData` is the automatic thumbnail generation.

5. Upload the file named **paw.svg** in the folder **\Assets\Misc** into the **For All Sites** folder, as shown in the following screenshot:



6. Leave the browser running, and in Visual Studio, in Solution Explorer, in the **App_Data** folder, refresh the **blobs** folder, and note that each uploaded product image has its own folder that matches a content GUID in the CMS database, and a file for a version, and a file for a thumbnail, as shown in the following screenshot:



7. Back in the browser, double-click each image in the **Assets** pane, switch to **All Properties** view, edit the **Name** and **Description** properties, and then **Publish** them:

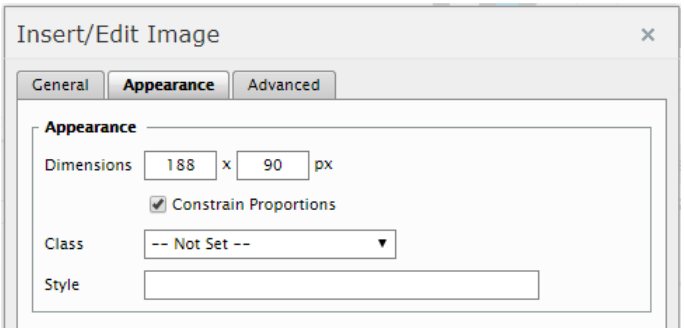
   a. **AlloyMeet.png**:
   Name: **Alloy Meet**, Description: **Logo of the Alloy Meet product**.

   b. **AlloyPlan.png**:
   Name: **Alloy Plan**, Description: **Logo of the Alloy Plan product**.

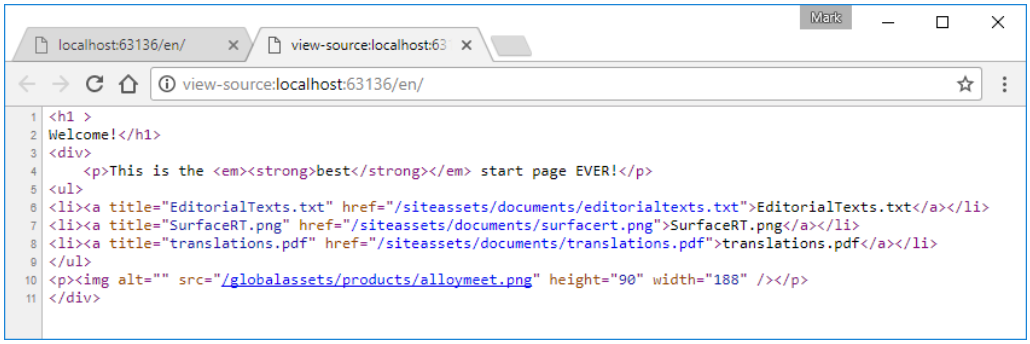
   c. **AlloyTrack.png**:
   Name: **Alloy Track**, Description: **Logo of the Alloy Track product**.

8. Edit the **Start** page.
9. Drag and drop one of the product images into the **Main body** and note the image renders as an `<img>` element instead of a clickable hyperlink.
10. Select the image.

11. In the toolbar, click **Insert/Edit Image**, click **Appearance**, and change the height to **90**, as shown in the following screenshot:



12. Click **Update**.
13. Publish the **Start** page.
14. Switch to **Live view**, to view the page as a visitor.
15. Right-click the page and choose **View page source**.
16. Note the HTML generated for visitors, especially the URLs for the documents and images that you uploaded as media assets, as shown in the following screenshot:



`/siteassets/` is the **For This Site** folder.
`/globalassets/` is the **For All Sites** folder.

Exercise B3 – Implementing design patterns and conventions

Once you start working on this exercise, do not try to run the website until you have completed all the tasks in the exercise. If you run the website in the middle of the exercise, then you will see exceptions.

In this exercise, you will create a layout file which will be used up by all page templates in the website, a base page type that will be inherited from by all the page types in the site, a base page controller that will be inherited from by all the page templates in the site, and a view model to make our Views and Layouts more flexible.

- The layout will be named **_Root.cshtml**.
- The layout will be created in the folder **~\Views\Shared\Layouts** and contain references to Bootstrap files you added in Exercise B1.

Prerequisites: complete Exercises B1 and B2.

~\Views\Shared\Layouts_Root.cshtml

```
<head>
</head>
<body>
```

~\Views\StartPage\Index.cshtml

```
</body>
```

Creating a page type base

This class will not be a page type, but it will serve as an abstract class inheriting and extending the **PageData** class. Therefore, no attributes are added to the class, but the class should be abstract.

1. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SitePageData.cs** for the Name, and click **Add**.
3. Import the `EPiServer.Core` namespace.
4. Modify the class to be `abstract` and inherit from `PageData`.
5. Define five `public virtual` properties:
   - **MetaTitle**: `string`
   - **MetaKeywords**: `string`
   - **MetaDescription**: `string`
   - **PageImage**: `ContentReference`
   - **TeaserText**: `string`

Do not apply any attributes to the properties yet. You will do that in the next section.

Applying property attributes

Before you add attributes, you will define some string constants to use in your site.

1. In **AlloyTraining**, right-click **AlloyTraining** project, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SiteTabNames.cs** for the name, and click **Add**.
3. Modify the file, as shown in the following code, and note the following:
    - To see properties on the **SEO** tab, the user must have **Edit** access level (access right).
    - To see properties on the **Site Settings** tab, the user must have **Administer** access level.

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

namespace AlloyTraining
{
    [GroupDefinitions]
    public static class SiteTabNames
    {
        [Display(Order = 10)]
        [RequiredAccess(AccessLevel.Edit)]
        public const string SEO = "SEO";

        [Display(Order = 20)]
        [RequiredAccess(AccessLevel.Administer)]
        public const string SiteSettings = "Site Settings";
    }
}
```

4. In **AlloyTraining**, open **SitePageData.cs**.
5. Apply attributes to the properties to:
    - Make the following properties support a plain text editor with multiple rows: **MetaDescription**, **TeaserText**.
    - Make **PageImage** property only able to point to images.
    - Make the following properties support multiple language branches: **MetaTitle**, **MetaKeywords**, **MetaDescription**, **TeaserText**.
    - Make the following properties appear on the **SEO** tab: MetaTitle, MetaKeywords, MetaDescription.
    - Make the following properties appear on the **Content** tab: PageImage, TeaserText.
    - Sort the properties within each tab appropriately.
    - Limit the **MetaTitle** to between 5 and 60 characters (Google's recommendation for page titles to get good SEO).

Your complete class should look something like the following:

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

namespace AlloyTraining.Models.Pages
```

```
{
    public abstract class SitePageData : PageData
    {
        [CultureSpecific]
        [Display(Name = "Meta title",
            GroupName = SiteTabNames.SEO, Order = 100)]
        [StringLength(60, MinimumLength = 5)]
        public virtual string MetaTitle { get; set; }

        [CultureSpecific]
        [Display(Name = "Meta keywords",
            GroupName = SiteTabNames.SEO, Order = 200)]
        public virtual string MetaKeywords { get; set; }

        [CultureSpecific]
        [Display(Name = "Meta description",
            GroupName = SiteTabNames.SEO, Order = 300)]
        [UIHint(UIHint.Textarea)] // multi-row text editor
        public virtual string MetaDescription { get; set; }

        [Display(Name = "Page image",
            GroupName = SystemTabNames.Content, Order = 100)]
        [UIHint(UIHint.Image)] // filters to only show images
        public virtual ContentReference PageImage { get; set; }

        [CultureSpecific]
        [Display(Name = "Teaser text",
            GroupName = SystemTabNames.Content, Order = 200)]
        [UIHint(UIHint.Textarea)]
        public virtual string TeaserText { get; set; }

    }
}
```

6. Drag and drop **\cmsdevfun-exercisefiles\Module B\B3\Resources** folder into **AlloyTraining** project.

~\Resources\LanguageFiles\ContentType.xml has entries to localize any page types that inherit from **SitePageData** and its meta properties. Even without needing Swedish, it is still worth having a localization language file to translate between programmer-speak, e.g. the MetaDescription property, and editor-speak, e.g. “Page description”.

Creating a view model interface

1. In **AlloyTraining**, right-click **~\Models**, and add a new folder named **ViewModels**.
2. Right-click **ViewModels**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Interface**, enter **IPageViewModel.cs** for the name, and click **Add**.
4. Modify the interface to define a covariant generic type that derives from **SitePageData**, with read-only properties named **CurrentPage** and **Section**, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using EPiServer.Core;

namespace AlloyTraining.Models.ViewModels
{
    public interface IPageViewModel<out T> where T : SitePageData
    {
        T CurrentPage { get; }
        IContent Section { get; }
    }
}
```

Section property will be used to reference the page that is shown in the top-level navigation menu. For example, the news article **Alloy Saves Bears** is in the **About us** section. You will now use an extension method to set this property programmatically.

Creating a default view model class

1. In **AlloyTraining**, right-click **ViewModels**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **DefaultPageViewModel.cs** for the name, and click **Add**.
3. Modify the class to implement the interface, have a constructor for setting the current page, and a property to store the parent section, as shown in the following code:

```
using AlloyTraining.Business.ExtensionMethods;
using AlloyTraining.Models.Pages;
using EPiServer.Core;

namespace AlloyTraining.Models.ViewModels
{
    public class DefaultPageViewModel<T>
        : IPageViewModel<T> where T : SitePageData
    {
        public DefaultPageViewModel(T currentPage)
        {
            CurrentPage = currentPage;
            Section = currentPage.ContentLink.GetSection();
        }

        public T CurrentPage { get; set; }
        public IContent Section { get; set; }
    }
}
```

Creating a shared layout

1. In **AlloyTraining**, right-click **~\Views\Shared**, and add a new folder named **Layouts**.
2. Drag and drop or copy the **_Root.cshtml** file from **\cmsdevfun-exercisefiles\Module B\B3\Views\Shared\Layouts** folder to the same folder in **AlloyTraining**.
3. Open the **_Root.cshtml** file, as shown in the following markup, and note the following:
   - The import of some namespaces for pages, view models and extension methods.
   - `@model` directive allows any type that implements the view model interface to be passed to the layout.
   - The `MetaTitle` (or `Name` as a fallback) of the page is used for the `<title>`
   - Links to stylesheet and script files are added to the `<head>`
   - `@Html.RenderEPiServerQuickNavigator()` adds the **epi** Quick Access menu

```
@using AlloyTraining.Business.ExtensionMethods
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model IPageViewModel<SitePageData>
<html lang="@Model.CurrentPage.Language">
    <head>
        <meta charset="utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=10" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>@(Model.CurrentPage.MetaTitle ?? Model.CurrentPage.Name)</title>
```

```
        <meta name="keywords" content="@Model.CurrentPage.MetaKeywords" />
        <meta name="description" content="@Model.CurrentPage.MetaDescription" />
        <link rel="stylesheet" href="@Url.Content("~/Static/css/bootstrap.css")" />
        <link rel="stylesheet"
              href="@Url.Content("~/Static/css/bootstrap-responsive.css")" />
        <link rel="stylesheet" href="@Url.Content("~/Static/css/media.css")" />
        <link rel="stylesheet" href="@Url.Content("~/Static/css/style.css")" />
        <link rel="stylesheet" href="@Url.Content("~/Static/css/editmode.css")" />
        <script type="text/javascript"
                src="@Url.Content("~/Static/js/jquery.js")"></script>
        <script type="text/javascript"
                src="@Url.Content("~/Static/js/bootstrap.js")"></script>
    </head>
    <body>
        @Html.RenderEPiServerQuickNavigator()
        <div class="container">
            <div class="row">
                <div id="header">
                    <div class="span2">
                        <img src="/Static/gfx/logotype.png" />
                    </div>
                    <div class="span10">
                        @if (User.Identity.IsAuthenticated)
                        {
                            <a href="/en/logout">Log out</a>
                        }
                        else
                        {
                            <a
href="@FormsAuthentication.LoginUrl?ReturnUrl=@Model.CurrentPage.PageLink.ExternalURLF
romReference()">Log in</a>
                        }
                    </div>
                </div>
            </div>
            <hr />
            @RenderBody()
        </div>
    </body>
</html>
```

Setting the layout as the default

1. In **AlloyTraining**, right-click **~\Views**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Web**, choose **MVC 5 View Page (Razor)**, enter **_ViewStart.cshtml** for the **Name**, and click **Add**.
3. Delete all the HTML markup in **_ViewStart.cshtml**.
4. Set the layout to the path for **_Root.cshtml**, as shown in the following code:

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

_ViewStart.cshtml is executed when the **View()** method is called inside a controller's action method. It is not executed when the **PartialView()** method is called. This is the only difference between a "full" and "partial" .cshtml file. All .cshtml files themselves are actually the same.

Using the view model in the Start page view

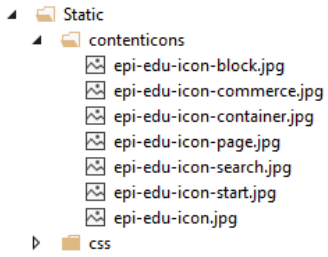
1. Open **~\Views\StartPage\Index.cshtml**.
2. Import namespaces for view models, modify the **@model** directive to use the default view model, and use the **CurrentPage** throughout the view markup, as shown in the following code:

```
@using AlloyTraining.Models.ViewModels
@using EPiServer.Web.Mvc.Html
@model DefaultPageViewModel<AlloyTraining.Models.Pages.StartPage>
<h1 @Html.EditAttributes(m => m.CurrentPage.Heading)>
    @(Model.CurrentPage.Heading ?? Model.CurrentPage.Name)
</h1>
<div>
    @Html.PropertyFor(m => m.CurrentPage.MainBody)
</div>
```

Creating site content icons

If you do not apply Episerver's **ImageUrl** attribute, then when editors create new pages and blocks, a missing icon is shown. You will create custom icons and apply them to your content types.

1. In **AlloyTraining**, copy the **contenticons** folder from **\cmsdevfun_exercisefiles\Module B\B3\Static** to the **Static** folder in **AlloyTraining**, as shown in the following screenshot:



2. Right-click **AlloyTraining**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SiteContentIcons.cs** for the name, and click **Add**.
4. Modify the contents, as shown in the following code:

```
using EPiServer.DataAnnotations;

namespace AlloyTraining
{
    public class SiteImageUrlAttribute : ImageUrlAttribute
    {
        public SiteImageUrlAttribute()
            : base("~/Static/contenticons/epi-edu-icon.jpg") { }

        public SiteImageUrlAttribute(string path)
            : base(path) { }
    }

    public class SitePageIconAttribute : ImageUrlAttribute
    {
        public SitePageIconAttribute()
            : base("~/Static/contenticons/epi-edu-icon-page.jpg") { }
    }

    public class SiteBlockIconAttribute : ImageUrlAttribute
    {
        public SiteBlockIconAttribute()
            : base("~/Static/contenticons/epi-edu-icon-block.jpg") { }
    }
```

```
        public class SiteStartIconAttribute : ImageUrlAttribute
        {
            public SiteStartIconAttribute()
                : base("~/Static/contenticons/epi-edu-icon-start.jpg") { }
        }

        public class SiteSearchIconAttribute : ImageUrlAttribute
        {
            public SiteSearchIconAttribute()
                : base("~/Static/contenticons/epi-edu-icon-search.jpg") { }
        }

        public class SiteCommerceIconAttribute : ImageUrlAttribute
        {
            public SiteCommerceIconAttribute()
                : base("~/Static/contenticons/epi-edu-icon-commerce.jpg") { }
        }

        public class SiteContainerIconAttribute : ImageUrlAttribute
        {
            public SiteContainerIconAttribute()
                : base("~/Static/contenticons/epi-edu-icon-container.jpg") { }
        }
    }
```

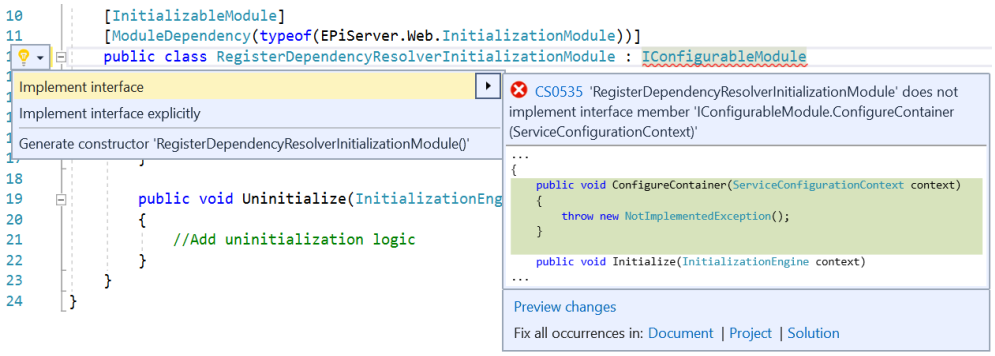
Registering a dependency resolver

To enable controllers to have constuctors with parameters to inject dependencies you must register a dependency resolver with ASP.NET MVC. You will create an initialization module to make sure that Episerver's integration with StructureMap is registered as the resolver when the website starts up.


You will learn more about initialization modules in Module F.

1. In **AlloyTraining**, right-click the **Business** folder and add a new folder named **DependencyResolvers**.
2. Drag and drop or copy the **StructureMapDependencyResolver.cs** file from **\cmsdevfun-exercisefiles\Module B\B3\Business\DependencyResolvers** folder to the same folder in **AlloyTraining**.
3. In **AlloyTraining**, expand the **Business** folder, right-click the **Initialization** folder, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
4. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Initialization Module**, enter **RegisterDependencyResolverInitializationModule.cs** for the name, and click **Add**.
5. Import the `AlloyTraining.Business.DependencyResolvers`, `System.Web.Mvc`, and `EPiServer.ServiceLocation` namespaces.
6. Modify the class to make it implement the `IConfigurableModule` interface.

7. Use Visual Studio IntelliSense to implement the missing `ConfigureContainer` method, as shown in the following screenshot:



8. Implement the `ConfigureContainer` and `Initialize` methods, as shown in the following code:

```
using AlloyTraining.Business.DependencyResolvers; // StructureMapDependencyResolver
using EPiServer.Framework; // [InitializableModule], [ModuleDependency]
using EPiServer.Framework.Initialization; // InitializationEngine
using EPiServer.ServiceLocation; // IConfigurableModule, ServiceConfigurationContext
using System.Web.Mvc; // DependencyResolver

namespace AlloyTraining.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class RegisterDependencyResolverInitializationModule : IConfigurableModule
    {
        public void ConfigureContainer(ServiceConfigurationContext context)
        {
            DependencyResolver.SetResolver(
                new StructureMapDependencyResolver(context.StructureMap()));

            //Implementations for custom interfaces can be registered here.

            context.ConfigurationComplete += (o, e) =>
            {
                //Register custom implementations that should be used in favour of the
default implementations
            };
        }

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

The `ConfigureContainer` method is where you can add statements to replace Episerver services like `IContentLoader` with your own implementations, and where you can register your own custom servies.

Creating a base page controller

To enable every page to have a shared layout with a **Log out** link, and to have access to the content loader service, you will create a base page controller with a **Logout** action method.

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.

2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **PageControllerBase.cs** for the name, and click **Add**.
3. Modify the class to make it abstract, restrict the generic type to be derived from `SitePageData`, and to have a `Logout` action method, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using EPiServer.Web.Mvc;
using System.Web.Mvc;
using System.Web.Security;

namespace AlloyTraining.Controllers
{
    public abstract class PageControllerBase<T>
        : PageController<T> where T : SitePageData
    {
        protected readonly IContentLoader loader;

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

        public ActionResult Logout()
        {
            FormsAuthentication.SignOut();
            return RedirectToAction("Index");
        }
    }
}
```

Modifying Start page to use the base classes

1. In **AlloyTraining**, open **StartPage.cs**.
2. Modify the class to inherit from **SitePageData** and apply the **SiteStartIcon** attribute, as shown in the following code:

```
[ContentType(...)]
[SiteStartIcon]
public class StartPage : SitePageData
```

3. Open **StartPageController.cs**.
4. Modify the class to derive from **PageControllerBase**, as shown in the following code:

```
public class StartPageController : PageControllerBase<StartPage>
```

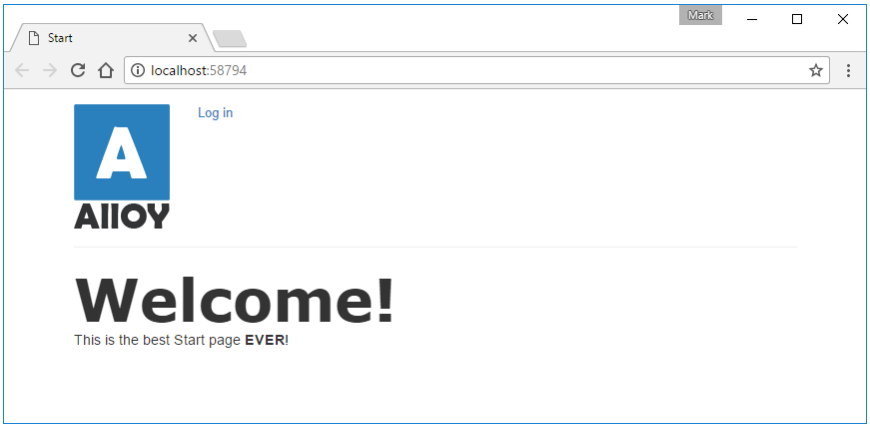
5. In the `Index` action method, create an instance of `DefaultPageViewModel<StartPage>` and pass the current page as a parameter, and then pass the view model into the view, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class StartPageController : PageControllerBase<StartPage>
    {
        public StartPageController(IContentLoader loader) : base(loader)
        {
        }

        public ActionResult Index(StartPage currentPage)
```

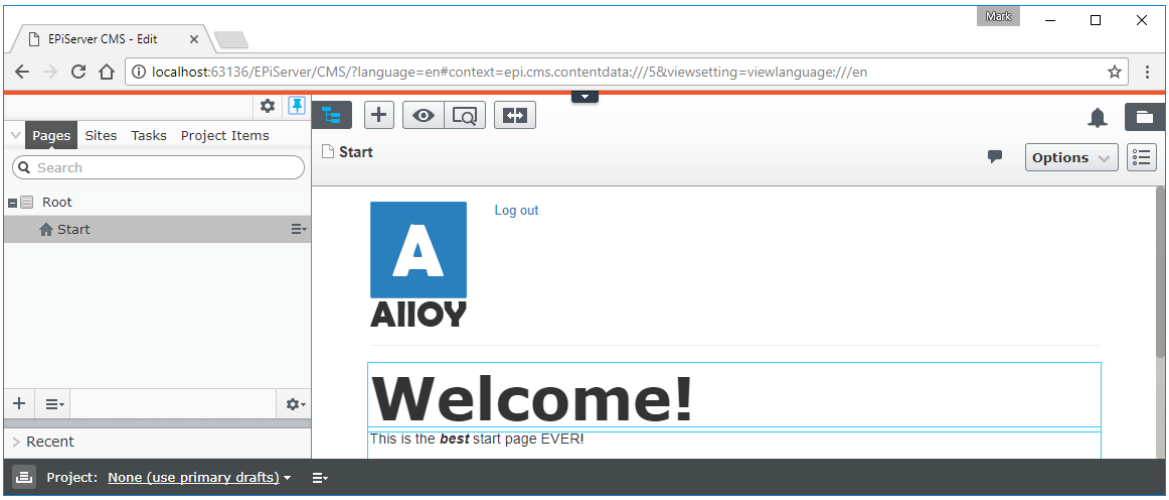
```
            {
                var viewmodel = new DefaultPageViewModel<StartPage>(currentPage);
                return View(viewmodel);
            }
        }
    }
```

Testing the website

1. Start the **AlloyTraining** website and note the layout is used for visitors, as shown in the following screenshot:



2. Log in and note the layout is used for on-page editing, as shown in the following screenshot:



Defining site settings on Start page for administrators only

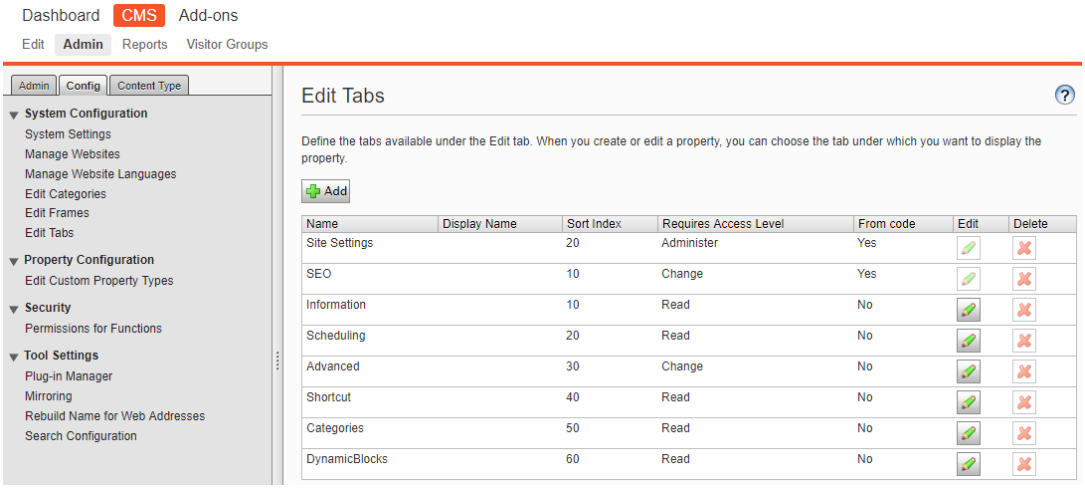
It is good practice to store settings that affect the site on the **Start** page for the site.

1. In the **AlloyTraining** project, open **StartPage.cs**.
2. Add a localizable property named **FooterText**, and put it under the **Site Settings** tab, as shown in the following code:

```
[CultureSpecific]
[Display(Name = "Footer text",
    Description = "The footer text will be shown at the bottom of every page.",
    GroupName = SiteTabNames.SiteSettings,
    Order = 10)]
public virtual string FooterText { get; set; }
```

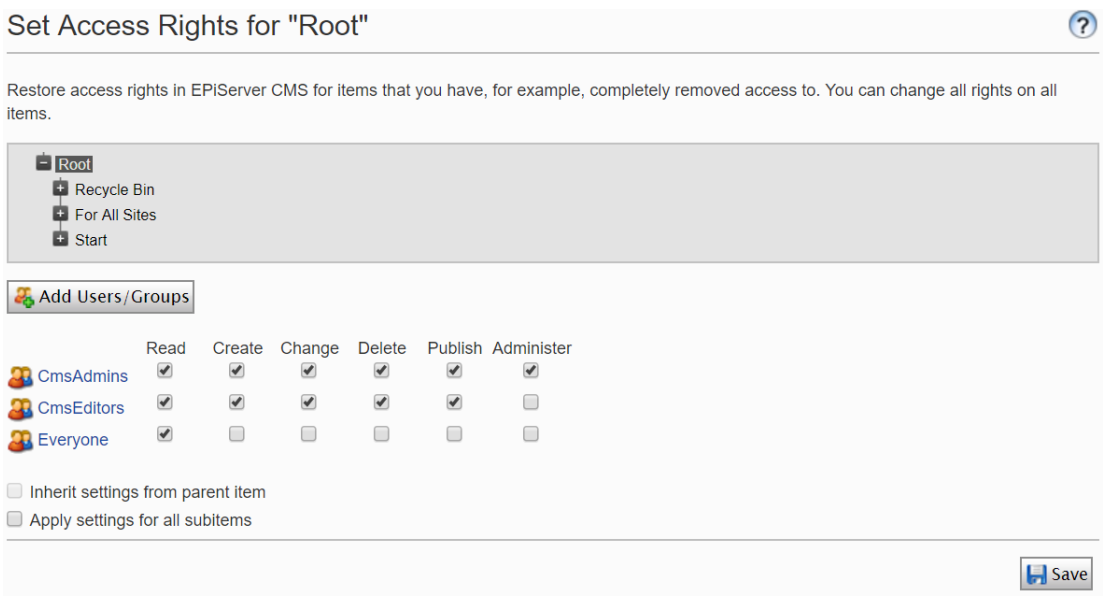
3. Start the **AlloyTraining** site, and log in as a CMS admin.

4. Navigate to **CMS** | **Admin** | **Config** | **Edit Tabs**, and note that the two tabs defined in **SiteTabNames** static class, **SEO** and **Site Settings**, have been registered from code, as shown in the following screenshot:



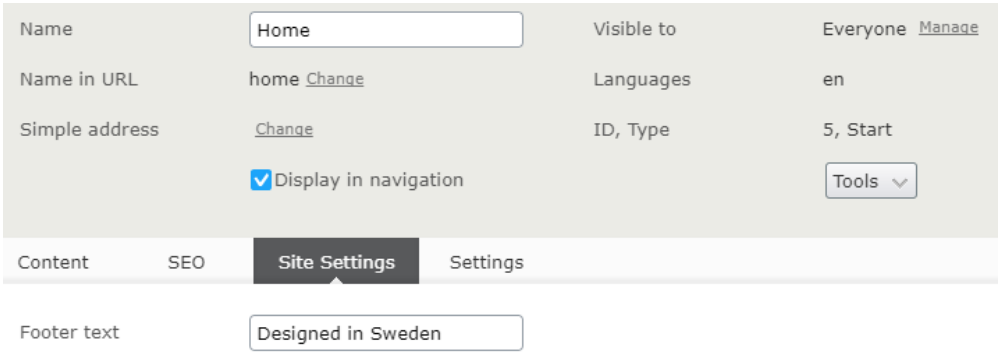
To see and edit properties on the **SEO** tab, a user must have **Change** access rights. To see and edit properties on the **Site Settings** tab, a user must have **Administer** access rights. Do not allow users to edit a property protected in this way using On-Page Editing because that would bypass this security. Therefore, properties on the Site Settings tab should never be output in the view using PropertyFor!

5. Navigate to **CMS** | **Admin** | **Admin** | **Administer Groups**, and add a new group named **WebEditors**.
6. Navigate to **CMS** | **Admin** | **Admin** | **Create User**, and add a new user named **Edward** and make him a member of **WebEditors**.
7. Navigate to **CMS** | **Admin** | **Admin** | **Set Access Rights**, select **Root**, add the virtual role named **CmsEditors**, set all access rights except **Administer**, click **Save**, as shown in the following screenshot:



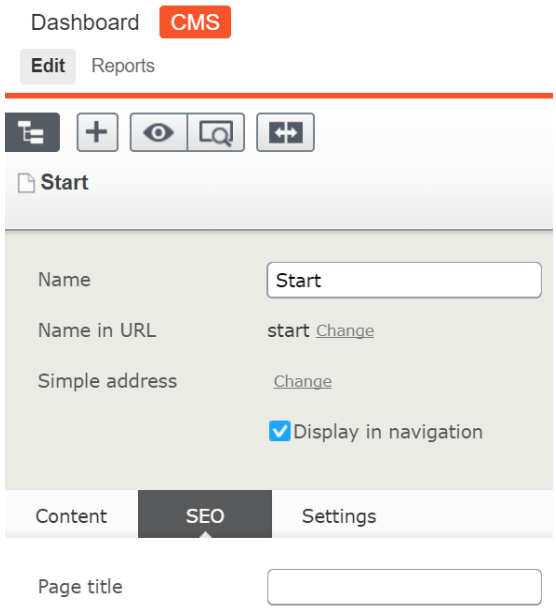
Since **CmsEditors** do not have **Administer** access rights to the **Root** (and all its children), they will not be able to see or edit properties on the **Site Settings** tab in **All Properties** view.

8. Edit the **Start** page, switch to **All Properties** view, click **Site Settings**, enter a value for the footer text, and note that it has tabs for **SEO** and **Site Settings**, as shown in the following screenshot:



We are not currently rendering the **Footer text** in the layout. As an optional challenge, output its value in a <footer> element in the **_Root.cshtml** layout. Hint: write an extension method that returns the FooterText property of the StartPage. A potential solution is in **\cmsdevfun-exercisefiles\Module B\B3\Business\ExtensionMethods\LayoutExtensionMethods.cs** and **\cmsdevfun-exercisefiles\Module B\B3\Views\Shared\Layout_Root-optional-challenge.cshtml**.

9. Log out as a CMS admin, and log back in as **Edward**.
10. Edit the **Start** page, switch to **All Properties** view, and note that **Edward** or other members of **CmsEditors** do not have the ability to see **Site Settings**, as shown in the following screenshot:



11. Click the **SEO** tab with three properties that have been localized, as shown in the following screenshot:

If the localization strings are not loading, try switching to Swedish, and then back to English. Doing this forces the cached strings to reload and normally fixes the problem.

12. Click the **Content** tab and note the new properties for **Page image** and **Teaser text**.
13. Pull down the Global menu, and click the globe icon to switch to visitor view, as shown in the following screenshot:

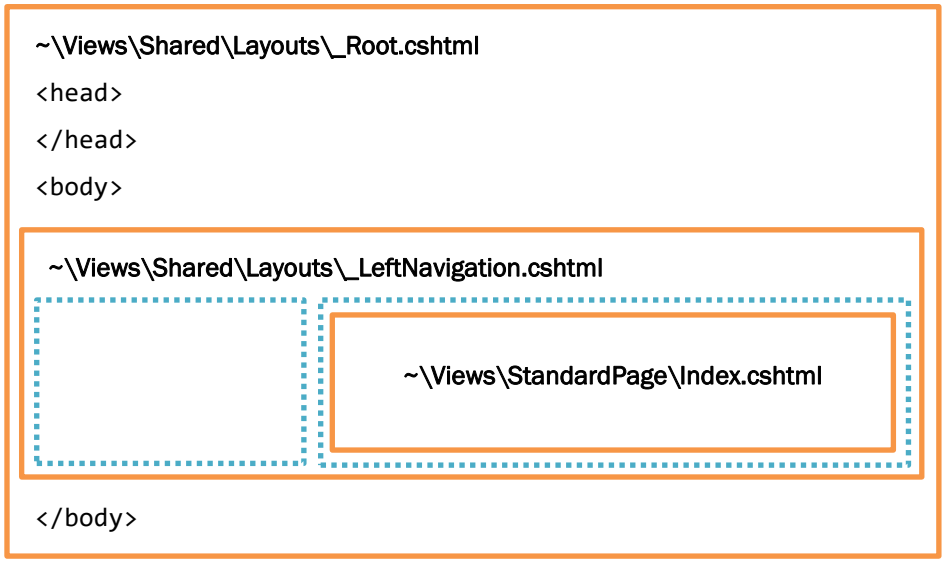
14. You are still logged in and can see the **epi** menu.
15. Click **Log out**. You are now back to an anonymous visitor.

Exercise B4 – Creating page types with a shared layout and navigation

Prerequisites: complete Exercises B1 to B3.

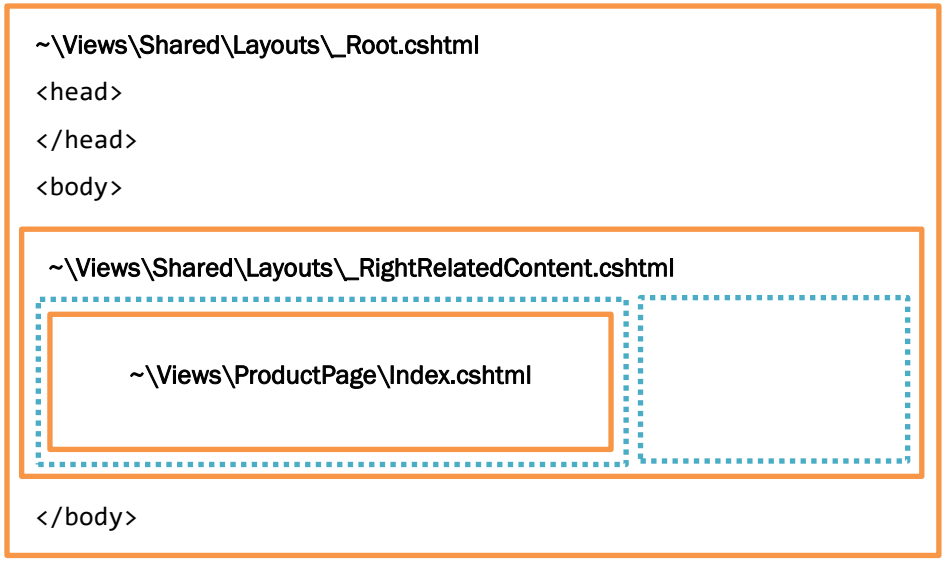
In this exercise, you will create a page type named Standard that will be used for generic pages in the site.

- It will inherit from **SitePageData** class, making sure it gets all the SEO properties and any other properties from this class.
- It will have a **MainBody** property for body text.
- It will use a **_LeftNavigation.cshtml** layout to have a navigation submenu.
- It will have a template, displaying some of the properties on the page type.
- You will create an instance of the Standard page named "About us".



You will create a page type named Product that will be used for product pages in the site.

- It will inherit from **StandardPage**.
- It will have a template with two thirds for main content, and one third for related content.
- You will create three instances of Product page, "Alloy Meet", "Alloy Track", and "Alloy Plan".



You will add a menu to the site to navigate between children of the Start page.

- You will use **IContentLoader** to retrieve all pages on the level below the Start page in the page tree.

The interface and its methods will be explained in more detail later in the course, this exercise merely shows how you can easily retrieve and display pages in the site programmatically.

Creating the Standard page type

1. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **StandardPage.cs** for the Name, and click **Add**.
3. Modify the class to inherit from **SitePageData**.
4. Change the **DisplayName** to **Standard**.
5. Set group to **SiteGroupNames.Common**.
6. Add a **Description** of "Use this page type unless you need a more specialized one."
7. Apply the **[SitePageIcon]** attribute to the class.
8. Uncomment the **MainBody** property with all its attributes and change its Order to 310.

The Order in this example is set in relation to the other properties that were added on the Content tab in the SitePageData base class that this page type inherits from, for example, the TeaserText property has Order = 200 and we want the MainBody property to be displayed below it in All Properties view.

Your code should look something like the following:

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

namespace AlloyTraining.Models.Pages
{
    [ContentType(DisplayName = "Standard",
        GroupName = SiteGroupNames.Common,
        Description = "Use this page type unless you need a more specialized one.")]
    [SitePageIcon]
    public class StandardPage : SitePageData
    {
        [CultureSpecific]
        [Display(Name = "Main body",
            Description = "The main body will be shown in the main content area of the
page, using the XHTML-editor you can insert for example text, images and tables.",
            GroupName = SystemTabNames.Content,
            Order = 310)]
        public virtual XhtmlString MainBody { get; set; }
    }
}
```

Creating a left navigation layout

1. In **AlloyTraining**, right-click **~\Views\Shared\Layouts**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **_LeftNavigation.cshtml** for the Name, and click **Add**.
3. Modify the view as shown in the following markup, and note the following:

- `@model` allows any view model that has a **CurrentPage** property whose type derives from `SitePageData` to be passed to the layout.
- This layout is nested inside the **_Root.cshtml** layout.
- Bootstrap is used to divide the layout into one third for the (future) left navigation submenu and two thirds for the body of the web page.

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model IPageViewModel<SitePageData>
@{
    // nest the left navigation inside the root layout
    Layout = "~/Views/Shared/Layouts/_Root.cshtml";
}
<div class="row">
    <div class="span4">
        @RenderSection("_LeftNavigationMenu", required: false)
    </div>
    <div class="span8">
        @RenderBody()
    </div>
</div>
```

Creating a controller for the Standard page type

1. In **Solution Explorer**, in **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **StandardPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace.
4. Modify the class to derive from `PageControllerBase<StandardPage>`, and the `Index` action method to use a view model, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class StandardPageController : PageControllerBase<StandardPage>
    {
        public StandardPageController(IContentLoader loader) : base(loader)
        {
        }

        public ActionResult Index(StandardPage currentPage)
        {
            var viewmodel =
                new DefaultPageViewModel<StandardPage>(currentPage);
            return View(viewmodel);
        }
    }
}
```

Make sure you pass the view model to the `View` method, not the current page, or you will see a type mismatch runtime exception message.

5. On the **Build** menu, click **Build Solution**.

Creating a view for the Standard page

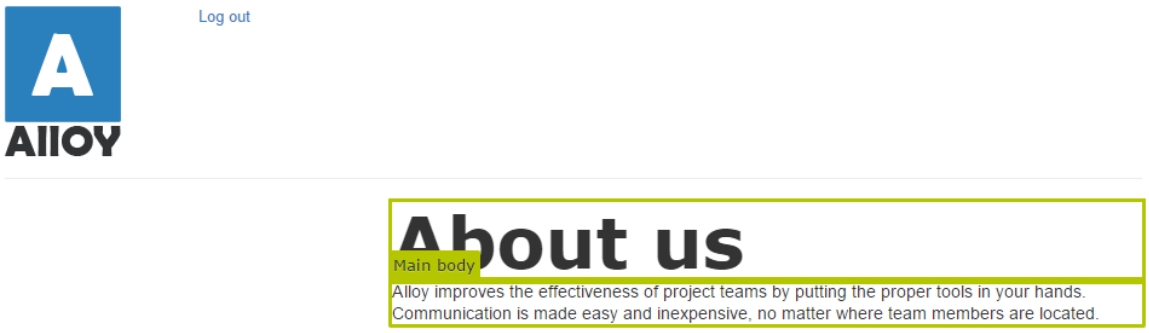
1. In **AlloyTraining**, right-click **Views**, and add a new folder named **StandardPage**.
2. Right-click **StandardPage**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the name, and click **Add**.
4. Modify the view, as shown in the following markup:

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<StandardPage>
@{
    Layout = "~/Views/Shared/Layouts/_LeftNavigation.cshtml";
}
<h1 @Html.EditAttributes(m => m.CurrentPage.MetaTitle)>
    @(Model.CurrentPage.MetaTitle ?? Model.CurrentPage.Name)
</h1>
<div class="row">
    <div class="span8 clearfix">
        @Html.PropertyFor(x => x.CurrentPage.MainBody)
    </div>
</div>
```

You would not normally output the `MetaTitle` in the body of a page but we will do it just as a simple example since we have not defined a `Heading` property.

Creating an instance of the Standard page

1. Start the **AlloyTraining** website and log in.
2. Add a new **Standard** page named **About us** underneath the **Start** page.
3. In on-page editing, set the **Main body** to some text, for example: “Alloy improves the effectiveness of project teams by putting the proper tools in your hands. Communication is made easy and inexpensive, no matter where team members are located.” Note the two-thirds layout, as shown in the following screenshot:



4. Switch to **All Properties** view and set some approproate **SEO** properties, for example, for the **Page title**: “About us title”, and **Page description**: “Alloy improves the effectiveness of project teams by putting the proper tools in your hands. Communication is made easy and inexpensive, no matter where team members are located.”
5. Publish the page.
6. Switch to Live view and note the page’s title.

Creating the Product page type

1. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **ProductPage.cs** for the Name, and click **Add**.
3. Modify the class to inherit from **StandardPage**.
4. Change the **DisplayName** to **Product**.
5. Group the page under **Specialized**.
6. Add a **Description** of "Use this for software products that Alloy sells."
7. Apply the **[SiteCommerceIcon]** attribute to the class.
8. Delete the **MainBody** property.
9. Add the following property and allow it to be localized into multiple languages:
    - Name: `UniqueSellingPoints`
    - Type: `string`
    - GroupName: `SystemTabNames.Content`
    - Order: 320
    - UIHint: Textarea
    - Required: force the editor to provide a value when adding a new product page

Your code should look something like the following:

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

namespace AlloyTraining.Models.Pages
{
    [ContentType(DisplayName = "Product",
        GroupName = SiteGroupNames.Specialized,
        Order = 20,
        Description = "Use this for software products that Alloy sells.")]
    [SiteCommerceIcon]
    public class ProductPage : StandardPage
    {
        [CultureSpecific]
        [Display(Name = "Unique selling points",
            GroupName = SystemTabNames.Content,
            Order = 320)]
        [UIHint(UIHint.Textarea)]
        [Required]
        public virtual string UniqueSellingPoints { get; set; }
    }
}
```

Creating a right related content layout

1. Right-click **~\Views\Shared**, and add a new folder named **Layouts**.
2. Right-click **~\Views\Shared\Layouts**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **_RightRelatedContent.cshtml** for the Name, and click **Add**.
4. Modify the view as shown in the following mark, noting the following:

- `@model` allows any view model that has a CurrentPage property that derives from `SitePageData` to be passed to the layout.
- This layout is nested inside the _Root.cshtml layout.
- Bootstrap is used to divide the layout into two thirds for the body of the web page and one third for the (optional) related content.

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model IPageViewModel<SitePageData>
@{
    // nest the right related content inside the root layout
    Layout = "~/Views/Shared/Layouts/_Root.cshtml";
}
<div class="row">
    <div class="span8">
        @RenderBody()
    </div>
    <div class="span4">
        @RenderSection("RelatedContent", required: false)
    </div>
</div>
```

Creating a controller for the Product page type

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **ProductPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace.
4. Modify the class to derive from **PageControllerBase<T>**, and the **Index** action method to use a view model, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class ProductPageController
        : PageControllerBase<ProductPage>
    {
        public ProductPageController(IContentLoader loader) : base(loader)
        {
        }

        public ActionResult Index(ProductPage currentPage)
        {
            var viewmodel = new DefaultPageViewModel<ProductPage>(currentPage);
            return View(viewmodel);
        }
    }
}
```

5. On the **Build** menu, click **Build Solution**.

Creating a view for the Product page

1. In **AlloyTraining**, right-click **Views**, and add a new folder named **ProductPage**.
2. Right-click **ProductPage**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
4. Modify the view as shown in the following mark, noting the following:
   - This view is nested inside the **_RightRelatedContent.cshtml** layout.
   - The body of the view will be injected into the layout where **RenderBody()** was called, inside the first two-thirds .
   - The section **RelatedContent** will be injected into the layout inside the last third .

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<ProductPage>
@{
    Layout = "~/Views/Shared/Layouts/_RightRelatedContent.cshtml";
}
<h1 @Html.EditAttributes(x => x.CurrentPage.Name)>@Model.CurrentPage.Name</h1>
<p class="introduction"
   @Html.EditAttributes(x => x.CurrentPage.MetaDescription)>
   @Model.CurrentPage.MetaDescription</p>
<div class="row">
    <div class="span8 clearfix">
        @Html.PropertyFor(x => x.CurrentPage.MainBody)
    </div>
</div>
@section RelatedContent
{
    <div @Html.EditAttributes(x => x.CurrentPage.PageImage)>
        <img src="@Url.ContentUrl(Model.CurrentPage.PageImage)" />
    </div>
    <div class="block colorBox theme1">
        <h2 @Html.EditAttributes(x => x.CurrentPage.Name)>
            @Model.CurrentPage.Name
        </h2>
        <p>@Html.PropertyFor(x => x.CurrentPage.UniqueSellingPoints)</p>
    </div>
}
```

Creating instances of Product page type

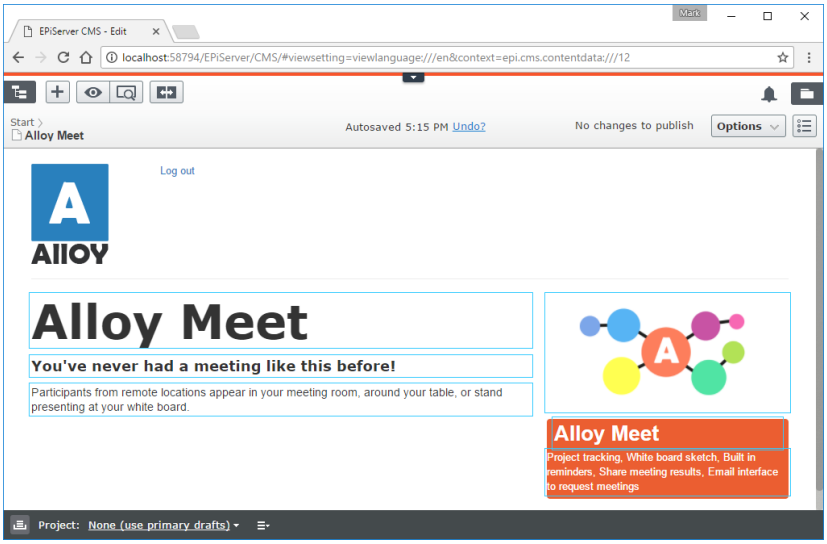
1. Build and start the **AlloyTraining** website.
2. Add three **Product** pages under the **Start** page, using the following table:

If you get exceptions, skip ahead to **Optional: Understanding common exceptions** on page 105.

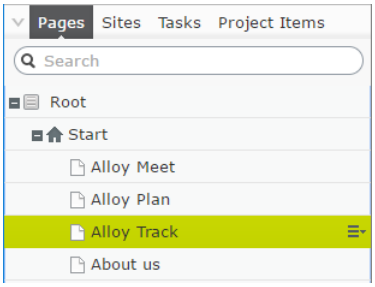
	Alloy Meet	Alloy Plan	Alloy Track
Unique selling points	Project tracking, White board sketch, Built in reminders, Share meeting results, Email interface to request meetings	Project planning, Reporting and statistics, Email handling of tasks, Risk calculations, Direct communication to members	Shared timeline, Project emails, To-do lists, Workflows, Status reports

Page description	You've never had a meeting like this before!	Project management has never been easier!	Projects have a natural lifecycle with well-defined stages.
Main body	Participants from remote locations appear in your meeting room, around your table, or stand presenting at your white board.	Planning is crucial to the success of any project. Alloy Plan takes into consideration all aspects of project planning; from well-defined objectives to staffing, capital investments and management support. Nothing is left to chance.	From start-up meetings to final sign-off, we have the solutions for today's market-driven needs. Leverage your assets to the fullest through the combination of Alloy Plan, Alloy Meet and Alloy Track.

3. Set **PageImage** for each product page to the appropriate media asset, and publish the pages, as shown in the following screenshot:



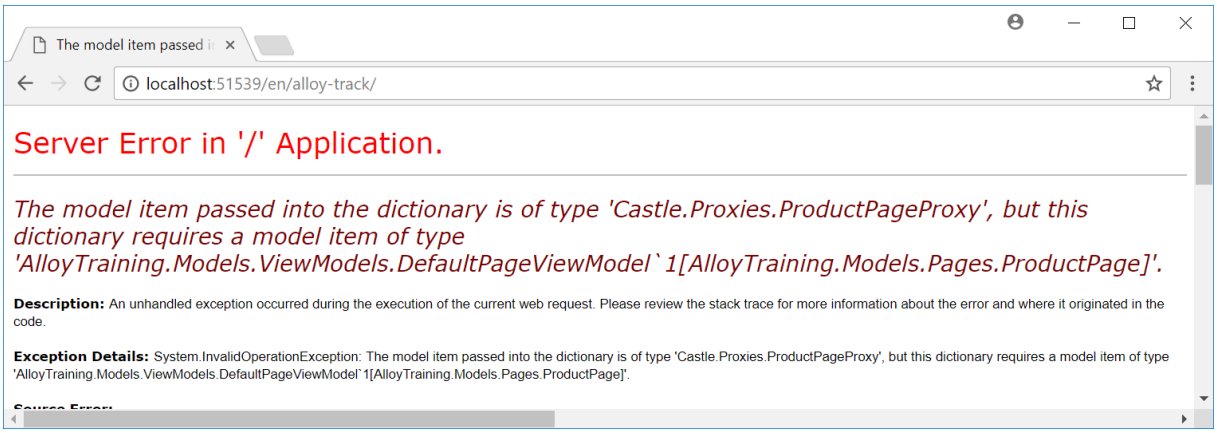
4. In **Navigation** pane, on **Pages** tab, drag and drop the product pages to order them alphabetically, as shown in the following screenshot:



Although we could set the Start page to sort its children alphabetically, that would put About us first.

Optional: Understanding common exceptions

If you see an exception like the following:



...it is because you are passing the wrong object into the view.

For example, in the following method, the code passes **currentPage** instead of **viewmodel** into the view:

```
public ActionResult Index(ProductPage currentPage)
{
    var viewmodel = new DefaultPageViewModel<ProductPage>(currentPage);
    return View(currentPage);
}
```

Creating a partial view for the navigation menu

To create a partial view that can automatically generate a navigation menu, we first need to create an extension method that gets the child pages of the site's Start page, and then call it in the partial view.

1. In **AlloyTraining**, right-click **~\Business\ExtensionMethods**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **StartPageHtmlHelperExtensions.cs** for the name, and click **Add**.
3. Modify the class, to define a **GetStartPage()** and a **GetStartPageChildren()** method, as shown in the following code:

```
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using AlloyStartPage = AlloyTraining.Models.Pages.StartPage;

namespace AlloyTraining.Business.ExtensionMethods
{
    public static class StartPageHtmlHelperExtensions
    {
        public static AlloyStartPage GetStartPage(
            this HtmlHelper html)
        {
            var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
            return loader.Get<AlloyStartPage>(ContentReference.StartPage);
        }

        public static IEnumerable<PageData> GetStartPageChildren(
            this HtmlHelper html)
        {
```

```
            var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
            var children = loader.GetChildren<PageData>(
                ContentReference.StartPage);
            return FilterForVisitor.Filter(children).Cast<PageData>()
                .Where(page => page.VisibleInMenu);
        }
    }
}
```

FilterForVisitor removes unpublished pages, pages that the current visitor does not have Read access rights to, and pages without a template. You will learn more about this class in Module E.

4. In **AlloyTraining**, right-click **~\Views\Shared**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.

Make sure you add the file to **~\Views\Shared**, and NOT to **~\Views\Shared\Layouts**.

5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **_NavigationMenu.cshtml** for the Name, and click **Add**.
6. Modify the view, as shown in the following markup:

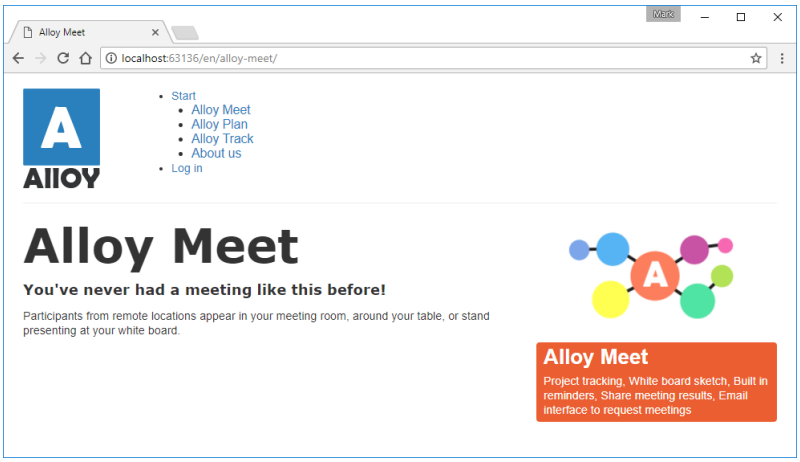
```
@using AlloyTraining.Business.ExtensionMethods
@using AlloyTraining.Models.Pages
@using AlloyTraining.Models.ViewModels
@using EPiServer.Core
@model IPageViewModel<SitePageData>
<ul>
    <li>
        @Html.ContentLink(ContentReference.StartPage)
        <ul>
            @foreach (PageData page in Html.GetStartPageChildren())
            {
                <li>@Html.ContentLink(page.ContentLink)</li>
            }
        </ul>
    </li>
    <li>
        @if (User.Identity.IsAuthenticated)
        {
            <a href="/en/logout">Log out @User.Identity.Name</a>
        }
        else
        {
            <a href="@FormsAuthentication.LoginUrl?ReturnUrl=
            @Model.CurrentPage.PageLink.ExternalURLFromReference()">Log in</a>
        }
    </li>
</ul>
```

Updating the root layout to use the navigation menu

1. Open **~\Views\Shared\Layouts_Root.cshtml**.
2. Inside the `<div class="span10">`, delete the `@if` statement that outputs the log in/out hyperlinks, and replace it with a call to output the navigation menu partial view, as shown in the following markup:

```
<div class="span10">
    @Html.Partial("_NavigationMenu")
</div>
```

3. Start the **AlloyTraining** website, and use the menu to navigate between pages, as shown in the following screenshot:



Module C – Rendering Content Templates

Goal

The overall goal of the exercises in this module is to see how you can customize the experience for visitors. You will:

1. Implement a content area property on the Start page and create a partial page template for Product pages to enable them to be rendered in the content area.
2. Create a partial template for all pages to enable them to be rendered in a content area.
3. Define display options to allow content editors to apply tags to select between multiple templates.
4. Apply tags programmatically to allow developers to apply tags to select between multiple templates
5. Create a display channel to set a tag automatically based on information in an incoming request to select between multiple templates.

Exercise C1 – Creating partial templates for product pages and image files for use in content areas

In this exercise, you will create a content area on the start page and add pages (and later blocks) to it.

Prerequisites: complete Exercises B1 to B4.

Adding a content area to the Start page type and template

The content area on the Start page will allow only blocks or standard pages to be included.

1. In **AlloyTraining**, open **~\Models\Pages\StartPage.cs**.
2. Add a **ContentArea** property named **MainContentArea** with appropriate attributes, as shown in the following code:

MainContentArea must only allow content references to: Standard pages, blocks, images, and folders.

```
[CultureSpecific]
[Display(Name = "Main content area",
    Description = "Drag and drop images, blocks, folders, and pages with
partial templates.",
    GroupName = SystemTabNames.Content,
    Order = 30)]
[AllowedTypes(typeof(StandardPage),
    typeof(BlockData), typeof(ImageData), typeof(ContentFolder))]
public virtual ContentArea MainContentArea { get; set; }
```

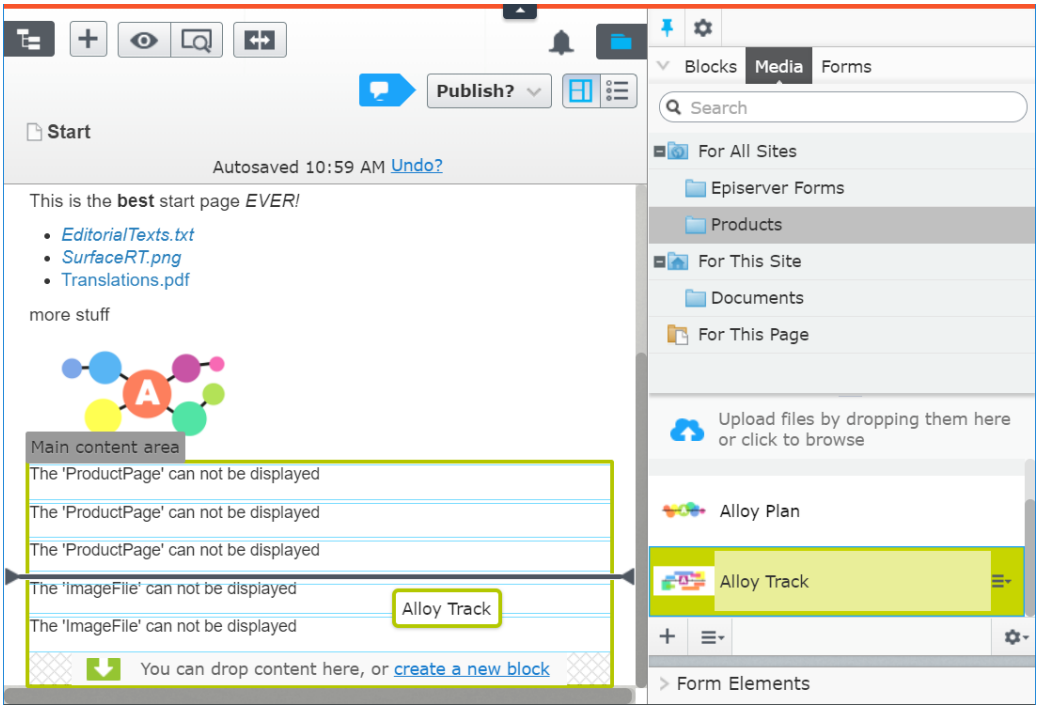
3. In **AlloyTraining**, open **~\Views\StartPage\Index.cshtml**.
4. At the bottom of the view, output the **MainContentArea** property so that editors get an on-page editing experience, and on the wrapper `<div>` for each content item set Bootstrap classes of **row** and **equal-height**, as shown in the following markup:

```
@Html.PropertyFor(m => m.CurrentPage.MainContentArea,
    new { CssClass = "row equal-height" })
```

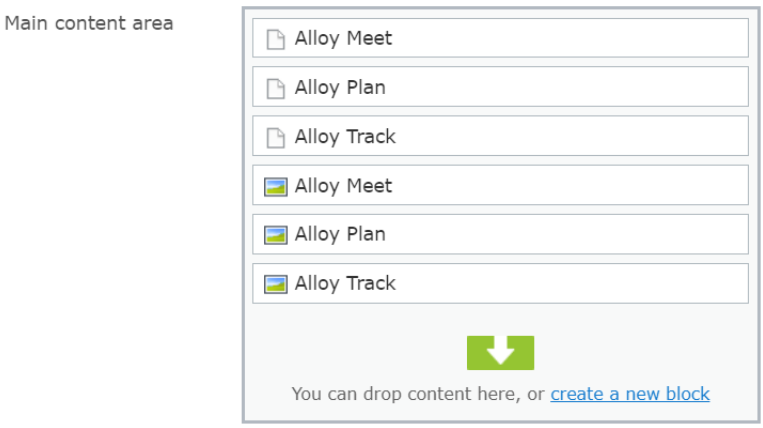
Testing the content area

1. Start the **AlloyTraining** website, and log in as a CMS admin.

2. In **Edit** view, open **Navigation** | **Pages**, drag and drop the three product pages into the **Main content area**, and note the warning message "The 'ProductPage' cannot be displayed".
3. Open **Assets** | **Media**, drag and drop the three product images from the **Products** folder into the **Main content area**, and note the warning message, "The 'ImageFile' cannot be displayed.", as shown in the following screenshot:



4. In **All Properties** view, **Main content area** looks like the following screenshot:



5. **Publish** the changes to the **Start** page.

If you try to drag and drop the **Start** page into the content area it is not allowed because the **MainContentArea** only allows **StandardPage**, **BlockData**, **ContentFolder**, or **ImageData** content.

6. Pull down the **Global** menu and switch to visitor view. Note that instead of showing the warning message, visitors see nothing.

You will fix this issue by creating a partial page template for product pages and a template view for images.

Creating a partial content controller for product pages

1. In **AlloyTraining**, expand **Controllers**, and copy and paste the **ProductPageController.cs** file.

2. Rename the copy to **ProductPagePartialController.cs**.
3. Open **ProductPagePartialController.cs**.
4. Import the `EPiServer.Web.Mvc` namespace.
5. Rename the class to **ProductPagePartialController**.
6. Change the class to inherit from `PartialContentController<ProductPage>`.
7. Add the `override` keyword to the **Index** action method.
8. Modify the `return` statement to use **PartialView**.

Using **PartialView** means it won't execute **_ViewStart.cshtml**, because partials do not need a layout.

The class should now look something like this:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using EPiServer.Web.Mvc;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class ProductPagePartialController
        : PartialContentController<ProductPage>
    {
        public override ActionResult Index(ProductPage currentPage)
        {
            var viewmodel = new DefaultPageViewModel<ProductPage>(currentPage);
            return PartialView(viewmodel);
        }
    }
}
```

Make sure that you call **PartialView**() method, not **View**() method, or the layout will render multiple times!

Creating a partial view for product pages

1. In **AlloyTraining**, right-click **Views**, and add a new folder named **ProductPagePartial**.
2. Expand **~\Views\ProductPage**, and copy the **Index.cshtml** file into the folder **ProductPagePartial**.
3. Open **~\Views\ProductPagePartial\Index.cshtml**.
4. Modify the contents, as shown in the following markup, and note the following:
    - The markup is like the "full" view, but it outputs only four properties: **Name**, **MetaDescription**, **UniqueSellingPoints**, and **PageImage**. This is a subset of the content.
    - The property output is wrapped in a with a border, wrapped in a with a Bootstrap **span4** class. This will allocate one third of a Bootstrap **row** width to each product.
    - The whole product is wrapped in a clickable hyperlink that would navigate the visitor to the "full" product page.

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<ProductPage>
<div class="block span4">
    <div class="border">
        <a href="@Url.ContentUrl(Model.CurrentPage.ContentLink)">
            <h2 @Html.EditAttributes(x => x.CurrentPage.Name)>
                @Model.CurrentPage.Name</h2>
```

```
            <p class="introduction"
               @Html.EditAttributes(x => x.CurrentPage.MetaDescription)>
               @Model.CurrentPage.MetaDescription</p>
            <div>
                @foreach(string usp in
                    Model.CurrentPage.UniqueSellingPoints.Split(','))
                {
                    <small class="label label-info" style="color:white;">@usp</small>
                }
            </div>
            <div @Html.EditAttributes(x => x.CurrentPage.PageImage)>
                <img src="@Url.ContentUrl(Model.CurrentPage.PageImage)" />
            </div>
        </a>
    </div>
</div>
```

Creating a template view for image files

1. In **AlloyTraining**, right-click **~\Views\Shared**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.

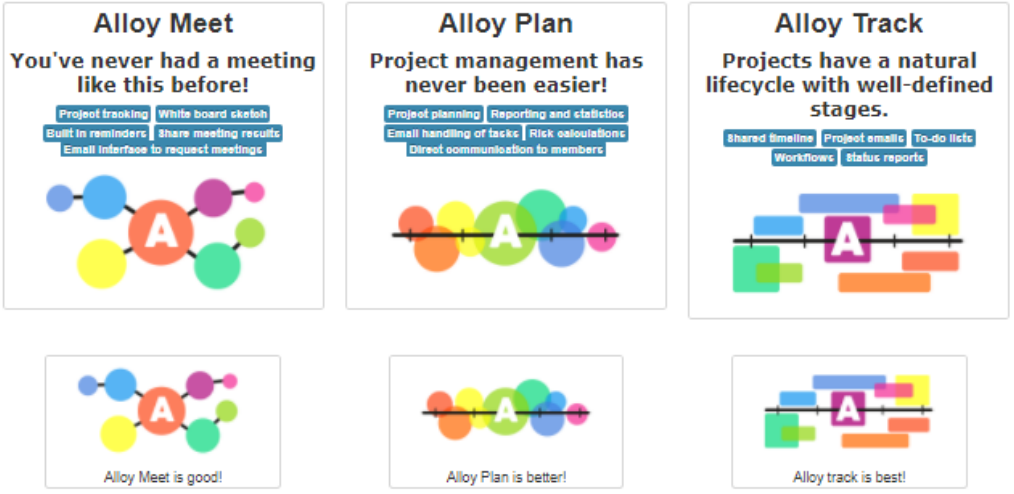
Make sure you add the file to ~\Views\Shared, and NOT to ~\Views\Shared\Layouts.

2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **ImageFile.cshtml** for the Name, and click **Add**.
3. Modify the contents, as shown in the following markup:

```
@using EPiServer.Web.Mvc.Html
@model AlloyTraining.Models.Media.ImageFile

<div class="block span4">
    <figure class="border">
        <img src="@Url.ContentUrl(Model.ContentLink)"
             alt="@Model.Description" />
        <figcaption>@Model.Description</figcaption>
    </figure>
</div>
```

Testing the partial page and image templates

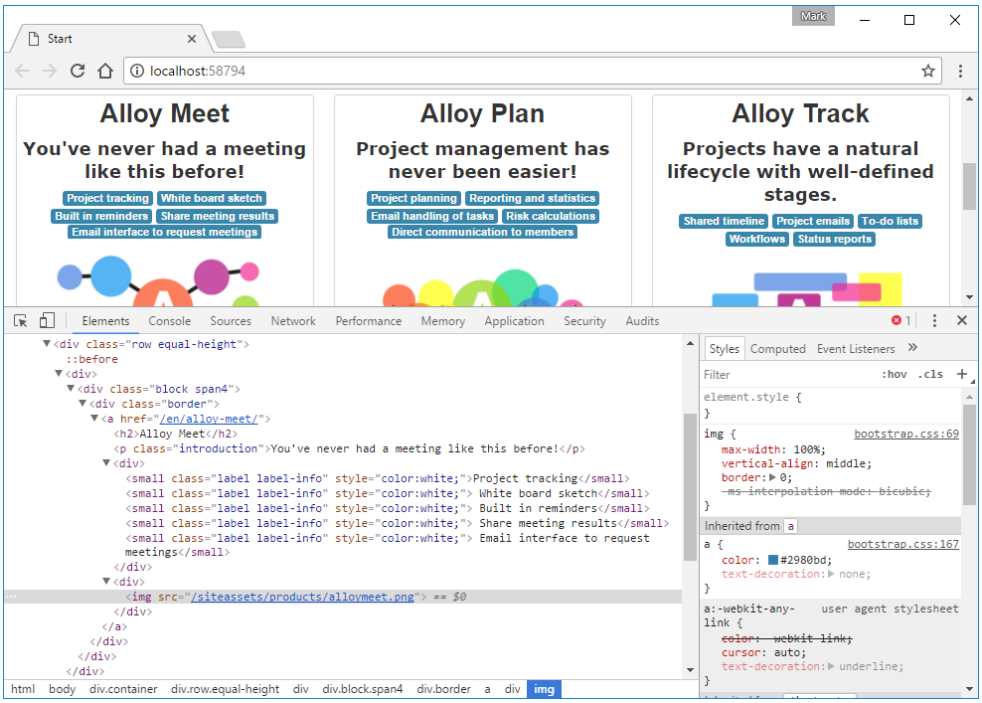
1. Start the **AlloyTraining** website, and note the visitor's view of the content area with three product pages and three product images rendered by their partial content templates:



2. Click each partial product page to confirm that it links to the correct full product page.

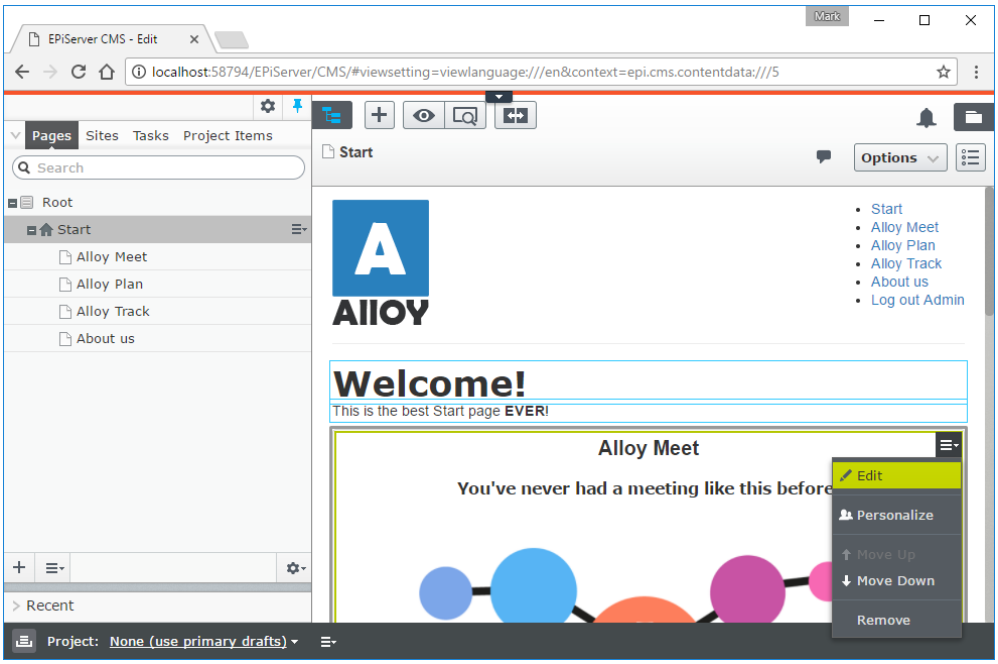
3. Right-click one of the product images, click **Inspect**, and note the following, as shown in the following screenshot:

    - A `<div>` element with Bootstrap class of `row` and `equal-height` that contains three `<div>` elements that were generated for each reference to a product page in the content area.
    - Three `<div>` elements with Bootstrap class of `block` and `span4` that were output by the partial view.
    - The URL used for the `src` attribute of the product images.



4. Log in as a CMS admin.

5. Due to using Bootstrap the output is responsive, and an editor can edit a product page using the partial page template and its context menu directly from the Start page, as shown in the following screenshot:



Creating a partial content template for folders

We might want to be able to add a reference to a folder inside a content area and see its name and how many items are in that folder.

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Controller (MVC)**, enter **ContentFolderController.cs** for the name, and click **Add**.
3. Fix the compilation error by changing the base class from `BlockController` to `PartialContentController`.
4. Modify the **Index** action method to use a parameter named **currentContent**, as shown in the following code:

```
using EPiServer.Core;
using EPiServer.Web.Mvc;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class ContentFolderController : PartialContentController<ContentFolder>
    {
        public override ActionResult Index(ContentFolder currentContent)
        {
            return PartialView(currentContent);
        }
    }
}
```

5. In **AlloyTraining**, right-click **~\Business\ExtensionMethods**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
6. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **ContentFolderExtensions.cs** for the Name, and click **Add**.
7. Modify the class, to define a **GetItemCount()** method, as shown in the following code:

```
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using System;
using System.Linq;

namespace AlloyTraining.Business
{
    public static class ContentFolderExtensions
    {
        public static int GetItemCount(this ContentFolder folder)
        {
            using (var cache = new ContentCacheScope
                                    { SlidingExpiration = TimeSpan.Zero })
            {
                var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
                var items = loader.GetChildren<IContent>(folder.ContentLink);
                return items.Count();
            }
        }
    }
}
```

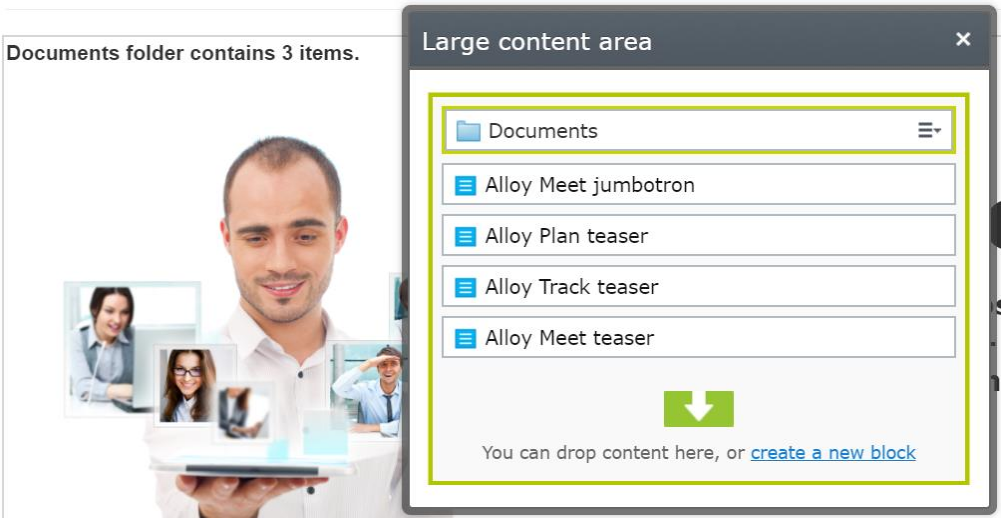
This extension method uses the `ContentCacheScope` class to prevent the content items that are loaded from staying in the object cache for the default of 12 hours to reduce memory pressure.

8. In **AlloyTraining**, right-click **Views**, and add a new folder named **ContentFolder**.
9. Right-click **ContentFolder**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
10. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
11. Modify the file, as shown in the following markup:

```
@using AlloyTraining.Business.ExtensionMethods
@model EPiServer.Core.ContentFolder
<h4>@Model.Name folder contains @Model.GetItemCount() items.</h4>
```

Testing the partial content template for folders

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **Start** page, and drag and drop the **Documents** folder from the **Assets** pane into the main content area, and note the name and number of items is displayed, as shown in the following screenshot:



3. Publish the **Start** page.

Exercise C2 – Creating a partial template for all pages

In this exercise, you will create partial page templates to enable any site page to render inside a content area.

Page templates can be defined for base classes and then inherited by any page type that derives from that base class. You will:

- Define a partial page template for all pages to render the page Name and MetaDescription in a "full" 3/3 width view with yellow background colour.
- Define a partial page template for all pages to render the page Name and MetaDescription in a "wide" 2/3 width view with pink background colour.
- Define a partial page template for all pages to render the page Name and MetaDescription in a "narrow" 1/3 width view with green background colour.
- Define **SiteTags** class with string constants for the three tag values: "Full", "Wide", and "Narrow".

Prerequisites: complete Exercises B1 to B4, C1.

Creating a partial content controller for all pages

1. In **AlloyTraining**, right-click **AlloyTraining**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SiteTags.cs** for the name, and click **Add**.
3. Modify the file, as shown in the following code:

```
namespace AlloyTraining
{
    public static class SiteTags
    {
        public const string Full = "full";
        public const string Wide = "wide";
        public const string Narrow = "narrow";
    }
}
```


These string constants will be used to apply tags to content templates.

4. In **AlloyTraining**, expand **Controllers**, and copy and paste the **ProductPagePartialController.cs** file.
5. Rename the copy to **AllPagesPartialController.cs**.
6. Open **AllPagesPartialController.cs**.
7. Rename the class to `AllPagesFullPartialController`.
8. Change all references from `ProductPage` to `SitePageData`.
9. Apply `TemplateDescriptor` attribute to mark this class as allowing page template inheritance.
10. Modify the call to `PartialView` to pass in the name of a view: `SiteTags.Full`

The class should now look something like this:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using EPiServer.Web.Mvc;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    [TemplateDescriptor(Inherited = true,
        Tags = new[] { SiteTags.Full }, AvailableWithoutTag = true)]
    public class AllPagesFullPartialController
```

```
        : PartialContentController<SitePageData>
    {
        public override ActionResult Index(SitePageData currentPage)
        {
            var viewmodel =
                new DefaultPageViewModel<SitePageData>(currentPage);
            return PartialView(viewName: SiteTags.Full, model: viewmodel);
        }
    }
}
```

When you explicitly specify a view name, the search path looks for that named view in **~\Views\Shared**, i.e. **~\Views\Shared\Full.cshtml**

Creating additional partial page templates for "wide" and "narrow"

1. Open **AllPagesPartialController.cs**.
2. Inside the namespace, copy the entire class and its attribute to the clipboard.
3. Paste twice to create two copies of the class.
    a. Rename the first copy to: **AllPagesWidePartialController**
    b. Rename the second copy to: **AllPagesNarrowPartialController**
4. Change the `TemplateDescriptor` attributes of the two copies to set a Tag named "wide" or "narrow" and make the two copies only available if the tag is set.
5. Explicitly pass the name of a partial view to use instead of Full: Wide or Narrow.

Your two copied classes should look something like the following code:

```
[TemplateDescriptor(Inherited = true,
     Tags = new[] { SiteTags.Wide }, AvailableWithoutTag = false)]
public class AllPagesWidePartialController
    : PartialContentController<SitePageData>
{
    public override ActionResult Index(SitePageData currentPage)
    {
        var viewmodel = new DefaultPageViewModel<SitePageData>(currentPage);
        return PartialView(viewName: SiteTags.Wide, model: viewmodel);
    }
}

[TemplateDescriptor(Inherited = true,
    Tags = new[] { SiteTags.Narrow }, AvailableWithoutTag = false)]
public class AllPagesNarrowPartialController
    : PartialContentController<SitePageData>
{
    public override ActionResult Index(SitePageData currentPage)
    {
        var viewmodel = new DefaultPageViewModel<SitePageData>(currentPage);
        return PartialView(viewName: SiteTags.Narrow, model: viewmodel);
    }
}
```

Creating partial views for all pages

1. Expand **~\Views\ProductPage**, and copy the **Index.cshtml** file into the folder **~\Views\Shared**.
2. Rename it to **Full.cshtml**.
3. Open **~\Views\Shared\Full.cshtml**.
4. Modify the contents, as shown in the following markup, and note the following:

- `DefaultPageViewModel<T>` now uses a `SitePageData` instead of `ProductPage`, so that the template will work with any type of page on the site.
- It has a Bootstrap class of `span12` for "full" width.
- It has a style that sets the background color to light yellow.
- It outputs three properties in this order vertically: `Name`, `MetaDescription`, and `PageImage`.

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<SitePageData>
<div class="block span12" style="background-color: lightyellow;">
    <div class="border">
        <a href="@Url.ContentUrl(Model.CurrentPage.ContentLink)">
            <h2 @Html.EditAttributes(x => x.CurrentPage.Name)>
                @Model.CurrentPage.Name</h2>
            <p class="introduction"
                @Html.EditAttributes(x => x.CurrentPage.MetaDescription)>
                @Model.CurrentPage.MetaDescription</p>
            <div @Html.EditAttributes(x => x.CurrentPage.PageImage)>
                <img src="@Url.ContentUrl(Model.CurrentPage.PageImage)" />
            </div>
        </a>
    </div>
</div>
```

5. Save and close **Full.cshtml**.
6. Copy and paste **Full.cshtml** twice:
   - Rename the first copy: **Wide.cshtml**
   - Rename the second copy: **Narrow.cshtml**
7. Open **Wide.cshtml**.
8. Modify the contents, as shown in the following markup, and note the following:
   - It has a Bootstrap class of **span8** for "wide" 2/3 width.
   - It has a style that sets the background color to light pink (lavenderblush).
   - It outputs two properties in this order vertically: **PageImage** and **Name**.

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<SitePageData>
<div class="block span8" style="background-color: lavenderblush;">
    <div class="border">
        <a href="@Url.ContentUrl(Model.CurrentPage.ContentLink)">
            <div @Html.EditAttributes(x => x.CurrentPage.PageImage)>
                <img src="@Url.ContentUrl(Model.CurrentPage.PageImage)" />
            </div>
            <h2 @Html.EditAttributes(x => x.CurrentPage.Name)>
                @Model.CurrentPage.Name</h2>
        </a>
    </div>
</div>
```

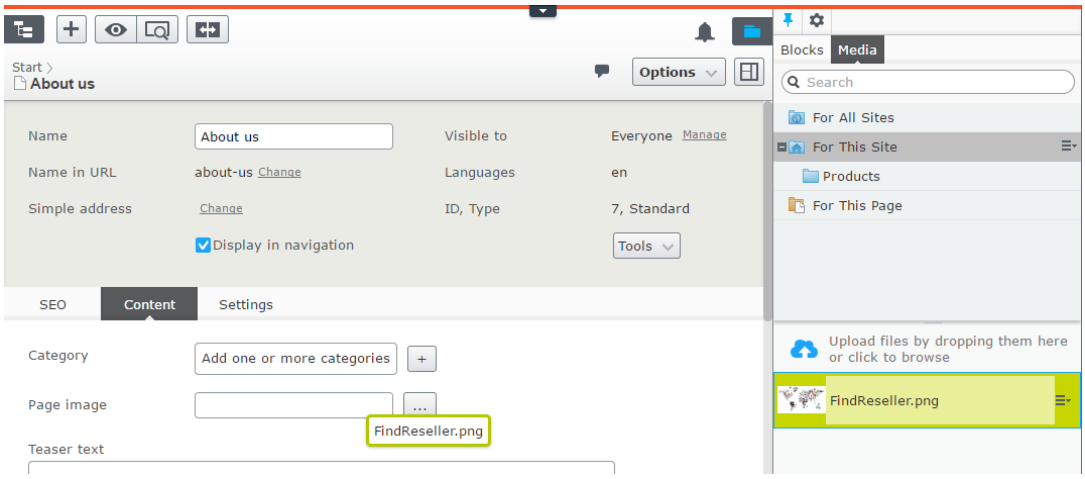
9. Open **Narrow.cshtml**.
10. Modify the contents, as shown in the following markup, and note the following:
    - It has a Bootstrap class of **span4** for "narrow" 1/3 width.
    - It has a style that sets the background color to pale green.

- It outputs two properties in this order vertically: **Name** and **MetaDescription**.

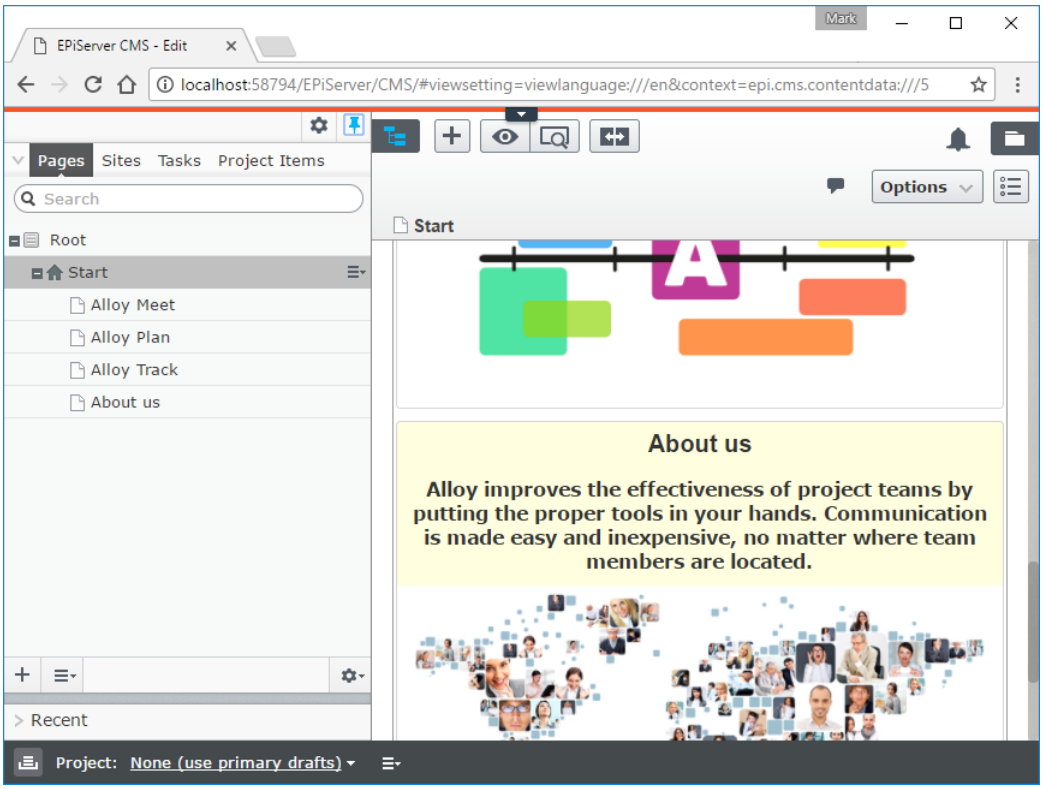
```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<SitePageData>
<div class="block span4" style="background-color: palegreen;">
    <div class="border">
        <a href="@Url.ContentUrl(Model.CurrentPage.ContentLink)">
            <h2 @Html.EditAttributes(x => x.CurrentPage.Name)>
                @Model.CurrentPage.Name</h2>
            <p class="introduction"
                @Html.EditAttributes(x => x.CurrentPage.MetaDescription)>
                @Model.CurrentPage.MetaDescription</p>
        </a>
    </div>
</div>
```

Testing the partial page template

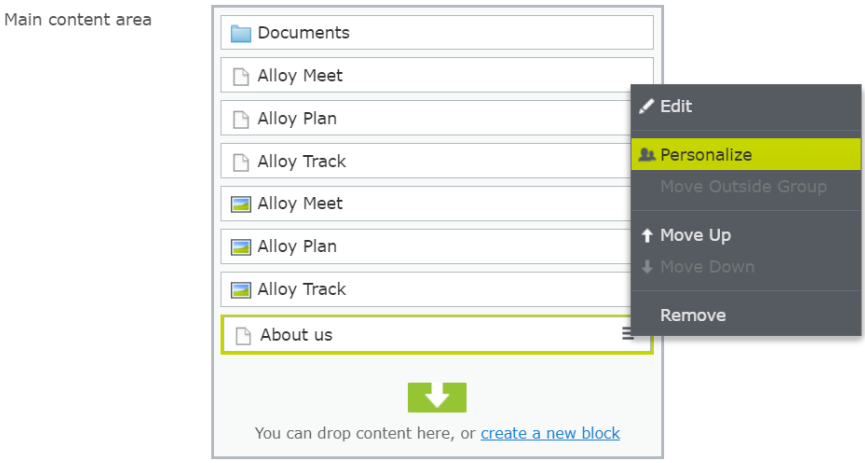
1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **About us** page, and switch to **All Properties** view.
3. In **Assets** pane, click **Media**.
4. Upload the image **~\Assets\Misc\FindReseller.png** to the folder **For This Site**.
5. Drag and drop the image **FindReseller.png** to the **Page image** property of the **About us** page, as shown in the following screenshot:



6. Publish the change.
7. Edit the **Start** page.
8. Drag and drop the **About us** page into the main content area, as shown in the following screenshot, and note that it uses the “full” partial page template:



9. Switch to **All Properties** view.
10. Select the context menu for the reference to the **About us** page, and note there are no display options to assign alternative tags, as shown in the following screenshot:



In the next exercise, you will allow the content editor to choose display options to apply one of three "tags" to switch between the three partial page templates.

Exercise C3 – Enabling editors to apply tags manually using display options

In this exercise, you will define three display options to alloy an editor to apply tags manually to individual content items in a content area to switch between partial page templates.

Prerequisites: complete Exercises B1 to B4, C1 and C2.

Adding display options using an initialization module

Display options for applying tags to content references should be registered during initialization.

1. In **AlloyTraining**, right-click **Business**, and add a folder named **Initialization**.
2. In **~\Business\Initialization**, add an **Episerver** | **Initialization Module** named **DisplayOptionsInitializationModule.cs**.
3. Modify the class, as shown in the following code:

```
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.Web;

namespace AlloyTraining.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class DisplayOptionsInitializationModule : IInitializableModule
    {
        public void Initialize(InitializationEngine context)
        {
            var options =
                context.Locate.Advanced.GetInstance<DisplayOptions>();

            options.Add(id: SiteTags.Full, name: "Full", tag: SiteTags.Full);
            options.Add(id: SiteTags.Wide, name: "Wide", tag: SiteTags.Wide);
            options.Add(
                id: SiteTags.Narrow, name: "Narrow", tag: SiteTags.Narrow);
        }

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

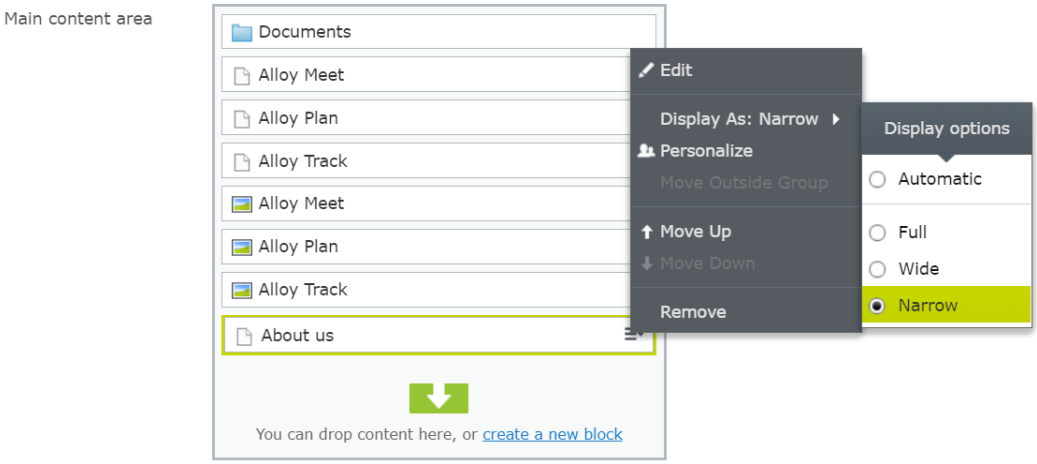
4. In **~\Resources\LanguageFiles**, add an XML file named **DisplayOptions.xml**.
5. Modify the file, as shown in the following markup:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<languages>
  <language name="English" id="en">
    <displayoptions>
      <full>Full</full>
      <wide>Wide</wide>
      <narrow>Narrow</narrow>
    </displayoptions>
  </language>
  <language name="Swedish" id="sv">
    <displayoptions>
      <full>Full</full>
      <wide>Bred</wide>
      <narrow>Smal</narrow>
    </displayoptions>
  </language>
```

```
</languages>
```

Testing display options

1. Start the **AlloyTraining** website and log in as a CMS admin.
2. Edit the **Start** page and switch to **All Properties** view.
3. Click **Content** tab and scroll down to the **Main content area** property.
4. Select **About us**, and click the context menu to choose a **Display As** option, for example **Narrow**, as shown in the following screenshot:



5. Publish the change and note the partial page template used to render the **About us** page has been changed to use the **Narrow** template (that doesn't show the image and it has a green background).
6. Try applying the **Wide** display option, and note it renders the image above the title, without a description, as shown in the following screenshot:

7. Publish the page.
8. Close the browser.

Display As options are shown for all content references in a content area, even if they would have no affect. For example, the references to the three images show the same **Display As** options, but the images do not have different templates. The references to the three product pages can have the display options applied because `ProductPage` inherits indirectly from `SitePageData`. It is possible to filter display options based on the content type of the current selection, but that is an advanced technique not covered in this course.

Exercise C4 – Applying tags to content areas with code

In this exercise, you will create two content areas on the product page and then control which partial page template is used by programmatically applying a tag: full, wide, or narrow.

Prerequisites: complete Exercises B1 to B4, C1 to C2.

Adding a content area to the Product page type and template

1. In **AlloyTraining**, open **~\Models\Pages\ProductPage.cs**.
2. Add two **ContentArea** properties with appropriate attributes: **MainContentArea** and **RelatedContentArea**, as shown in the following code:

```
[Display(Name = "Main content area",
    Description = "Drag and drop blocks and pages with partial templates.",
    GroupName = SystemTabNames.Content,
    Order = 330)]
public virtual ContentArea MainContentArea { get; set; }

[Display(Name = "Related content area",
    Description = "Drag and drop blocks and pages with partial templates.",
    GroupName = SystemTabNames.Content,
    Order = 340)]
public virtual ContentArea RelatedContentArea { get; set; }
```

3. In **AlloyTraining**, open **~\Views\ProductPage\Index.cshtml**.
4. Import the **AlloyTraining** namespace, as shown in the following markup:

```
@using AlloyTraining
```

5. Above `@section RelatedContent`, render the **MainContentArea** property so that editors get an on-page editing experience, set Bootstrap classes of **row** and **equal-height**, and set the wide site tag, as shown in the following markup:

```
@Html.PropertyFor(m => m.CurrentPage.MainContentArea,
    new { CssClass = "row equal-height", Tag = SiteTags.Wide })
```

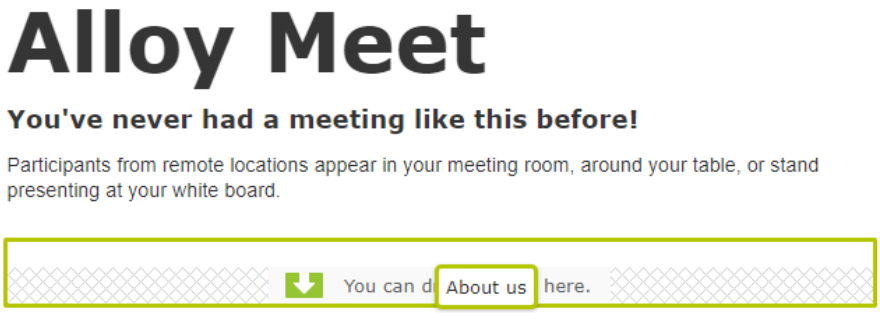
6. At the bottom of `@section RelatedContent`, before the close brace, render the **RelatedContentArea** property so that editors get an on-page editing experience, set Bootstrap classes of **row** and **equal-height**, and set the narrow site tag, as shown in the following markup:

```
@Html.PropertyFor(m => m.CurrentPage.RelatedContentArea,
    new { CssClass = "row equal-height", Tag = SiteTags.Narrow })
```

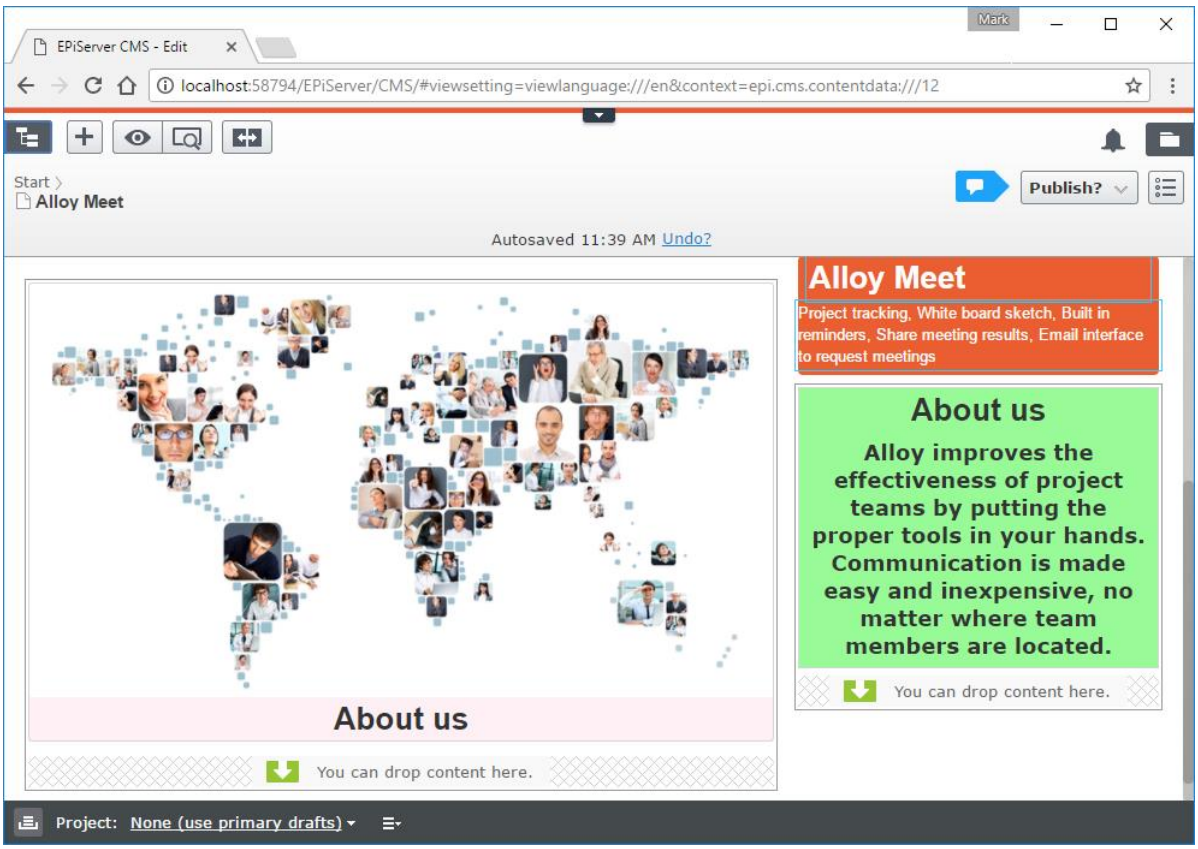
Testing the content areas on product pages

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit one of the product pages, for example, **Alloy Meet**.

3. Drag and drop **About us** from the **Navigation** pane's **Pages** tree, into the **Main content area**, as shown in the following screenshot:



4. Drag and drop **About us** from the **Navigation** pane's **Pages** tree, into the **Related content area**, and note that different partial page templates are used (Wide.cshtml and Narrow.cshtml) due to the tags applied programmatically, as shown in the following screenshot:



5. Publish the page.
6. Close the browser.

Exercise C5 – Optional: Applying tags automatically using a display channel

In this exercise, you will create a page template for Start page optimized for mobile devices and a display channel that automatically sets the mobile tag based on information in an incoming HTTP request.

Prerequisites: complete Exercises B1 – B4.

Creating a mobile page template controller and view

1. In **AlloyTraining**, open **~\Controllers**.
2. Copy and paste the **StartPageController.cs** file.
3. Rename the copy to **StartPageMobileController.cs**, and open it.
4. Import the **EPiServer.Framework.Web** namespace, as shown in the following code:

```
using EPiServer.Framework.Web;
```

5. Apply an attribute to ensure this controller is only used when the "mobile" rendering tag is set, as shown in the following code:

```
[TemplateDescriptor(Tags = new[] { RenderingTags.Mobile },
    AvailableWithoutTag = false)]
public class StartPageMobileController : PageControllerBase<StartPage>
```

Episerver defines the **RenderingTags** class with a **Mobile** string constant value of "mobile".

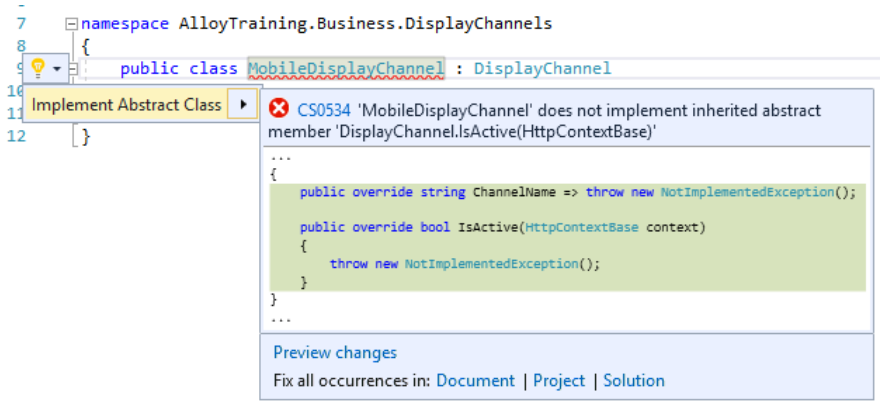
6. In **AlloyTraining**, right-click **Views**, and add a new folder named **StartPageMobile**.
7. Expand **~\Views\StartPage**, and copy the **Index.cshtml** file into the folder **StartPageMobile**.
8. Open **~\Views\StartPageMobile\Index.cshtml**.
9. At the bottom of the view, delete the outputting of the **MainContentArea** using **PropertyFor**, and replace it with an enumeration of the filtered items in the content area that outputs a link to each page, as shown in the following markup:

```
<ul>
    @foreach(var item in Model.CurrentPage.MainContentArea.FilteredItems)
    {
        <li>@Html.ContentLink(item.ContentLink)</li>
    }
</ul>
```

FilteredItems removes any content references that the current visitor should not be able to see.

Creating a display channel

1. In **AlloyTraining**, right-click **~\Business**, and add a new folder named **DisplayChannels**.
2. Right-click **~\Business\DisplayChannels**, and add a new class named **MobileDisplayChannel**.
3. Import the **EPiServer.Web** namespace.
4. Make the class inherit from **DisplayChannel**.
5. Use Visual Studio to implement the abstract class, as shown in the following screenshot:



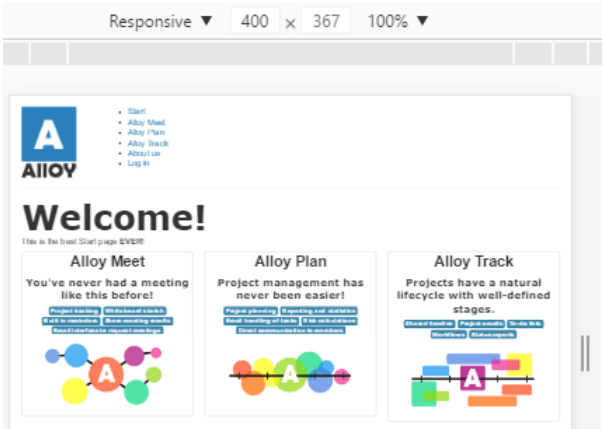
6. For the **ChannelName** property, return the mobile rendering tag.
7. For the **IsActive** method, return if the current request is from a mobile device.

Your completed class should look something like the following code:

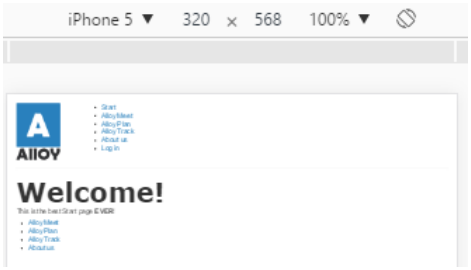
```
using EPiServer.Framework.Web;
using EPiServer.Web;
using System.Web;

namespace AlloyTraining.Business.DisplayChannels
{
    public class MobileDisplayChannel : DisplayChannel
    {
        // C# 6.0 syntax for a read-only property
        public override string ChannelName => RenderingTags.Mobile;

        public override bool IsActive(HttpContextBase context)
        {
            return context.Request.Browser.IsMobileDevice;
        }
    }
}
```

Testing the display channel

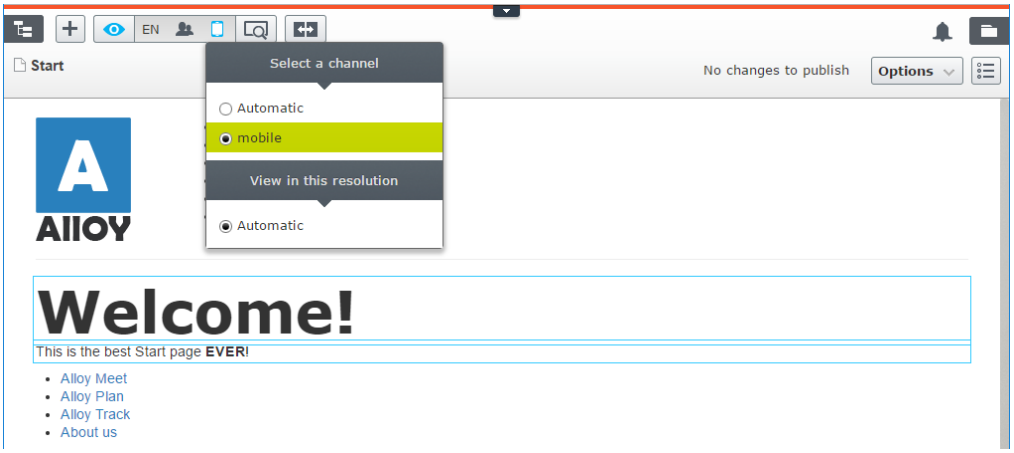
1. Start the **AlloyTraining** website.
2. Press *F12* to show developer tools.
3. In Chrome's developer tools pane, click **Toogle device toolbar**, or press *Ctrl* + *Shift* + *M*, and note that by default Chrome shows a Responsive page, as shown in the following screenshot:



4. In the device toolbar, choose **iPhone 5**, click **Refresh** or press *F5*, and note Chrome now shows the "mobile" page template response with links instead of partial content, as shown in the following screenshot:



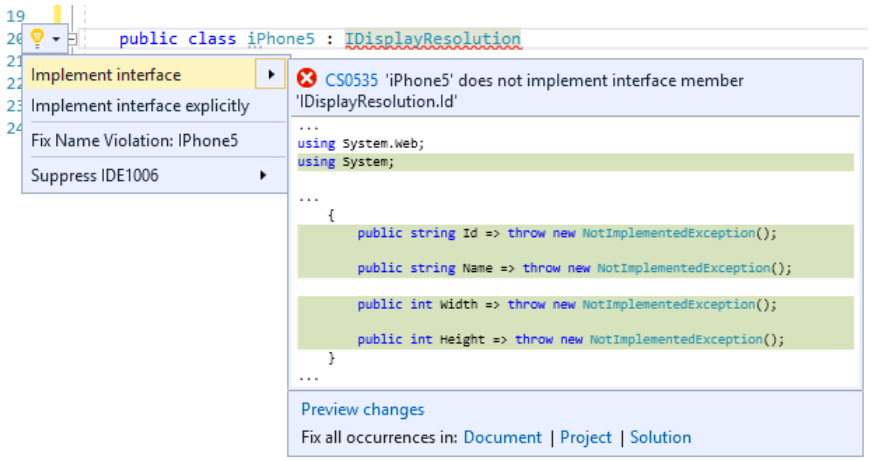
5. Close the developer tools.
6. Log in as a CMS admin.
7. Edit the **Start** page.
8. In the toolbar, click **Toggle view settings**, and select the **mobile** channel, as shown in the following screenshot:



Improving the editors' preview

You can combine a display channel with some common resolutions, and a background image, but this only affects editors preview options. It has no affect at runtime for visitors.

1. In **AlloyTraining**, open **~\Business\DisplayChannels\MobileDisplayChannel.cs**.
2. Define a class named **iPhone5** that implements the **IDisplayResolution** interface, and use Visual Studio to implement the interface, as shown in the following screenshot:



3. Return appropriate values for the four properties, as shown in the following code:

```
public class iPhone5 : IDisplayResolution
{
    public string Id => "iphone5";
```

```
    public string Name => "iPhone 5 (320 x 568)";
    public int Width => 320;
    public int Height => 568;
}
```

4. Copy and paste the class for an **iPhone4** resolution, as shown in the following code:

```
public class iPhone4 : IDisplayResolution
{
    public string Id => "iphone4";
    public string Name => "iPhone 4 (320 x 480)";
    public int Width => 320;
    public int Height => 480;
}
```

5. In **MobileDisplayChannel**, override the **ResolutionId** property, as shown in the following code:

```
public override string ResolutionId => "iphone5";
```


In **AlloyDemo**, you will see that a base class has been used for the resolutions and strings are retrieved from resource files using **LocalizationService**. This is recommended. In this exercise, we have used a simplified example to highlight the main functionality of **DisplayChannel** and **IDisplayResolution**.

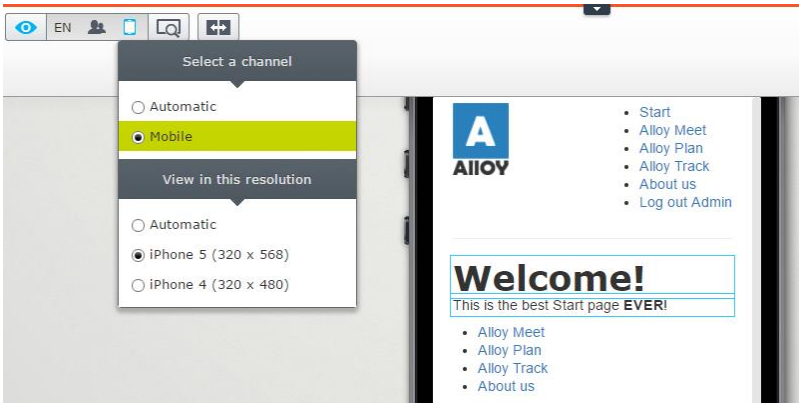
7. Right-click **~\Resources\LanguageFiles**, and add a new **XML** file named **DisplayChannels.xml**.
8. Modify the file, as shown in the following markup:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<languages>
  <language name="English" id="en">
    <displaychannels>
      <displaychannel name="mobile">
        <name>Mobile</name>
      </displaychannel>
    </displaychannels>
    <resolutions>
      <iphone4>iPhone 4 (320 x 480)</iphone4>
      <iphone5>iPhone 5 (320 x 568)</iphone5>
    </resolutions>
  </language>
  <language name="Swedish" id="sv">
    <displaychannels>
      <displaychannel name="mobile">
        <name>Mobil</name>
      </displaychannel>
    </displaychannels>
    <resolutions>
      <iphone4>iPhone 4 (320 x 480)</iphone4>
      <iphone5>iPhone 5 (320 x 568)</iphone5>
    </resolutions>
  </language>
</languages>
```

Testing the improvements

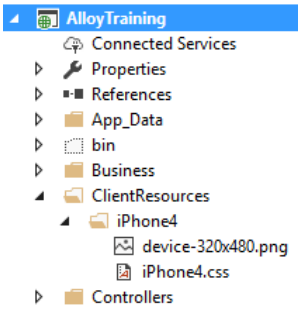
1. Start the **AlloyTraining** website, and log in as a CMS admin.

2. In the page editing toolbar, click **Toogle view settings**, select **Mobile**, and note the default resolution is **iPhone 5**, which has a built-in background image simulating that phone, as shown in the following screenshot:

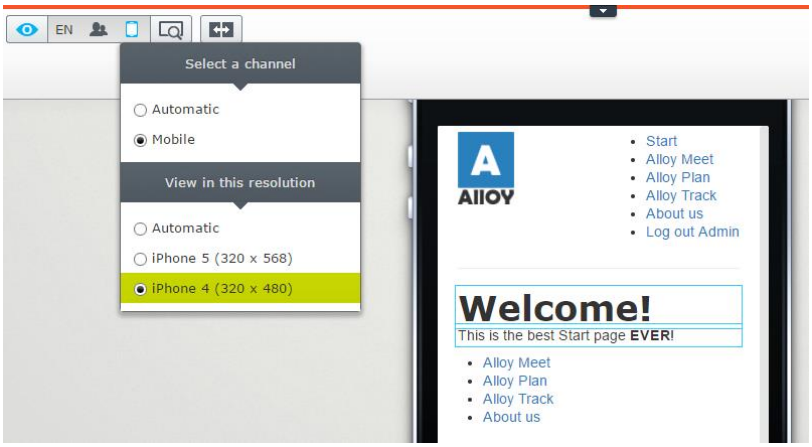


Display channels can only be associated with one resolution, and that association is one-way. This means that when the **Mobile** display channel is selected, the **iPhone 5 (320 x 568)** resolution will be selected by default. But other resolutions must be manually selected. And if an editor leaves the channel set to **Automatic**, and then selects a non-Automatic resolution, the **Mobile** channel will not be selected. For example, if the Editor chooses a combination of **Automatic** and **iPhone 5 (320 x 568)** they will not see the result of having the **Mobile** display channel active.

3. Select **iPhone 4**, and note that although it changes resolution, it does not have a background image.
4. Close the browser.
5. From the solution files, drag and drop the **\cmsdevfun-exercisefiles\Module C\C5\ClientResources** folder and the **\cmsdevfun-exercisefiles\Module C\C5\module.config** file into **AlloyTraining**, as shown in the following screenshot:



6. Start the **AlloyTraining** website, and log in as a CMS admin, and select the **iPhone 4** resolution, as shown in the following screenshot:



7. Close the browser.

Module D – Working with Blocks

Goal

The overall goal of the exercises in this module is to implement working examples of various block types and uses for blocks. You will:

1. Create a controller-less block for efficiency.
2. Create a block with a controller.
3. Create a preview template for blocks and partial pages.
4. Move important block properties to the basic info area.
5. Use a block type as a property type.

Exercise D1 – Creating a controller-less block for editorial content

In this exercise, you will create a block type with a block template that consists of a view without a controller.

Controller-less blocks are more efficient than block templates with controllers, so you should use them whenever possible.

Prerequisites: complete Exercises B1 to B4, C1 to C4.

If you have not completed Exercises C1 to C4, then you can just complete the task, **Adding a content area to the Product page type and template** in Exercise C4 on page 122, to define two content areas on a product page, and then you can complete exercise D1.

Creating an editorial block type

1. In **AlloyTraining**, expand **Models**, right-click **Blocks**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Type**, enter **EditorialBlock.cs** for the Name, and click **Add**.
3. Change the **DisplayName** to **Editorial**.
4. Add a **Description** of "Use this for a rich editorial text that will be reused in multiple places."
5. Apply the **[SiteBlockIcon]** attribute to the class.
6. Add an **XhtmlString** property named **MainBody**, and allow it to be localized into multiple languages:

Your code should look something like the following:

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

namespace AlloyTraining.Models.Blocks
{
    [ContentType(DisplayName = "Editorial",
        GroupName = SiteGroupNames.Common,
        Description = "Use this for a rich editorial text that will be reused in
multiple places.")]
    [SiteBlockIcon]
    public class EditorialBlock : BlockData
    {
        [CultureSpecific]
        [Display(
```

```
            Name = "Main body",
            Description = "The main body will be shown in the main content area of the
page, using the XHTML-editor you can insert for example text, images and tables.",
            GroupName = SystemTabNames.Content,
            Order = 10)]
        public virtual XhtmlString MainBody { get; set; }
    }
}
```

Creating a controller-less block template

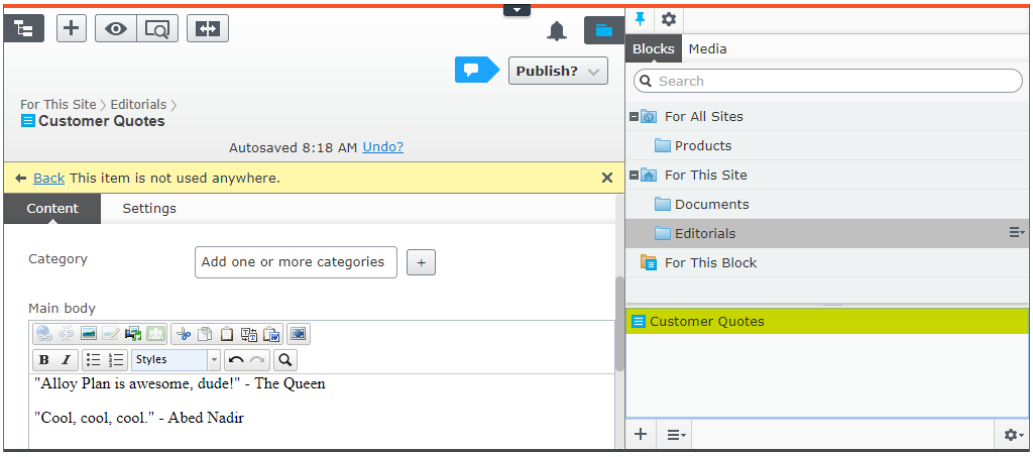
1. In **AlloyTraining**, right-click ~**Views****Shared**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **EditorialBlock.cshtml** for the name, and click **Add**.
3. Modify the view to use `EditorialBlock` as its model and to output the `MainBody` property to support on-page editing, as shown in the following markup:

```
@model AlloyTraining.Models.Blocks.EditorialBlock
<div class="block span">
    @Html.PropertyFor(m => m.MainBody)
</div>
```

Creating an instance of the block and using it in multiple places

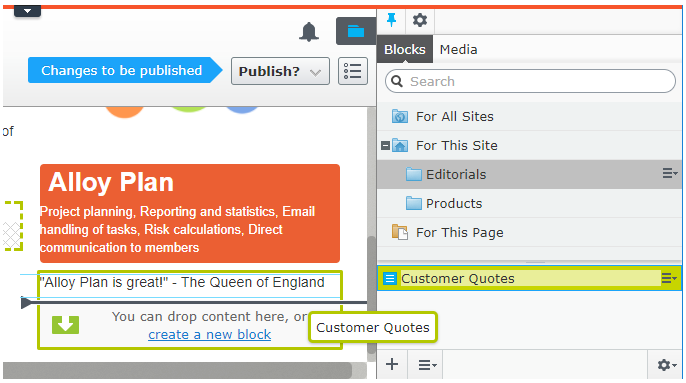
1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. In **Assets** pane, click **Blocks**.
3. Add a new folder named **Editorials** to the **For This Site** folder.
4. In the **Editorials** folder, add a new **Editorial** block named **Customer Quotes**, as shown in the following screenshot:

5. In the **Main body** write some quotes from Alloy's customers, as shown in the following screenshot:

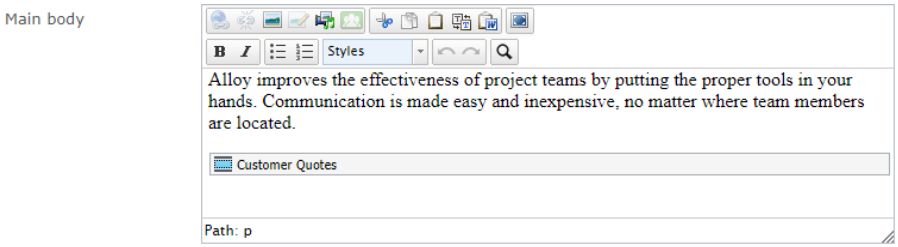


6. Publish the block and note that it is not current used anywhere.

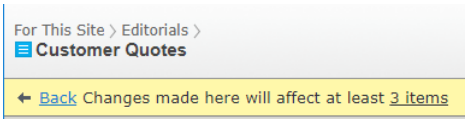
7. Edit the **Alloy Plan** page, and drag and drop the **Customer Quotes** block into its **Related content area** property, as shown in the following screenshot:



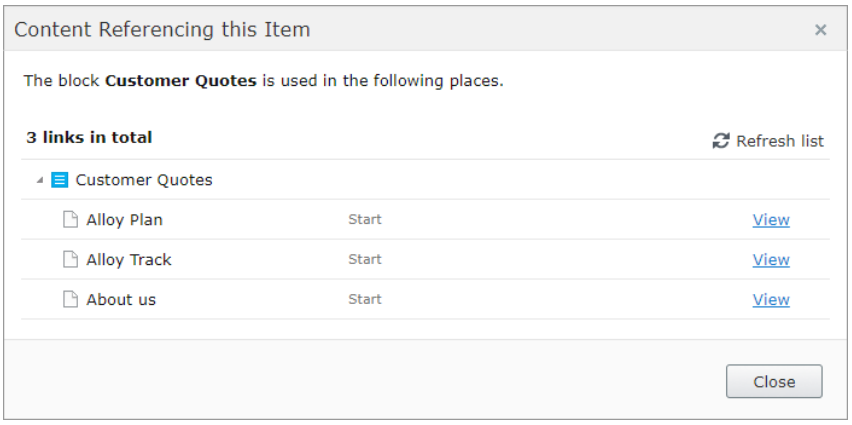
8. Publish the page.
9. Edit the **Alloy Track** page and drag and drop the **Customer Quotes** block into its **Related content area** property.
10. **Publish** the page.
11. Edit the **About us** page, and drag and drop the **Customer Quotes** block into its **Main body** property, as shown in the following screenshot:



12. **Publish** the page.
13. In the **Assets** pane, double-click **Customer Quotes**, and note the block now shows that it is referenced on three items, as shown in the following screenshot:



14. Click **3 items**, and note it shows the pages that reference the shared block, as shown in the following screenshot:



15. Close the browser.

Exercise D2 – Creating a block with a controller for teaser content

In this exercise, you will create a block type with a block template that is a view with a controller.

Blocks with controllers are more resource hungry, so you should only use them when necessary. For example, when content stored in the CMS must be combined with data from a web service, or a calculated value. In this example, we will generate a random number for a visitor count that will be shown in the block.

Prerequisites: complete Exercises B1 to B4, C1 to C4.

If you have not completed Exercises C1 to C4, then you can just complete the task, **Adding a content area to the Start page type and template** in Exercise C1 on page 108, to define two content areas on a product page, and then you can complete exercise D2.

Creating a teaser block type

1. In **AlloyTraining**, expand **Models**, right-click **Blocks**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Type**, enter **TeaserBlock.cs** for the Name, and click **Add**.
3. Change the **DisplayName** to **Teaser**.
4. Add a **Description** of "Use this for rich text with heading, image and page link that will be reused in multiple places."
5. Apply the **[SiteBlockIcon]** attribute to the class.
6. Add the following four properties with appropriate attributes:
    - **TeaserHeading**: string (with support for multiple languages)
    - **TeaserText**: XhtmlString (with support for multiple languages)
    - **TeaserImage**: ContentReference (images only)
    - **TeaserLink**: PageReference

Your code should look something like the following:

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

namespace AlloyTraining.Models.Blocks
{
    [ContentType(DisplayName = "Teaser",
        GroupName = SiteGroupNames.Common,
        Description = " Use this for rich text with heading, image and page link
that will be reused in multiple places.")]
    [SiteBlockIcon]
    public class TeaserBlock : BlockData
    {
        [CultureSpecific]
        [Display(Name = "Heading", Order = 10)]
        public virtual string TeaserHeading { get; set; }

        [CultureSpecific]
        [Display(Name = "Rich text", Order = 20)]
        public virtual XhtmlString TeaserText { get; set; }

        [Display(Name = "Image", Order = 30)]
        [UIHint(UIHint.Image)]
```

```
        public virtual ContentReference TeaserImage { get; set; }

        [Display(Name = "Link", Order = 40)]
        public virtual PageReference TeaserLink { get; set; }
    }
}
```

Creating a teaser view model, controller, and view

For complex content types, its content template is often a combination of: a controller, a view model, and a view.

1. In **AlloyTraining**, right-click **ViewModels**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **TeaserBlockViewModel.cs** for the Name, and click **Add**.
3. Modify the view model to have two properties: **CurrentBlock** and **VisitorCount**, as shown in the following code:

```
using AlloyTraining.Models.Blocks;

namespace AlloyTraining.Models.ViewModels
{
    public class TeaserBlockViewModel
    {
        public TeaserBlock CurrentBlock { get; set; }
        public int TodaysVisitorCount { get; set; }
    }
}
```

4. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Controller (MVC)**, enter **TeaserBlockController.cs** for the Name, and click **Add**.
6. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Blocks** namespace.
7. Modify the class to create an instance of the teaser view model and set its properties, before passing it to a partial view, as shown in the following code:

```
using AlloyTraining.Models.Blocks;
using AlloyTraining.Models.ViewModels;
using EPiServer.Web.Mvc;
using System;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class TeaserBlockController : BlockController<TeaserBlock>
    {
        public override ActionResult Index(TeaserBlock currentBlock)
        {
            var viewmodel = new TeaserBlockViewModel
            {
                CurrentBlock = currentBlock,
                TodaysVisitorCount = (new Random()).Next(300, 900)
            };
            return PartialView(viewmodel);
        }
    }
}
```

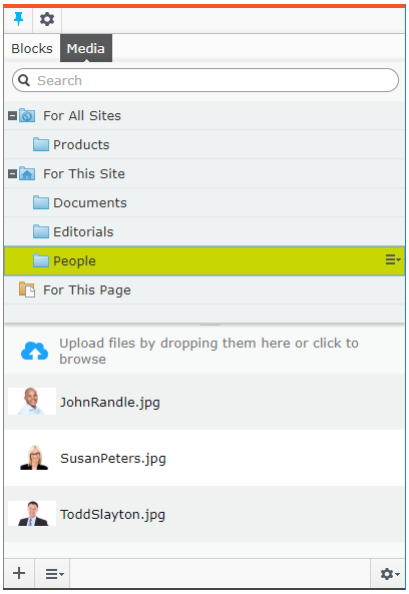
We have simulated some data for the visitor count using the `Random` class.

8. In **AlloyTraining**, right-click **Views**, and add a new folder named **TeaserBlock**.
9. Right-click **TeaserBlock**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
10. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
11. Modify the view, as shown in the following markup:

```
@model AlloyTraining.Models.ViewModels.TeaserBlockViewModel
<div class="media">
    <div style="clear:both;">
        <div class="mediaImg">
            @Html.PropertyFor(m => m.CurrentBlock.TeaserImage)
        </div>
        <div class="mediaText">
            <h2 @Html.EditAttributes(m => m.CurrentBlock.TeaserHeading)>
                @Model.CurrentBlock.TeaserHeading
            </h2>
            <p>@Html.PropertyFor(m => m.CurrentBlock.TeaserText)</p>
            <small>There have been @Model.TodaysVisitorCount visitors today.</small>
        </div>
    </div>
</div>
```

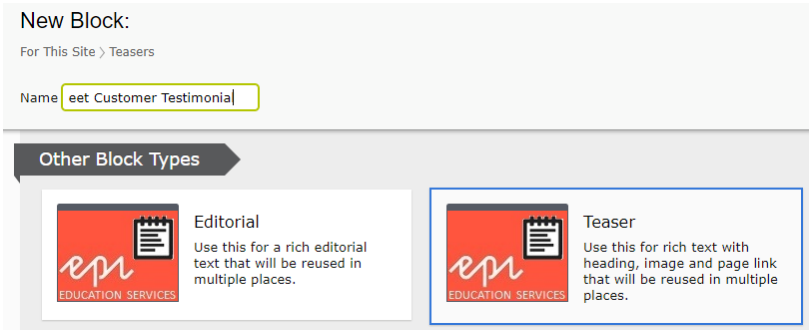
Creating an instance of the teaser block

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. In **Assets** pane, click **Media**.
3. Add a new folder named **People** to the **For This Site** folder.
4. Upload the three images of people in the **~\Assets\People** folder, as shown in the following screenshot:

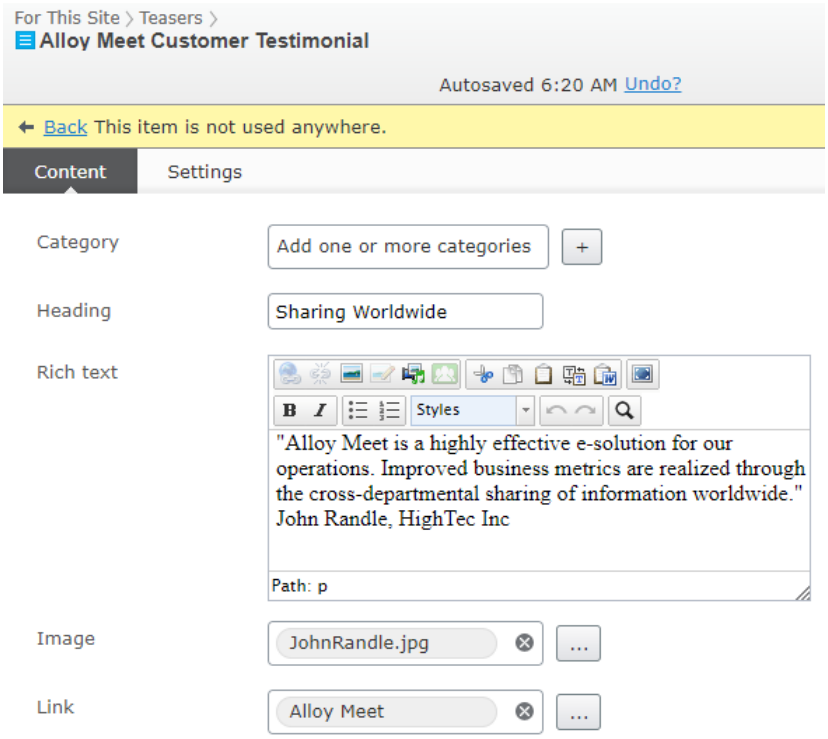


5. In **Assets** pane, click **Blocks**.
6. Add a new folder named **Teasers** to the **For This Site** folder.

7. Add a new **Teaser** block to the **Teasers** folder named **Alloy Meet Customer Testimonial**, as shown in the following screenshot:



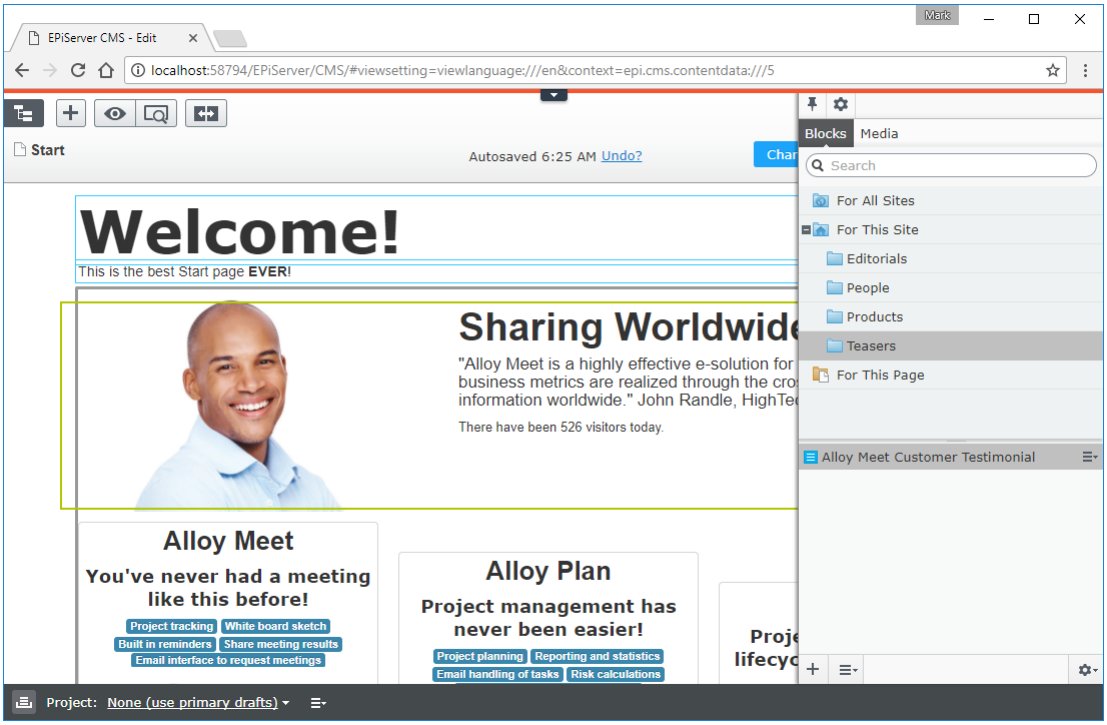
8. Add values for the properties, as shown in the following screenshot:
    a. **Heading**: Sharing Worldwide
    b. **Rich text**: "Alloy Meet is a highly effective e-solution for our operations. Improved business metrics are realized through the cross-departmental sharing of information worldwide." John Randle, HighTec Inc
    c. **Image**: JohnRandle.jpg (use the picker or drag and drop from Assets pane)
    d. **Link**: Alloy Meet (use the picker or drag and drop from Navigation pane)



By default, blocks do not support on-page editing, so content editors must use **All Properties** view. In the next exercise, you will create a preview template for blocks to enable an on-page editing experience.

9. **Publish** the block.

10. Edit the **Start** page, drag and drop the **Alloy Meet Customer Testimonial** to the top of its main content area, and publish the page, as shown in the following screenshot:



This implementation of the teaser block is okay, but it has two limitations: (1) the context menu shows the display options that you defined for product pages, but the teaser block ignores the selection a content editor might make, and (2) editors must use **All Properties** view to change its properties. In the next exercise, you will fix both limitiations.

Exercise D3 – Creating a preview renderer for partial pages and shared blocks

In this exercise, you will create preview renderer for the teaser block.

The renderer will give previews of how the block will look when rendered with various templates.

Prerequisites: complete Exercises B1 – B4, C1 – C4, D2.

Creating three teaser block templates for full, wide, and narrow

1. In **AlloyTraining**, open **TeaserBlockController.cs**.
2. Copy and paste the statements that define the **TeaserBlockController** class so that you have three controllers, and apply **TemplateDescriptor** to make the template resolver select them when the tags: “full”, “wide”, and “narrow” have been applied, as shown in the following code:

```
using AlloyTraining.Models.Blocks;
using AlloyTraining.Models.ViewModels;
using EPiServer.Framework.DataAnnotations;
using EPiServer.Web.Mvc;
using System;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    [TemplateDescriptor(Tags = new[] { SiteTags.Full },
        AvailableWithoutTag = true)]
    public class TeaserBlockController : BlockController<TeaserBlock>
    {
        public override ActionResult Index(TeaserBlock currentBlock)
        {
            var viewmodel = new TeaserBlockViewModel
            {
                CurrentBlock = currentBlock,
                TodaysVisitorCount = (new Random()).Next(300, 900)
            };
            return PartialView(viewmodel);
        }
    }

    [TemplateDescriptor(Tags = new[] { SiteTags.Wide })]
    public class TeaserBlockWideController : BlockController<TeaserBlock>
    {
        public override ActionResult Index(TeaserBlock currentBlock)
        {
            var viewmodel = new TeaserBlockViewModel
            {
                CurrentBlock = currentBlock,
                TodaysVisitorCount = (new Random()).Next(300, 900)
            };
            return PartialView(viewmodel);
        }
    }

    [TemplateDescriptor(Tags = new[] { SiteTags.Narrow })]
    public class TeaserBlockNarrowController : BlockController<TeaserBlock>
    {
        public override ActionResult Index(TeaserBlock currentBlock)
        {
            var viewmodel = new TeaserBlockViewModel
            {
```

```
                CurrentBlock = currentBlock,
                TodaysVisitorCount = (new Random()).Next(300, 900)
            };
            return PartialView(viewmodel);
        }
    }
}
```

3. In **Views** folder, copy and paste the **TeaserBlock** folder twice to create two copies named: **TeaserBlockWide** and **TeaserBlockNarrow**.
4. Open **~\Views\TeaserBlockNarrow\Index.cshtml**.
5. Add class **span4** to the outer , and delete the that outputs the image.
6. Open **~\Views\TeaserBlockWide\Index.cshtml**.
7. Add class **span8** to the outer .

Creating a preview view model, controller, and view

When a content editor edits the teaser block, we would like them to see a preview of what the block would look like if any of the three display options are selected. To do this, we need to simulate a page that has a property that contains an instance of a block. We can do this by defining a view model with a `ContentArea` and add the instance to the area. Then on the simulated page, we can easily output the block by calling `PropertyFor` three times, once for each display option.

1. In **AlloyTraining**, expand **Models**, right-click **ViewModels**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **PreviewPageViewModel.cs** for the name, and click **Add**.
3. Modify the contents, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using EPiServer.Core;

namespace AlloyTraining.Models.ViewModels
{
    public class PreviewPageViewModel : DefaultPageViewModel<SitePageData>
    {
        public PreviewPageViewModel(SitePageData currentPage,
            IContent contentToPreview) : base(currentPage)
        {
            this.PreviewArea = new ContentArea();
            this.PreviewArea.Items.Add(new ContentAreaItem
                { ContentLink = contentToPreview.ContentLink });
        }

        public ContentArea PreviewArea { get; set; }
    }
}
```

4. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **PreviewPageController.cs** for the Name, and click **Add**.
6. Modify its contents, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using EPiServer.Core;
using EPiServer.Framework.DataAnnotations;
using EPiServer.Framework.Web;
```

```
using EPiServer.ServiceLocation;
using EPiServer.Web;
using EPiServer.Web.Mvc;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    [TemplateDescriptor(Inherited = true,
        TemplateTypeCategory = TemplateTypeCategories.MvcController,
        Tags = new[] { RenderingTags.Preview },
        AvailableWithoutTag = false)]
    public class PreviewPageController :
        ActionControllerBase, IRenderTemplate<BlockData>
    {
        public ActionResult Index(IContent currentContent)
        {
            var loader = ServiceLocator.Current
                .GetInstance<EPiServer.IContentLoader>();
            var startPage = loader.Get<SitePageData>
                (ContentReference.StartPage);
            var viewmodel = new PreviewPageViewModel(
                startPage, currentContent);
            return View(viewmodel);
        }
    }
}
```

Our site enforces a rule for view models passed to layouts: they must have a `CurrentPage` property with a page instance in it. For our preview, we can get the Start page and pass that in. The preview won't use it, but some page must be passed in.

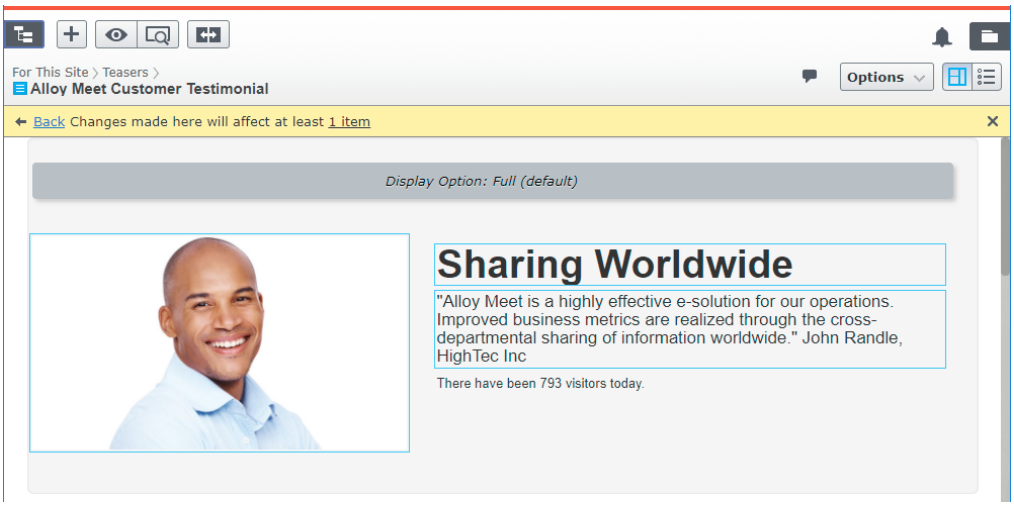
7. In **AlloyTraining**, right-click **Views**, and add a new folder named **PreviewPage**.
8. Right-click **PreviewPage**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
9. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
10. Modify the view, as shown in the following markup:

```
@using AlloyTraining
@using EPiServer.Web.Mvc.Html
@model AlloyTraining.Models.ViewModels.PreviewPageViewModel
@{
    Layout = null;
}
<head>
    <title>@(Model.CurrentPage.MetaTitle ?? Model.CurrentPage.Name)</title>
    <link rel="stylesheet" href="@Url.Content("~/Static/css/bootstrap.css")" />
    <link rel="stylesheet"
          href="@Url.Content("~/Static/css/bootstrap-responsive.css")" />
    <link rel="stylesheet" href="@Url.Content("~/Static/css/media.css")" />
    <link rel="stylesheet" href="@Url.Content("~/Static/css/style.css")" />
    <link rel="stylesheet" href="@Url.Content("~/Static/css/editmode.css")" />
    <script type="text/javascript"
            src="@Url.Content("~/Static/js/jquery.js")"></script>
    <script type="text/javascript"
            src="@Url.Content("~/Static/js/bootstrap.js")"></script>
</head>
<body>
    <div class="container">
        <div class="well well-large">
            <div class="row">
```

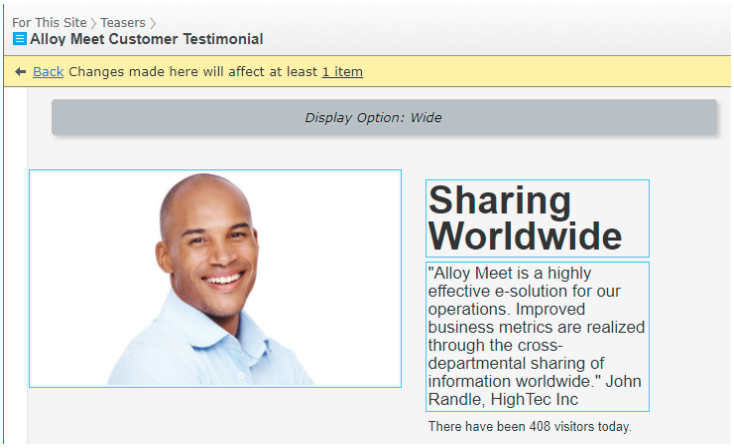
```
                <div class="alert alert-info">Display Option: Full (default)</div>
            </div>
            <div class="row">
                @Html.PropertyFor(m => m.PreviewArea, new { Tag = SiteTags.Full })
            </div>
        </div>
        <div class="well well-large">
            <div class="row">
                <div class="alert alert-info span8">Display Option: Wide</div>
            </div>
            <div class="row">
                @Html.PropertyFor(m => m.PreviewArea, new { Tag = SiteTags.Wide })
            </div>
        </div>
        <div class="well well-large">
            <div class="row">
                <div class="alert alert-info span4">Display Option: Narrow</div>
            </div>
            <div class="row">
                @Html.PropertyFor(m => m.PreviewArea, new { Tag = SiteTags.Narrow })
            </div>
        </div>
    </div>
</body>
```

Testing the preview

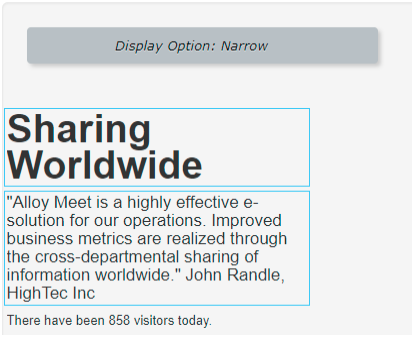
1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **Alloy Meet Customer Testimonial** teaser block, and note that instead of only allowing **All Properties** view, it defaults to using an **On-Page Editing** view, and shows a preview of the **Full** (default) display option, as shown in the following screenshot:



3. Scroll down the preview page to see the **Wide** display option, as shown in the following screenshot:



4. Scroll down the preview page to see the **Narrow** display option, as shown in the following screenshot:



5. Close the browser.

Exercise D4 – Optional: Moving properties to the basic info area

In this exercise, you will move a property from the tabbed area up to the basic info properties area.

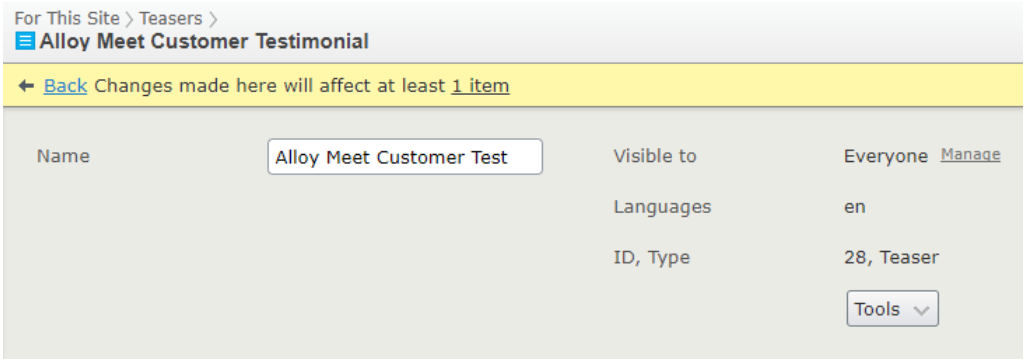
Prerequisites: complete Exercises B1 – B4, C1 – C4, D2.

Understanding the existing basic info area

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **Start** page. Note the on-page editing view contains several properties which are reached by scrolling beyond the top of the page to reveal the top gray area. These are called basic info properties and can be used to:
   - Give the page a simple address.
   - Set access rights for a page.
   - Change the name in the URL, and so on.

   This basic info area is always displayed in the **All Properties** editing view, as shown in the following screenshot:

3. Edit the **Alloy Meet Customer Testimonial** teaser and view the basic information area. Note for a block, like **TeaserBlock**, it cannot have a URL, a simple address, or display in navigation, so there is empty space available that we can make better use of, as shown in the following screenshot:



Making use of space in the basic information area for blocks

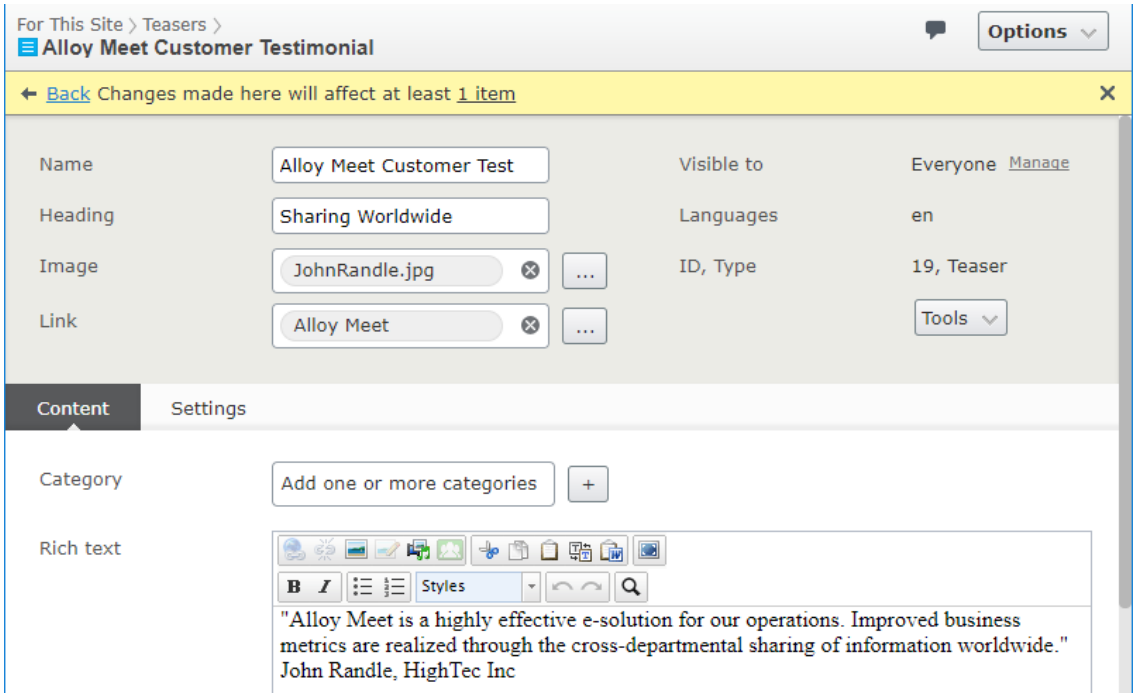
1. In **AlloyTraining**, open **~/Models/Blocks/TeaserBlock.cs**.
2. Modify the **[Display]** attribute for the **TeaserHeading**, **TeaserImage**, and **TeaserLink** properties to set the **GroupName** to **PageHeader**, as shown in the following code:

```
[Display(Name = "Link", Order = 40,
    GroupName = SystemTabNames.PageHeader)]
public virtual PageReference TeaserLink { get; set; }
```

The "magic string" for `SystemTabNames.PageHeader` is `"EPiServerCMS_SettingsPanel"`.

3. Start the **AlloyTraining** website, and log in as a CMS admin.
4. Edit the **Start** page.

5. In the main content area, edit the **Alloy Meet Customer Testimonial** teaser, switch to **All Properties** view, and note the the three properties are now in the basic information area, as shown in the following screenshot:



For your own block types, consider moving the two or three most commonly set properties to the basic information area.

Exercise D5 – Optional: Using a block as a content property type

In this exercise, you will define and use an employee block type as a property type.

Prerequisites: complete Exercises B1 to B4.

Using a block as a property type

1. In **AlloyTraining**, add an **EmployeeBlock** block type for storing information about employees: **FirstName**, **LastName**, and **HireDate**, as shown in the following code:

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

namespace AlloyTraining.Models.Blocks
{
    [ContentType(DisplayName = "Employee",
        GroupName = SiteGroupNames.Specialized,
        Order = 10,
        Description = "Use this to store information about an employee.")]
    [SiteBlockIcon]
    public class EmployeeBlock : BlockData
    {
        [Display(Name = "First name",
            GroupName = SystemTabNames.Content,
            Order = 10)]
        public virtual string FirstName { get; set; }

        [Display(Name = "Last name",
            GroupName = SystemTabNames.Content,
            Order = 20)]
        public virtual string LastName { get; set; }

        [Display(Name = "Hire date",
            GroupName = SystemTabNames.Content,
            Order = 30)]
        public virtual DateTime? HireDate { get; set; }
    }
}
```

4. In **AlloyTraining**, right-click **~\Views\Shared**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **EmployeeBlock.cshtml** for the Name, and click **Add**.
6. Modify the view to use **EmployeeBlock** as its model and to output the **FirstName**, **LastName**, and **HireDate** properties, as shown in the following markup:

```
@model AlloyTraining.Models.Blocks.EmployeeBlock
<div class="block span">
    @Model.FirstName @Model.LastName
    @(Model.HireDate.HasValue ? "was hired on " +
      Model.HireDate.Value.ToString("dddd, d MMMM yyyy") : "")
</div>
```

7. In **AlloyTraining**, open **~\Models\Pages\StandardPage.cs**.
8. Import the blocks namespace, as shown in the following code:

```
using AlloyTraining.Models.Blocks;
```

9. Add a property, as shown in the following code:

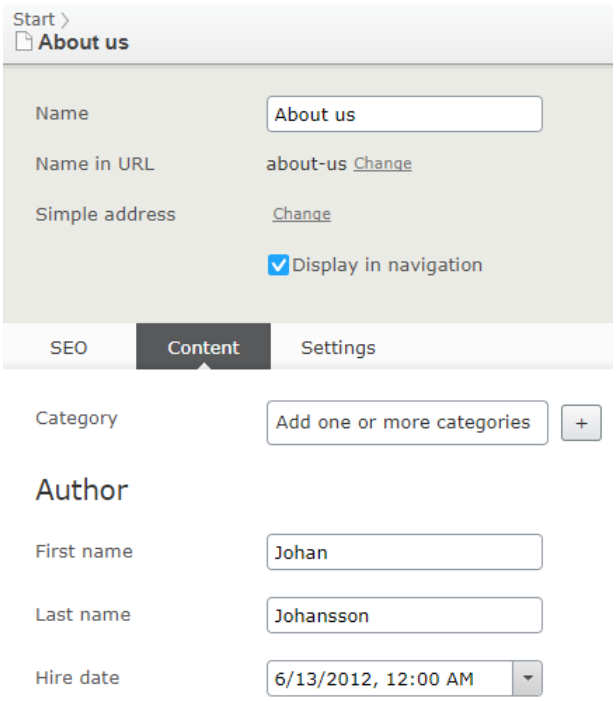
```
public virtual EmployeeBlock Author { get; set; }
```

10. In **AlloyTraining**, open **~\Views\StandardPage\Index.cshtml**.
11. At the bottom of the view, render the **Author** property with support for on-page editing:

```
@Html.PropertyFor(m => m.CurrentPage.Author)
```

Testing the block type property

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **About us** page, switch to **All Properties** view, and click the **Content** tab.
3. Modify the **Author** property's three properties, as shown in the following screenshot:



4. **Publish** the page.
5. Switch to **Live** view and note the author's details are rendered onto the page.
6. Close the browser.

Module E – Navigating Content

Goal

The overall goal of the exercises in this module is to implement various ways for a visitor to navigate a website. You will:

1. Create a child page listing block.
2. Use the child page listing block in a page for navigating news articles.
3. Create a submenu for navigation.
4. Create a search page for visitors and implement it using Episerver Search or Episerver Find.
5. Add a search box to the top navigation menu.

Exercise E1 – Creating a page listing block

In this exercise, you will create a new block type containing a listing of child pages.

The block will have two properties; a heading and a page picker to select a parent page to show a list of its children.

Prerequisites: complete Exercises B1 to B4.

Creating a page listing block type

1. In **AlloyTraining**, expand **Models**, right-click **Blocks**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Type**, enter **ListingBlock.cs** for the name, and click **Add**.
3. Change the **DisplayName** to **Listing**.
4. Add a **Description** of "Choose a page in the tree, and its children will be listed, with a heading."
5. Apply the **[SiteBlockIcon]** attribute to the class.
6. Add the following properties with appropriate attributes:
    a. **Heading**: `string`
    b. **ShowChildrenOfThisPage**: `PageReference`

Your code should look something like the following:

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

namespace AlloyTraining.Models.Blocks
{
    [ContentType(DisplayName = "Listing",
        GroupName = SiteGroupNames.Common,
        Description = "Choose a page in the tree, and its children will be
listed, with a heading.")]
    [SiteBlockIcon]
    public class ListingBlock : BlockData
    {
        [Display(Name = "Heading", Order = 10)]
        public virtual string Heading { get; set; }

        [Display(Name = "Show children of this page", Order = 20)]
```

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

Creating a page listing view model, controller, and view

1. In **AlloyTraining**, right-click **ViewModels**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **ListingBlockViewModel.cs** for the name, and click **Add**.
3. Add two properties: **Heading** and **Pages**, as shown in the following code:

```
using EPiServer.Core;
using System.Collections.Generic;

namespace AlloyTraining.Models.ViewModels
{
    public class ListingBlockViewModel
    {
        public string Heading { get; set; }
        public IEnumerable<PageData> Pages { get; set; }
    }
}
```

4. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
5. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Controller (MVC)**, enter **ListingBlockController.cs** for the Name, and click **Add**.
6. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Blocks** namespace.
7. Modify the class to create an instance of the teaser view model and set its properties, before passing it to a partial view, as shown in the following code:

```
using AlloyTraining.Models.Blocks;
using AlloyTraining.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using EPiServer.Filters;
using EPiServer.Web.Mvc;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class ListingBlockController : BlockController<ListingBlock>
    {
        private readonly IContentLoader loader;

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

        public override ActionResult Index(ListingBlock currentBlock)
        {
            var viewmodel = new ListingBlockViewModel
            {
                Heading = currentBlock.Heading
            };
```

```
            if (currentBlock.ShowChildrenOfThisPage != null)
            {
                IEnumerable<PageData> children = loader.GetChildren<PageData>(
                    currentBlock.ShowChildrenOfThisPage);

                // Remove pages:
                // 1. that are not published
                // 2. that the visitor does not have Read access to
                // 3. that do not have a page template
                IEnumerable<IContent> filteredChildren =
                    FilterForVisitor.Filter(children);

                // 4. that do not have "Display in navigation" selected
                viewmodel.Pages = filteredChildren.Cast<PageData>()
                    .Where(page => page.VisibleInMenu);
            }

            return PartialView(viewmodel);
        }
    }
}
```

8. In **AlloyTraining**, right-click **Views**, and add a new folder named **ListingBlock**.
9. Right-click **ListingBlock**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
10. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
11. Modify the view, as shown in the following markup, and note the following:
    - When there are no pages to show, we render alert messages visible to content editors.
    - `PageData` does not have a strongly-typed `MainBody` property, but the page it references might, so we can use the `Property` property to check if the current page in the listing has a `MainBody` and render it if it does.

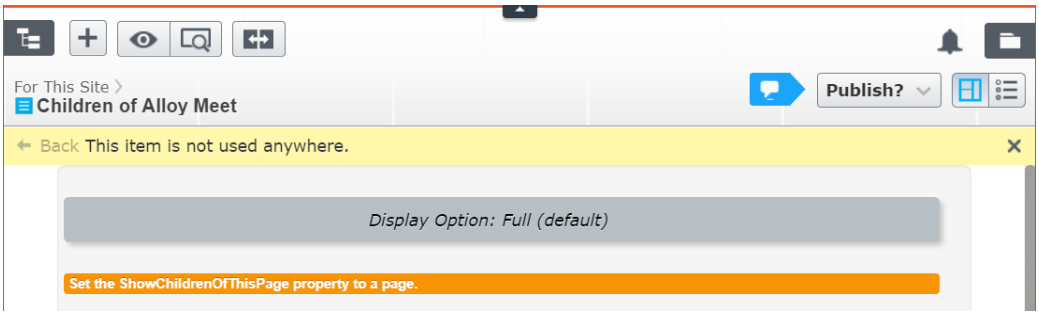
```
@using EPiServer.Core
@model AlloyTraining.Models.ViewModels.ListingBlockViewModel
@if (Model.Pages == null)
{
    if (EPiServer.Editor.PageEditing.PageIsInEditMode)
    {
        <div class="label label-warning">Set the ShowChildrenOfThisPage property to a
page.</div>
    }
}
else
{
    <h2 @Html.EditAttributes(x => x.Heading)>@Model.Heading</h2>
    if (Model.Pages.Count() == 0)
    {
        <div class="label label-warning">The page selected has no children.</div>
    }
    foreach (PageData page in Model.Pages)
    {
        <div class="listresult theme1">
            <h3>@Html.ContentLink(page.ContentLink)</h3>
            @if (page.StartPublish.HasValue)
            {
                <p class="date">Published on
                    @page.StartPublish.Value.ToString("dddd, d MMMM yyyy")
                </p>
            }
            @if (page.Property["MainBody"] != null)
```

```
            {
                @Html.Raw(page.Property["MainBody"].Value)
            }
            <hr />
        </div>
    }
}
```

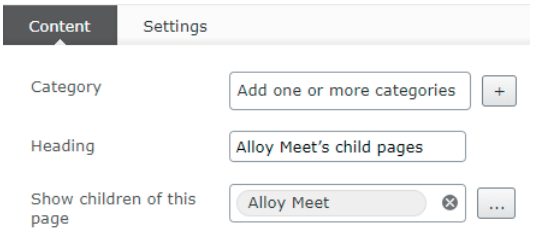
Testing the page listing block

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. In **Assets** pane, click **Blocks**.
3. Add a new **Listing** block to the **For This Site** folder named **Children of Alloy Meet**, as shown in the following screenshot:

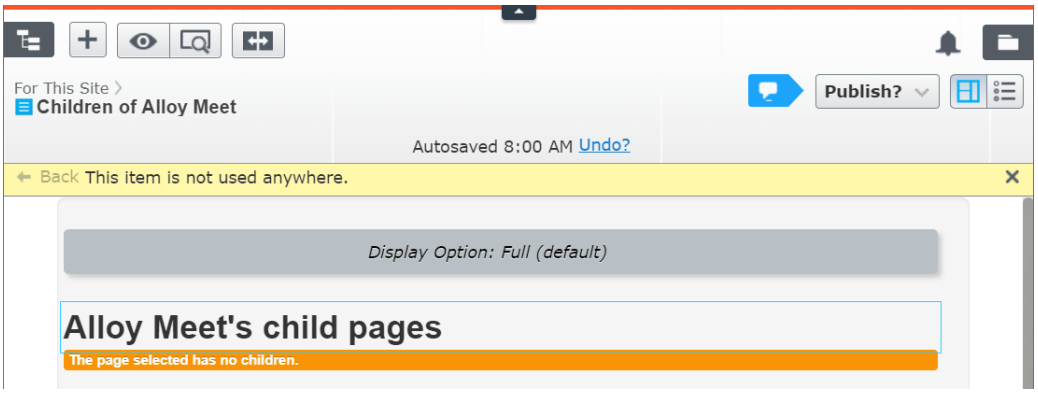
4. Switch to **On-Page Editing** view, and note the warning to editors, as shown in the following screenshot:



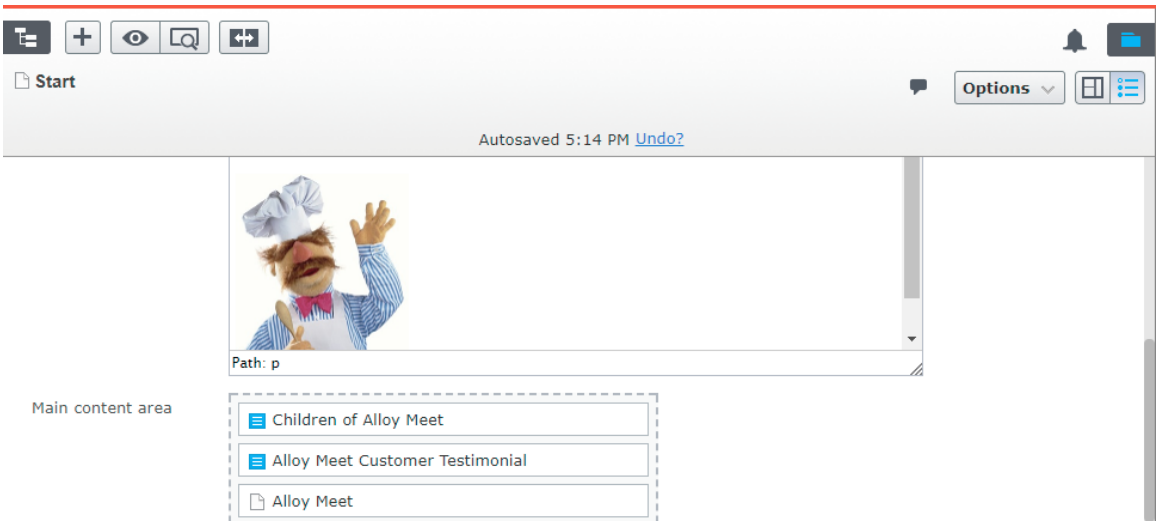
5. Switch to **All Properties** view, and set the properties, as shown in the following screenshot:
    a. **Heading**: Alloy Meet's child pages
    b. **ShowChildrenOfThisPage**: Alloy Meet



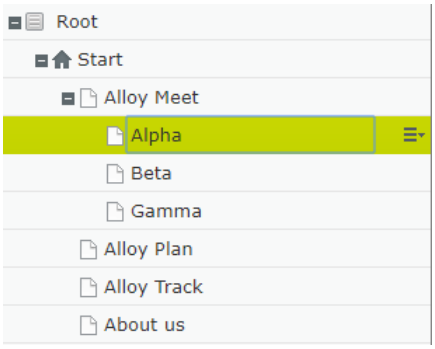
6. Switch to **On-Page Editing** view, and note the warning to editors, as shown in the following screenshot:



7. **Publish** the block.
8. Edit the **Start** page, and drag and drop **Children of Alloy Meet** into the top of its main content area, as shown in the following screenshot:



9. **Publish** the page.
10. Add some standard pages underneath **Alloy Meet** in the page tree, named **Alpha**, **Beta**, and **Gamma**, as shown in the following screenshot:

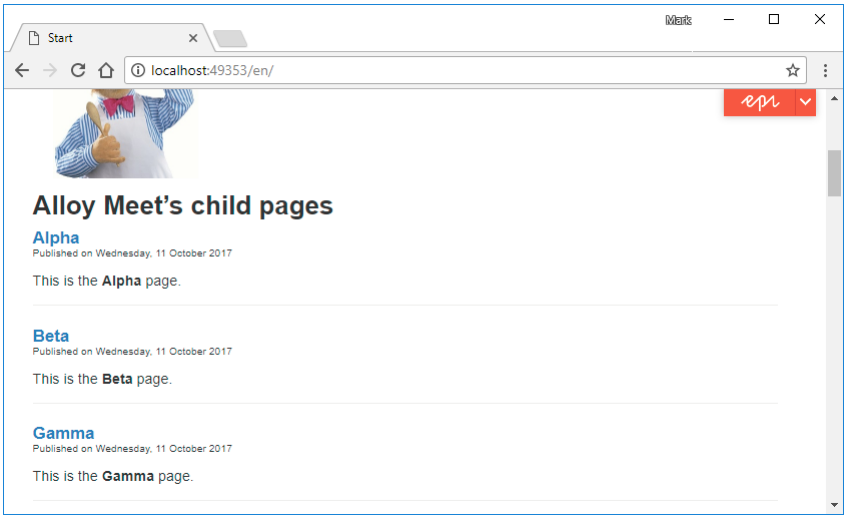


11. For each page, set their **Main body** properties using a Lorem Ipsum generator.

> Lorem Ipsum is dummy text of the printing and typesetting industry. Learn about and generate some at the following link: https://www.lipsum.com/

12. Switch to **Live** view.

13. View the **Start** page, and note the output, as shown in the following screenshot:



14. Close the browser.

You have now implemented a simple listing block. In the next exercise, you will use it to implement a news page that lists articles.

Exercise E2 – Creating a news landing page

In this exercise, you will create a news landing page type that lists news articles beneath it in the page tree.

You will use the standard page type as a base for the news landing page, you will use the page listing block from the previous exercise to add a news listing function to the news landing page, and you will create instances of the standard page under the news landing page to be its child news articles.

Prerequisites: complete Exercises B1 – B4, E1.

Creating a news landing page type

1. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **NewsLandingPage.cs** for the Name, and click **Add**.
3. Modify the class to inherit from **StandardPage**.
4. Change the **DisplayName** to **News Landing**.
5. Add a **Description** of "Use this as a landing page for a list of news articles."
6. Put the page in the **News** group.
7. Apply the **[SitePageIcon]** attribute to the class.
8. Delete the **MainBody** property.
9. Add a **ListingBlock** property named **NewsListing** with appropriate attributes.

It would be nice if we could override the **SetDefaultValues** property and set the **NewsListing** property's **Heading** to the **Name** of the page and the **Parent** to reference this page, i.e. the news landing page will show a listing of its child pages, but unfortunately, the Name and PageLink properties are empty during SetDefaultValues. You could use an initialization module and listen for the **CreatedContent** event to set these properties automatically instead.

Your code should look something like the following:

```
using AlloyTraining.Models.Blocks;
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;
using System.ComponentModel.DataAnnotations;

namespace AlloyTraining.Models.Pages
{
    [ContentType(DisplayName = "News Landing",
        GroupName = SiteGroupNames.News,
        Description = "Use this as a landing page for a list of news
articles.")]
    [SitePageIcon]
    public class NewsLandingPage : StandardPage
    {
        [Display(Name = "News listing", Order = 315)]
        public virtual ListingBlock NewsListing { get; set; }
    }
}
```

Creating a news landing page template

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.

2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **NewsLandingPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace.
4. Modify the class to derive from **PageControllerBase<T>**, and the **Index** action method to use a view model, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class NewsLandingPageController
        : PageControllerBase<NewsLandingPage>
    {
        public NewsLandingPageController(IContentLoader loader) : base(loader)
        {
        }

        public ActionResult Index(NewsLandingPage currentPage)
        {
            var viewmodel =
                new DefaultPageViewModel<NewsLandingPage>(currentPage);
            return View(viewmodel);
        }
    }
}
```

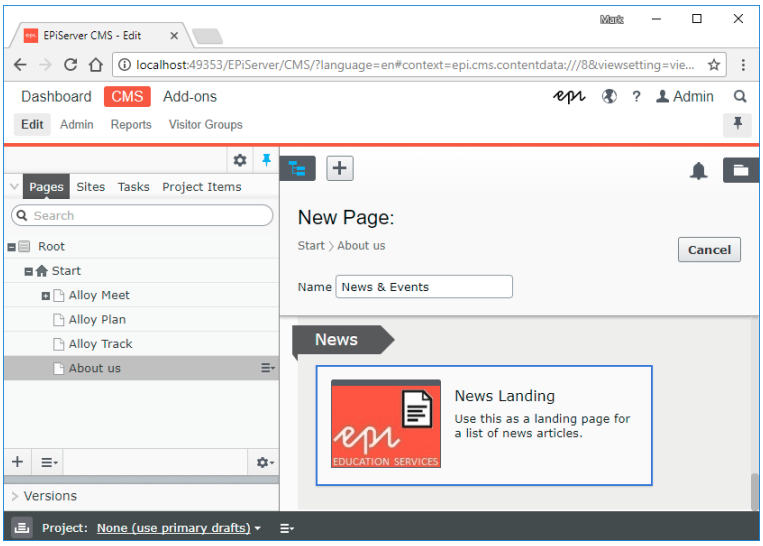
5. In **AlloyTraining**, right-click **Views**, and add a new folder named **NewsLandingPage**.
6. Right-click **NewsLandingPage**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
7. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
8. Modify the view, as shown in the following markup:

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model DefaultPageViewModel<NewsLandingPage>
@{
    Layout = "~/Views/Shared/Layouts/_LeftNavigation.cshtml";
}
<h1 @Html.EditAttributes(m => m.CurrentPage.MetaTitle)>
    @(Model.CurrentPage.MetaTitle ?? Model.CurrentPage.Name)
</h1>
<p class="introduction">
    @Html.PropertyFor(m => m.CurrentPage.MetaDescription)
</p>
<div class="row">
    <div class="span8">
        @Html.PropertyFor(m => m.CurrentPage.MainBody)
        @Html.PropertyFor(m => m.CurrentPage.NewsListing)
    </div>
</div>
```

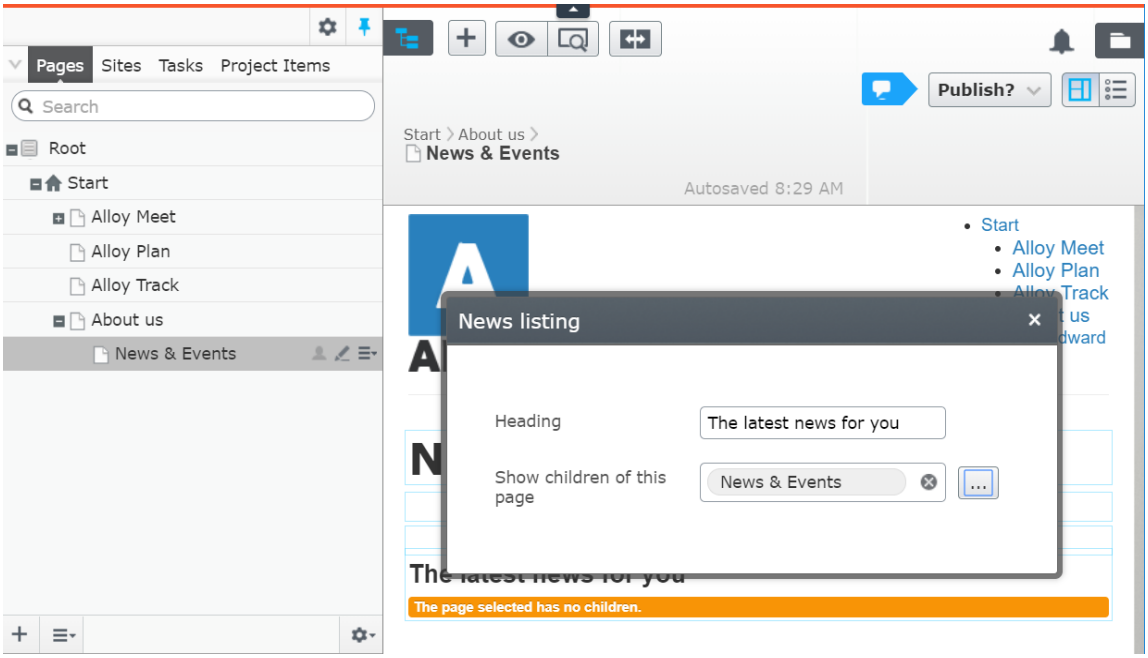
Testing the news landing page

1. Start the **AlloyTraining** website, and log in as a CMS admin.

2. Under **About us**, add a new **News Landing** page named **News & Events**, as shown in the following screenshot:



3. In **On-Page Editing** view, click the **News listing** property.
4. Set **Heading** to **The latest news for you**, and **Show children of this page** to reference the **News & Events** page, as shown in the following screenshot:



5. **Publish** the page.
6. Under **News & Events**, add two **Standard** pages named **Alloy Saves Bears** and **Join Us at Our Customer Event**, and publish them.
7. View the **News & Events** page as a visitor and note the two news articles listed on the page.
8. Close the browser.

Exercise E3 – Improving navigation menus

In this exercise, you will replace the simple menu from an earlier exercise with a better-looking menu using an extension method named MenuList.

The basics of what this **MenuList** helper does is that it allows you to define a markup template for each item in your menu and it does filtering to remove unwanted items (for example, pages with no renderer can be filtered out).

Prerequisites: complete Exercises B1 to B4.

Improving the top navigation menu

1. In **AlloyTraining**, open **~\Views\Shared_NavigationMenu.cshtml**.
2. Modify the view, as shown in the following markup:

To save time, and avoid line breaks, drag and drop or copy and paste the file from the exercise files ZIP.

```
@using EPiServer.Core
@using AlloyTraining.Business.ExtensionMethods
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@model IPageViewModel<SitePageData>
<div class="alloyMenu">
    <div class="navbar">
        <div class="navbar-inner">
            <div class="container">
                <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </a>
                <div class="nav-collapse">
                    <ul class="nav">
                        <li
class="@(Model.CurrentPage.ContentLink.CompareToIgnoreWorkID(ContentReference.StartPag
e) ? "active" : null)">
                            @Html.ContentLink(ContentReference.StartPage)
                        </li>
                        @Html.MenuList(ContentReference.StartPage,
                        @<li class="@(item.Selected ? "active" : null)">
                            @Html.PageLink(item.Page)
                        </li>)
                        <li>
                            @if (User.Identity.IsAuthenticated)
                            {
                                <a href="/en/logout">Log out @User.Identity.Name</a>
                            }
                            else
```

```
                            {
                                <a
href="@FormsAuthentication.LoginUrl?ReturnUrl=@Model.CurrentPage.PageLink.ExternalURLF
romReference()">Log in</a>
                            }
                        </li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>
```

Adding a left navigation submenu

1. In **AlloyTraining**, right-click **~\Views\Shared**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.

Make sure you add the file to ~\Views\Shared, and NOT to ~\Views\Shared\Layouts.

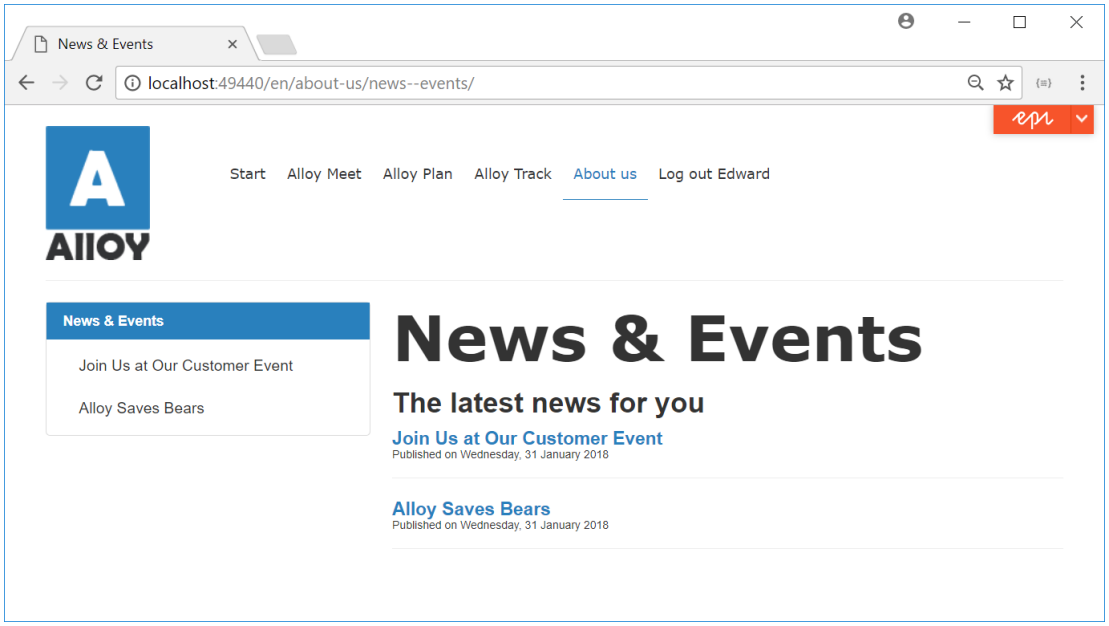
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **_LeftNavigationMenu.cshtml** for the Name, and click **Add**.
3. Modify the view, as shown in the following markup:

```
@using AlloyTraining.Models.ViewModels
@using AlloyTraining.Models.Pages
@using AlloyTraining.Business.ExtensionMethods
@model IPageViewModel<SitePageData>
@helper ItemTemplate(MenuItem firstLevelItem)
{
    <div class="accordion-heading">
        <a href="@Url.ContentUrl(firstLevelItem.Page.PageLink)"
class="@(firstLevelItem.Page.ContentLink.CompareToIgnoreWorkID(Model.CurrentPage.Conte
ntLink) ? "accordion-toggle active" : "accordion-toggle")" data-parent="#alloyDrop">
            @firstLevelItem.Page.PageName
            <i class="@(firstLevelItem.HasChildren.Value ? "icon-chevron-down right" :
"right")"></i>
        </a>
    </div>
    <div id="collapse-@firstLevelItem.Page.ContentLink.ID" class="accordion-body
collapse @(firstLevelItem.Selected ? "in" : "")">
        <ul>
            @Html.MenuList(firstLevelItem.Page.ContentLink, SubLevelItemTemplate)
        </ul>
    </div>
}
@helper SubLevelItemTemplate(MenuItem subLevelItem)
{
    <li class="@(subLevelItem.Selected ? "active" : null)">
        @Html.PageLink(subLevelItem.Page)
        @*To show more levels call Html.MenuList recursively here if
subLevelItem.Selected == true*@
    </li>
}
<div id="alloyDrop" class="accordion">
    <div class="accordion-group">
        @if (Model.Section != null)
            {
            @Html.MenuList(Model.Section.ContentLink, ItemTemplate)
        }
    </div>
</div>
```

4. Open **~\Views\Shared\Layouts_LeftNavigation.cshtml**.
5. Add a statement to render the **_LeftNavigationMenu**, inside `<div class="span4">`, as shown in the following markup:

   ```
   @Html.Partial("_LeftNavigationMenu", Model)
   ```

6. Start **AlloyTraining** website, and navigate to **About us** | **News & Events**, and note the top and left navigation menus, as shown in the following screenshot:



7. Close the browser.

Exercise E4 – Creating a search page for visitors

In this exercise, you will add a search page to the AlloyTraining site, allowing visitors to perform free-text searches on pages using either Episerver Find or Episerver Search.

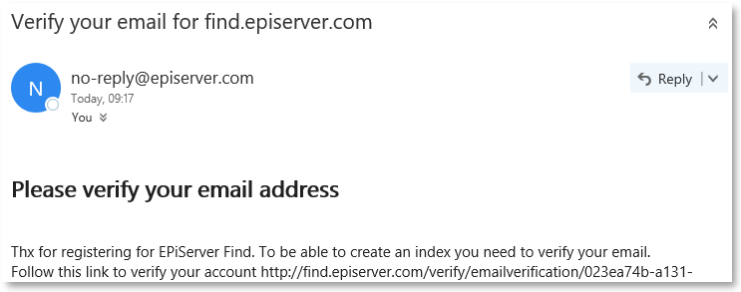
If you drag and drop from the exercise files solution, then be careful not to add *both* controllers because they will conflict! The solution files named **SearchPageControllerUsingEpiserverFind.cs** and **SearchPageControllerUsingEpiserverSearch.cs**.

Prerequisites: complete Exercises B1 to B4.

The search page will be implemented with either Episerver Find (choose this if you will use DXC Service or you have paid for an Episerver Find license) or Episerver Search. To use Episerver Search, skip ahead to **Creating a search page type** on page 161, otherwise complete the following tasks.

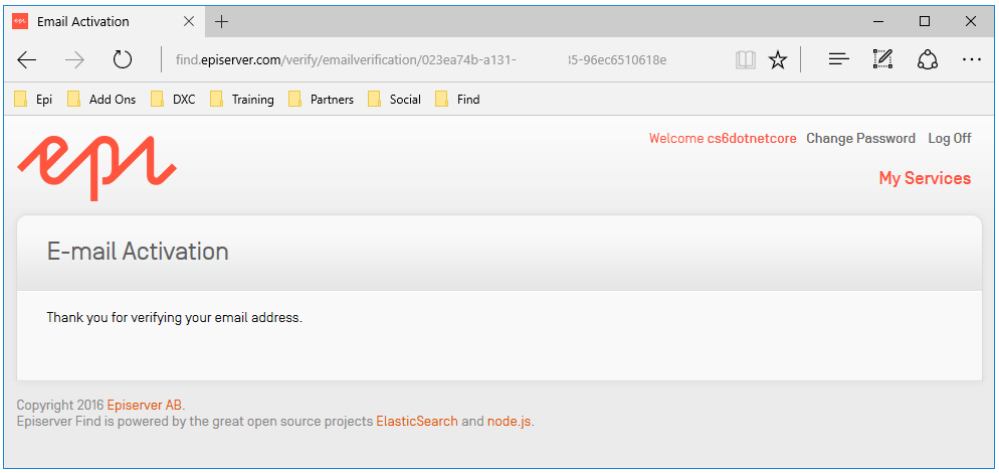
Signing up for a free Episerver Find index

1. Open a web browser and navigate to `http://find.episerver.com/`
2. Click **Sign up for a demo 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.



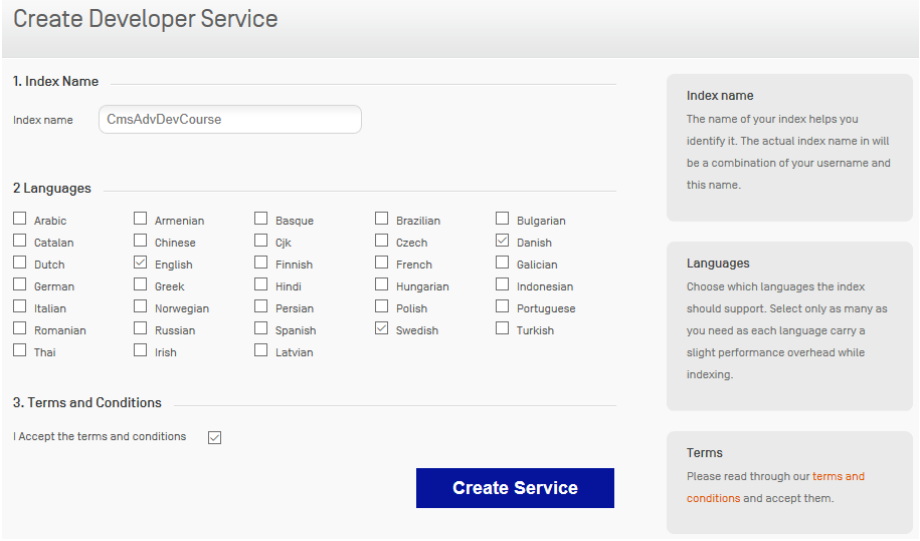
If you don't receive your verification email, check your junk mail folder!

5. On the **E-mail Activation** verification page, click **My Services**, as shown in the following screenshot:

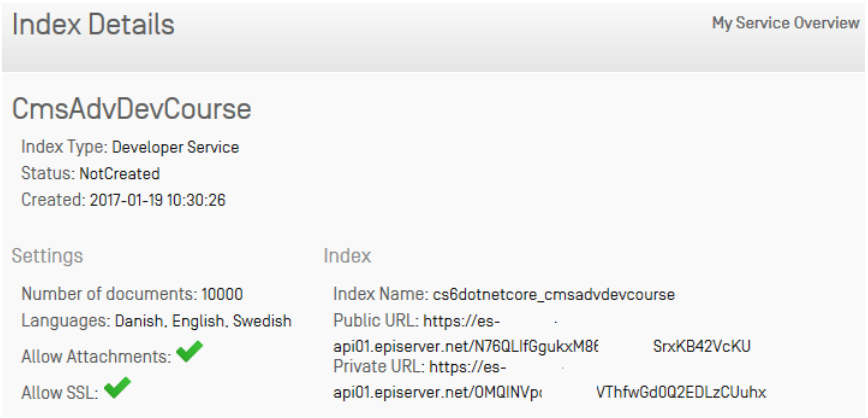


6. In **My Services**, click **Add Developer Service**.
7. Fill in a name for your index. The name is technically irrelevant but should preferably represent what you're using the index for, such as, ***CustomerName*TestIndex**.

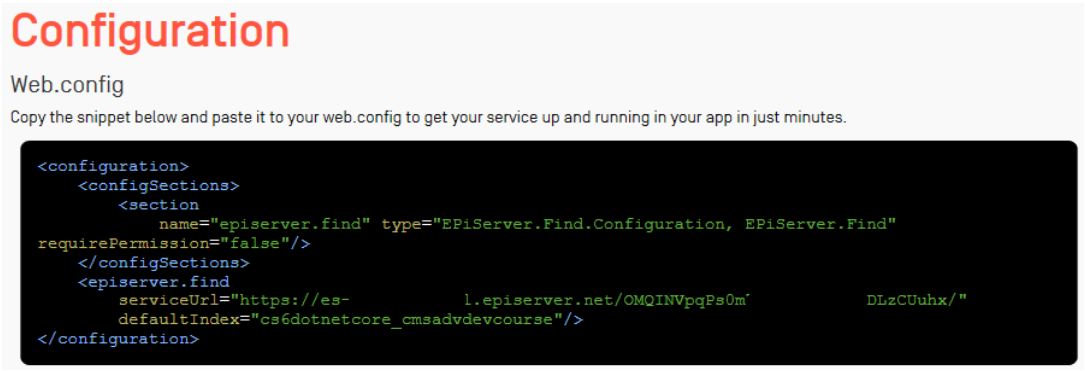
8. Select at least **English**, **Swedish**, and **Danish** from the list of languages, check the terms and conditions checkbox, and click **Create Service**, as shown in the following screenshot:



9. 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**.



10. 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 Episerver Find

1. Open the solution with the **AlloyTraining** project.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.

3. Enter the following commands:

```
Install-Package -ProjectName AlloyTraining EPiServer.Find.Cms
Update-EPiDatabase
```

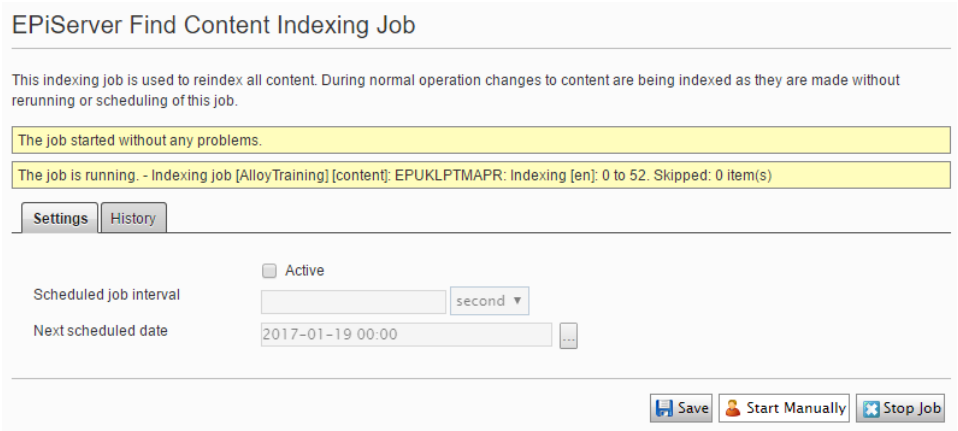
4. Open **~\Web.config**.
5. Add the following to the **<configSections>** element (you can copy and paste it from the Find Configuration page):

```
<section name="episerver.find" requirePermission="false"
      type="EPiServer.Find.Configuration, EPiServer.Find" />
```

6. 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-eu-api01.episerver.net/PlS....kxv"
    defaultIndex="episervertraining_index58384"/>
```

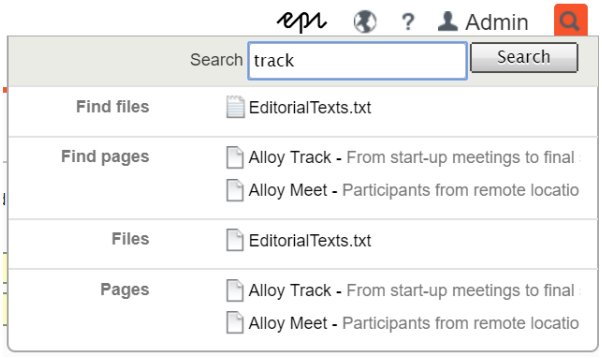
7. Start the site, and log in as a CMS admin.
8. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **EPiServer Find Content Indexing Job**.
9. Click **Start Manually**, and wait for the job to complete, as shown in the following screenshot:



10. Click **History**. Note the number of content items 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

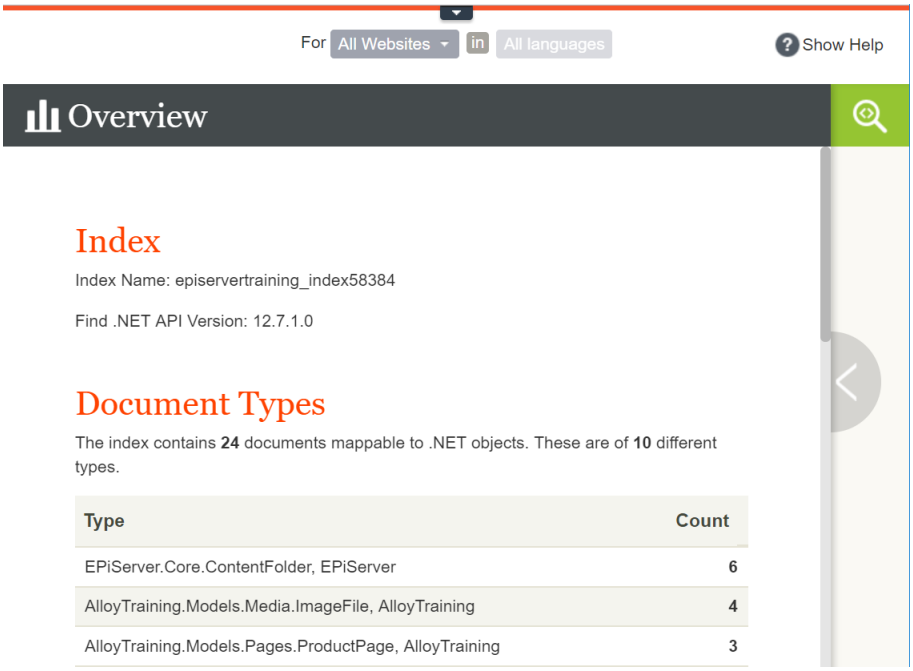
11. In the **Global** menu, click inside the **Search** box, and enter **track**. You should see multiple matches, with results shown from both Episerver Search (labeled **Files** and **Pages**) and Episerver Find (labeled **Find files** and **Find pages**), as shown in the following screenshot:



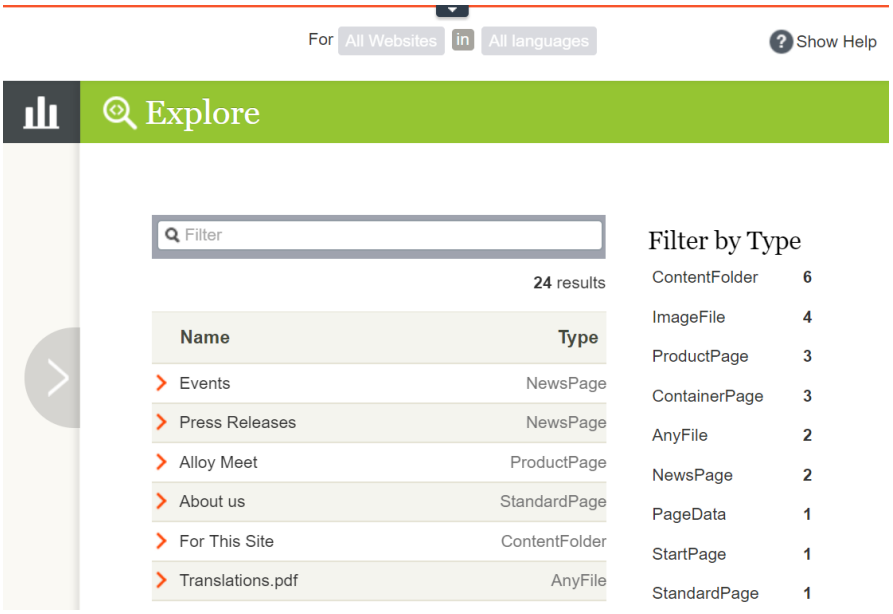
12. Navigate to **CMS** | **Admin** | **Config** | **Search Configuration**. Note how the **Search Providers** have check boxes to disable them in **Global Search**, and they can be dragged and dropped to change the order in which they are called and displayed in the search results.

Reviewing the administrator's view of Episerver 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. Close the browser.

Creating a search page type

These steps apply for both Episerver Search and Episerver Find.

1. In **AlloyTraining**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.

2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **SearchPage.cs** for the Name, and click **Add**.
3. Modify the class to inherit from **SitePageData**.
4. Change the **DisplayName** to **Search**.
5. Group the page type under **Specialized**.
6. Add a **Description** of "Use this to enable visitors to search for pages and media on the site."
7. Apply the **[SiteSearchIcon]** attribute to the class.
8. Delete the **MainBody** property.

Your code should look something like the following:

```
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;

namespace AlloyTraining.Models.Pages
{
    [ContentType(DisplayName = "Search",
        GroupName = SiteGroupNames.Specialized,
        Order = 30,
        Description = "Use this to enable visitors to search for pages and
media on the site.")]
    [SiteSearchIcon]
    public class SearchPage : SitePageData
    {
    }
}
```

Creating a search page view model

1. In **AlloyTraining**, expand **Models**, right-click **ViewModels**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Code**, choose **Class**, enter **SearchPageViewModel.cs** for the Name, and click **Add**.
3. Inherit from `DefaultPageViewModel<SearchPage>` and fix the compile error using autofix to add the required constructor.
4. Modify the contents, to add some properties for the visitor's free-text search term and for the results, as shown in the following code:

```
using AlloyTraining.Models.Pages;
using System.Collections.Generic;

namespace AlloyTraining.Models.ViewModels
{
    public class SearchPageViewModel : DefaultPageViewModel<SearchPage>
    {
        public SearchPageViewModel(SearchPage currentPage) : base(currentPage)
        {
        }

        public string SearchText { get; set; }
        public List<Result> SearchResults { get; set; }
    }

    public class Result
    {
        public string Title { get; set; }
        public string Description { get; set; }
        public string Url { get; set; }
```

```
        }
    }
```

We need to declare a class to represent each result because we want to keep search results abstract and not tied to a specific technology like Episerver Find or Episerver Search.

Implementing a search page controller using Episerver Find

To use Episerver Search, skip ahead to **Implementing a search page controller using Episerver Search** on page 164.

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **SearchPageController.cs** for the name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace.
4. Modify the class to inherit from **PageControllerBase**, create an instance of the search page view model and set its properties, before passing it to a view, as shown in the following code:

```
using AlloyTraining.Business.ExtensionMethods;
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using EPiServer.Find;
using EPiServer.Find.Cms;
using EPiServer.Find.Framework;
using EPiServer.Security;
using System.Linq;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class SearchPageController : PageControllerBase<SearchPage>
    {
        public SearchPageController(IContentLoader loader) : base(loader)
        {
        }

        public ActionResult Index(SearchPage currentPage, string q)
        {
            var viewmodel = new SearchPageViewModel(currentPage);

            if (!string.IsNullOrWhiteSpace(q))
            {
                var query = SearchClient.Instance
                    .Search<SitePageData>() // 1. only pages
                    .For(q)                 // 2. free-text query
                    // 3. only what the current user can read
                    .FilterForVisitor()
                    // 4. only under the Start page (to exclude Wastebasket)
                    .FilterOnCurrentSite();

                var results = query.GetContentResult();

                viewmodel.SearchText = q;

                viewmodel.SearchResults = results
                    .Select(x => new Result
                    {
```

```
                    Title = x.MetaTitle ?? x.Name,
                    Description = x.MetaDescription.TruncateAtWord(20),
                    Url = x.PageLink.ExternalURLFromReference()
                }).ToList();
            }
            return View(viewmodel);
        }
    }
}
```

Implementing a search page controller using Episerver Search

If you have already implemented the search page controller using Episerver Find, skip ahead to **Creating a search page view** on page 165.

1. In **AlloyTraining**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **SearchPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyTraining.Models.Pages** namespace.
4. Modify the class to inherit from **PageControllerBase**, create an instance of the search page view model and set its properties, before passing it to a view, using extension methods to generate the URL to the page and to truncate the descriptions of the pages to 20 words each, as shown in the following code:

```
using AlloyTraining.Business.ExtensionMethods;
using AlloyTraining.Models.Pages;
using AlloyTraining.Models.ViewModels;
using EPiServer.Core;
using EPiServer.Search;
using EPiServer.Search.Queries.Lucene;
using EPiServer.Security;
using System.Linq;
using System.Web.Mvc;

namespace AlloyTraining.Controllers
{
    public class SearchPageController : PageControllerBase<SearchPage>
    {
        private readonly SearchHandler searchHandler;

        public SearchPageController(IContentLoader loader,
            SearchHandler searchHandler) : base(loader)
        {
            this.searchHandler = searchHandler;
        }

        public ActionResult Index(SearchPage currentPage, string q)
        {
            var viewmodel = new SearchPageViewModel(currentPage);

            if (!string.IsNullOrWhiteSpace(q))
            {
                // 1. only pages
                var onlyPages = new ContentQuery<SitePageData>();

                // 2. free-text query
                var freeText = new FieldQuery(q);
```

```
                // 3. only what the current user can read
                var readAccess = new AccessControlListQuery();
                readAccess.AddAclForUser(PrincipalInfo.Current, HttpContext);

                // 4. only under the Start page (to exclude Wastebasket, for example)
                var underStart = new VirtualPathQuery();
                underStart.AddContentNodes(ContentReference.StartPage);

                // build the query from the expressions
                var query = new GroupQuery(LuceneOperator.AND);
                query.QueryExpressions.Add(freeText);
                query.QueryExpressions.Add(onlyPages);
                query.QueryExpressions.Add(readAccess);
                query.QueryExpressions.Add(underStart);

                // get the first page of ten results
                SearchResults results = searchHandler.GetSearchResults(query, 1, 10);

                viewmodel.SearchText = q;

                viewmodel.SearchResults = results.IndexResponseItems
                    .Select(x => new Result
                    {
                        Title = x.Title,
                        Description = x.DisplayText.TruncateAtWord(20),
                        Url = x.GetExternalUrl().ToString()
                    }).ToList();
            }
            return View(viewmodel);
        }
    }
}
```

Creating a search page view

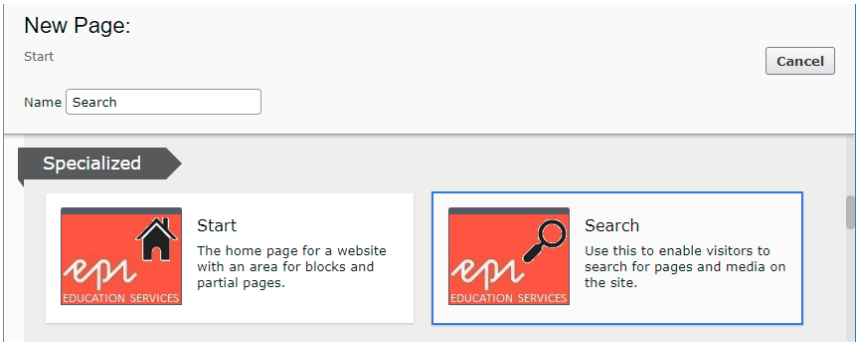
1. In **AlloyTraining**, right-click **Views**, and add a new folder named **SearchPage**.
2. Right-click **SearchPage**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyTraining**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
4. Modify the view, as shown in the following markup, and note the following:
    a. To support searching in Edit view, the form must POST instead of GET.
    b. If there is search text, the Search Results are shown, and then either shows a warning if there are no matches, or the total hits and the list of results.

```
@using EPiServer.Editor
@model AlloyTraining.Models.ViewModels.SearchPageViewModel
<div class="row">
    <div class="span8">
        @using (Html.BeginForm(actionName: null,
          controllerName: null, routeValues: null,
          method: PageEditing.PageIsInEditMode ? FormMethod.Post : FormMethod.Get))
        {
            <input tabindex="1" name="q" value="@Model.SearchText" />
            <input type="submit" tabindex="2" class="btn" value="Search" />
        }
        @if (!string.IsNullOrWhiteSpace(Model.SearchText))
        {
            <div class="row">
```

```
                <div class="span8 grayHead">
                    <h2>Search Results</h2>
                </div>
            </div>
            if (Model.SearchResults.Count == 0)
            {
                <div class="row">
                    <div class="span8 SearchResults">
                        <span class="label label-warning">No matching results.</span>
                    </div>
                </div>
            }
            else
            {
                <div class="row">
                    <div class="span8 SearchResults">
                        <span class="label label-success">@Model.SearchResults.Count
matching results.</span>
                        @foreach (var item in Model.SearchResults)
                        {
                            <div class="listResult">
                                <h3><a href="@item.Url">@item.Title</a></h3>
                                <p>@item.Description</p>
                                <hr />
                            </div>
                        }
                    </div>
                </div>
            }
        }
    </div>
</div>
```

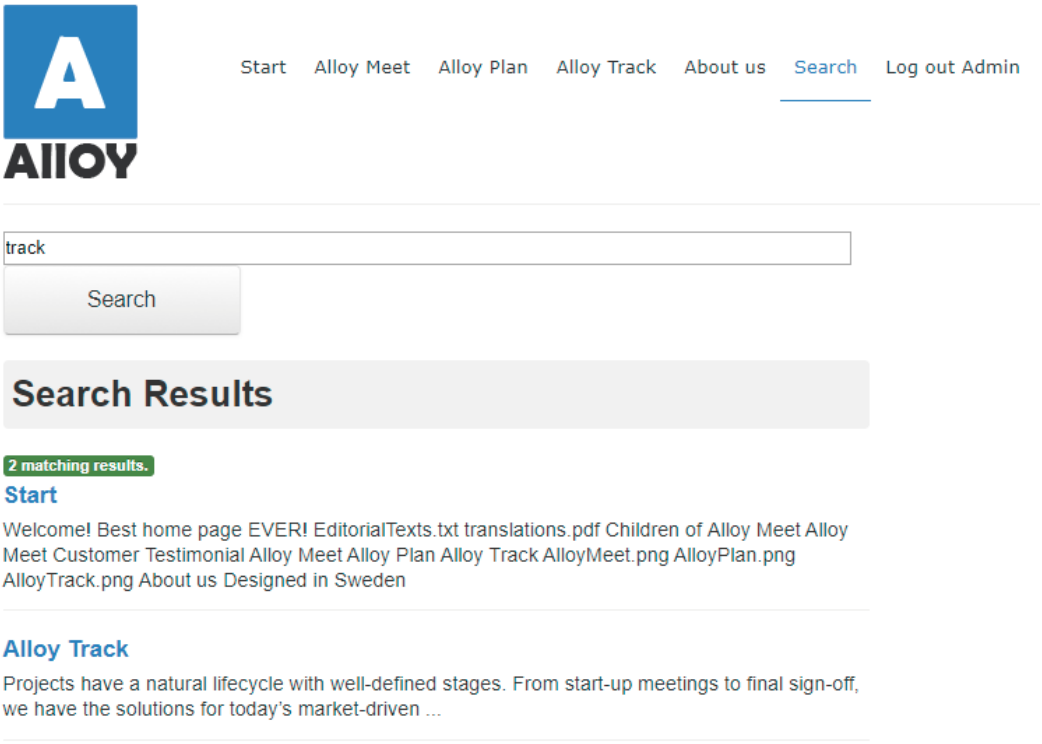
Creating and testing the search page

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Add a new **Search** page named **Search** under the **Start** page, as shown in the following screenshot:



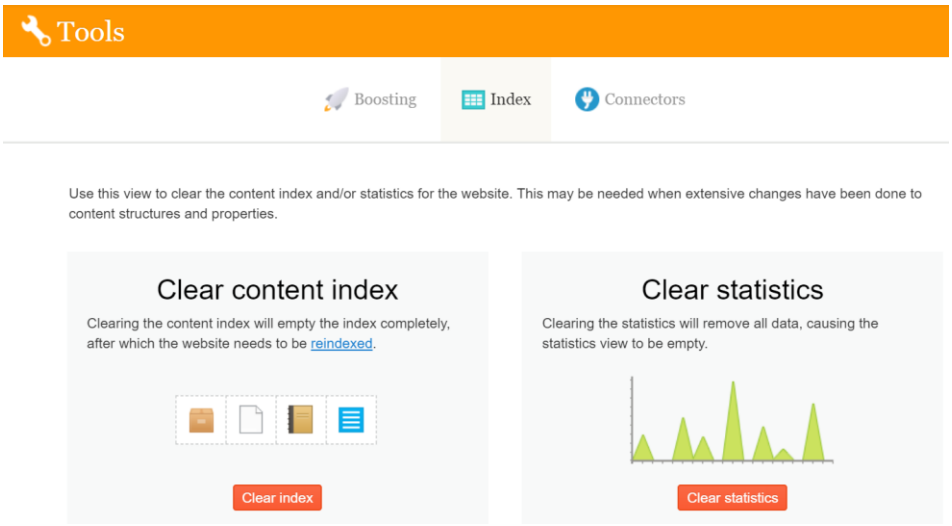
3. Publish the **Search** page.
4. Switch to **Live view** and click **Search** to navigate to the **Search** page as a visitor.

5. Enter the search text **track**, and press *ENTER* or click **Search**, as shown in the following screenshot:



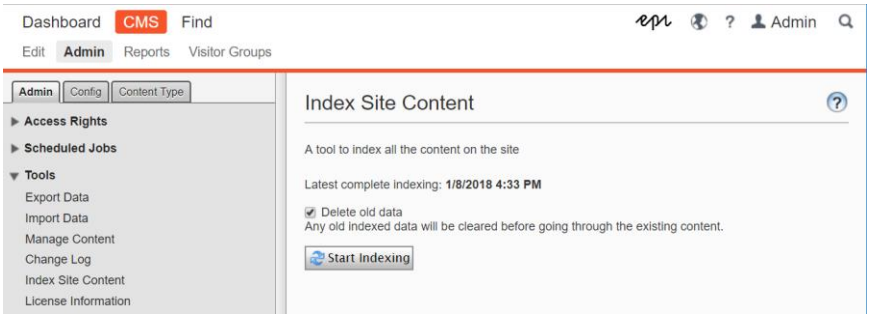
Resetting the search index for Episerver Find

1. Navigate to **Find** | **Configure** | **Index**.
2. Click **Clear Index**, as shown in the following screenshot:



Resetting the search index for Episerver Search

1. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Index Site Content**, as shown in the following screenshot:



2. Note the date and time of the latest complete indexing, and then click **Start Indexing**.
3. Press *F5* to refresh the page and see the latest complete indexing date and time.

Exercise E5 – Optional: Adding a search box to the top navigation menu

In this exercise, you will implement a search box to use the logic that was added in the Search Page in the previous exercise to perform searches from other places in your site, giving visitors to the site a better experience and more options.

You will add a search field and a button to the existing main navigation control, add a new property to the start page that holds a reference to the Search page and then write logic that, when the search button in the main navigation is clicked, adds the search text to the query string and does a redirect to the search page.

Prerequisites: complete Exercises B1 – B4, E4.

Adding a property to the Start page to reference the Search page

1. In **AlloyTraining**, open **~\Models\Pages\StartPage.cs**, and add a property to reference an instance of the search page type, as shown in the following code:

```
[Display(Name = "Search page",
    Description = "If you add a Search page to the site, set this property to
reference it to enable search from every page.",
    GroupName = SiteTabNames.SiteSettings,
    Order = 40)]
[AllowedTypes(typeof(SearchPage))]
public virtual PageReference SearchPageLink { get; set; }
```

Adding a search box to the navigation menu

1. In **AlloyTraining**, open **~\Views\Shared_NavigationMenu.cshtml**.
2. At the top of the view, add a statement to import Alloy's extension methods, as shown in the following code:

```
@using AlloyTraining.Business.ExtensionMethods
```

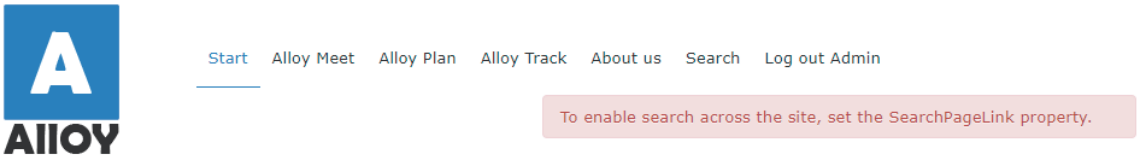
3. After the element, inside the nested elements, add the following markup:

```
<div class="navbar-search pull-right">
    @if (PageReference.IsNullOrEmpty(Html.GetStartPage().SearchPageLink))
    {
        if (EPiServer.Editor.PageEditing.PageIsInEditMode)
        {
            <div class="alert alert-danger">To enable search across the site,
                set the SearchPageLink property.</div>
        }
    }
    else
    {
        <form action="@Html.GetStartPage().SearchPageLink.ExternalURLFromReference()"
              method="post">
            <input type="text" class="search-query" name="q"
                   id="SearchKeywords" placeholder="Search" />
            <input type="submit" class="searchButton" id="SearchButton" value="" />
        </form>
    }
```

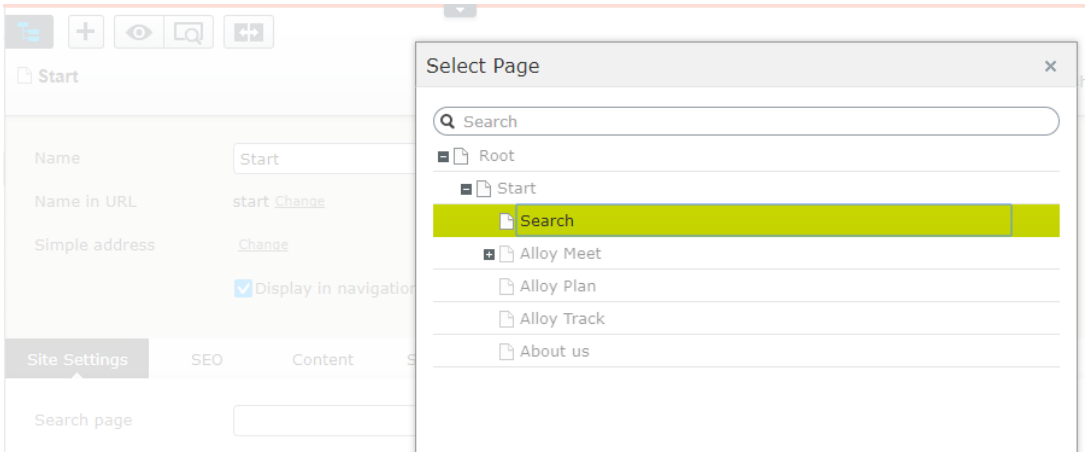
```
</div>
```

Enable the search box and test the website

1. Start the **AlloyTraining** website, and log in as a CMS admin.
2. Edit the **Start** page, and note the warning for editors, as shown in the following screenshot:



3. Switch to **All Properties** view, click **Site Settings**, and set the **Search page** property to reference the **Search** page, as shown in the following screenshot:



4. Publish the **Start** page.
5. Switch to **Live view**.
6. Navigate to any page, enter **track** in the search box, click the search button or press *ENTER*, and note you are directed to the **Search** page to see the results.

Module F – Working with Episerver Framework

Goal

The overall goal of the exercises in this module is to learn how to work with Episerver Framework and content APIs. You will:

1. Export and import content.
2. Implement FAQs by programmatically creating pages using content APIs.
3. Validate content by listening for publishing events.
4. Process content by implementing a scheduled job.
5. Decompile Episerver assemblies to better understand how to work with our APIs.

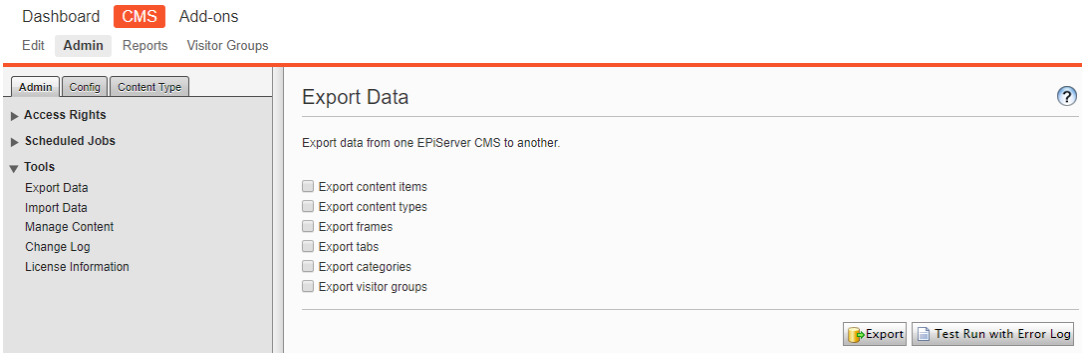
Exercise F1 – Optional: Exporting and importing content

In this exercise, you will export data from an Episerver CMS site, and then import it back in as if it were new content.

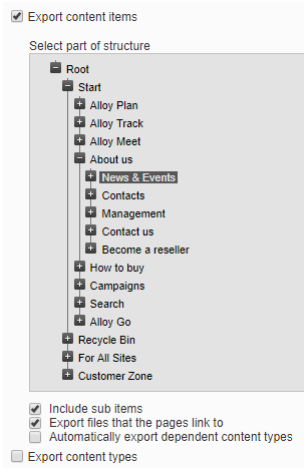
Prerequisites: complete Exercise A1.

Exporting news & events from AlloyDemo website

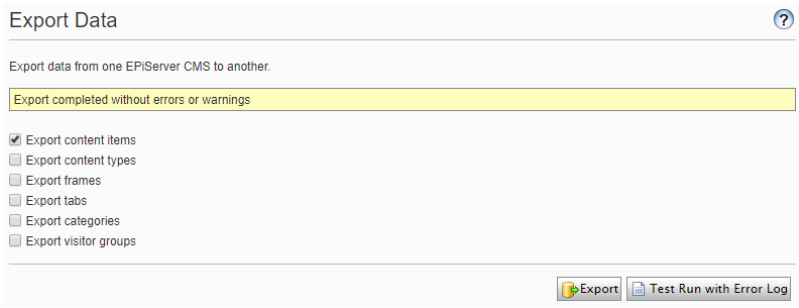
1. Open the **Training** solution with the **AlloyDemo** project.
2. Start the **AlloyDemo** website, and log in as **Admin**.
3. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Export Data**, as shown in the following screenshot:



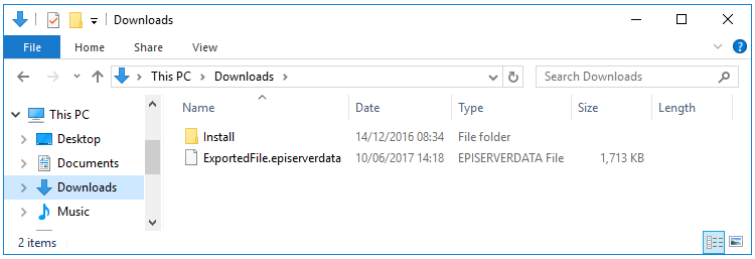
4. Click **Export content items**, and select **News & Events**, and note that by default the export will include sub items and any files (media assets) that the pages link to, as shown in the following screenshot:



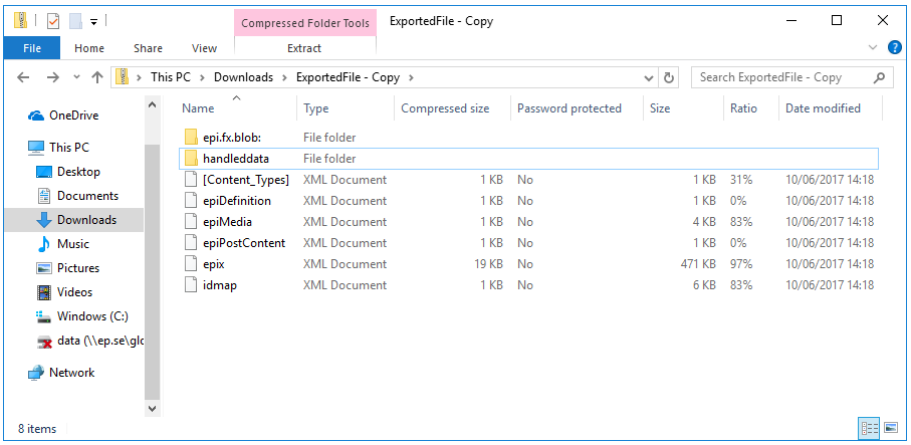
5. Click **Test Run with Error Log**, and note that you get no errors or warnings, as shown in the following screenshot:



6. Click **Export**.
7. Start **File Explorer**, open your **Downloads** folder, and note that a file has been created named **ExportedFile.episerverdata**, as shown in the following screenshot:



8. Copy the file, and change its file extension to ZIP.
9. Open the ZIP file, and note the contents, as shown in the following screenshot:

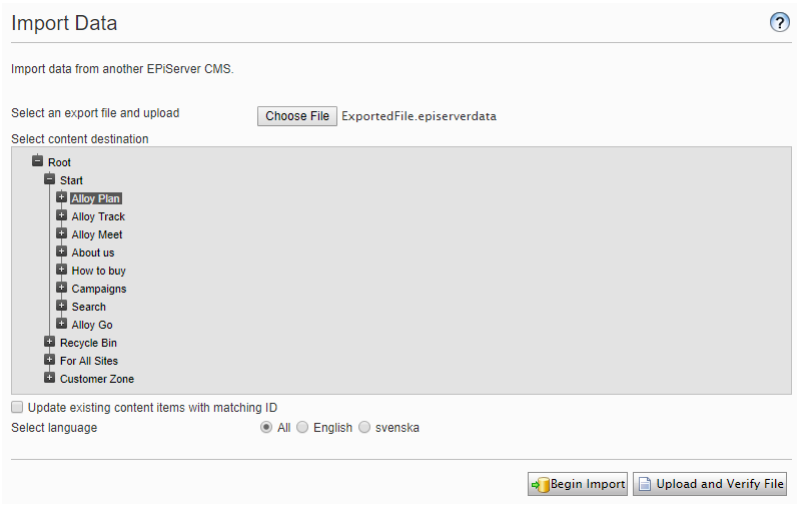


The **epi.fx.blob:** folder contains the media assets like images used in the news & event articles exported. The **epix.xml** file contains the page and block content and their property values.

Importing news & events to AlloyDemo website

1. Start the **AlloyDemo** website, and log in as **Admin**.
2. Navigate to **CMS** | **Admin** | **Admin** | **Tools** | **Import Data**, choose the **ExportedFile.episerverdata** file, and select **Alloy Plan** as the content destination.

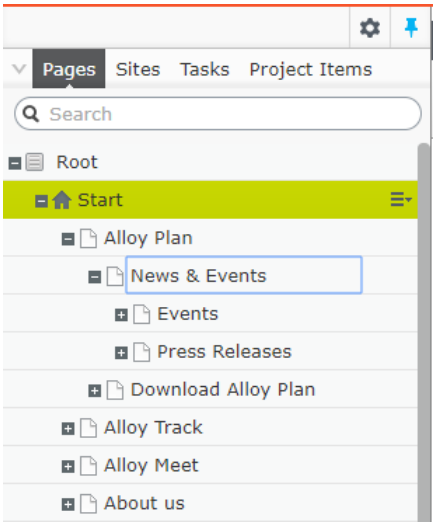
3. Clear the **Update existing content items with matching ID** check box, and click **Begin Import**, as shown in the following screenshot:



If you do not clear the **Update existing content items with matching ID** check box, then the pages will appear not to import, because the it will just update existing items rather than add pages as new items!

4. After a few moments, the import should complete, as shown in the following screenshot:

5. Navigate to **CMS** | **Edit**, and expand the **Pages** tree to see that **Alloy Plan** now has some new child pages, as shown in the following screenshot:

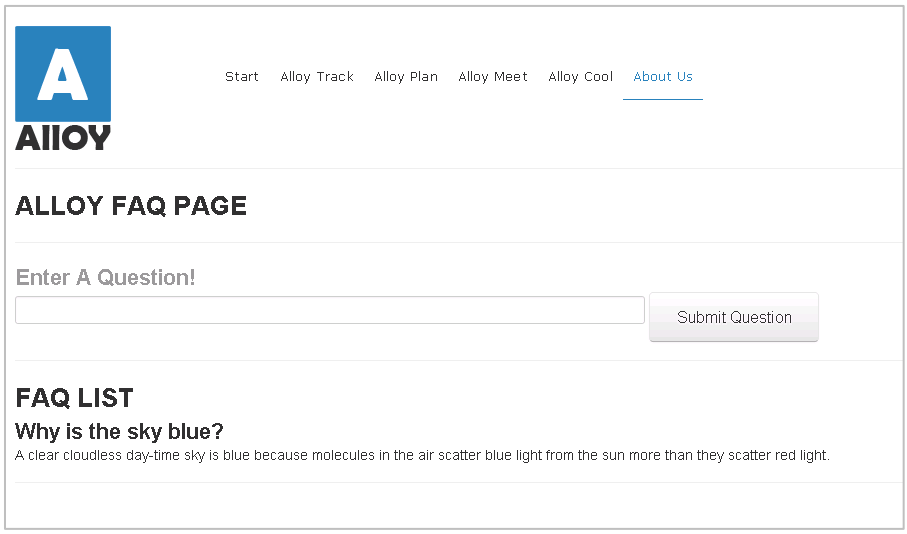


Exercise F2 – Optional: Implementing FAQs with content APIs

In this exercise, you will implement FAQs by implementing data pages and automating their creation using the content repository API.

You will create two page types; one to be used as the visible **FAQListPage** containing both a form for visitors to enter a question and a listing of answered questions, and one named **FAQItemPage** that will act as a data page for a single question and its answer.

Here's an example of an FAQ page with one question answered:



Prerequisites: complete Exercise A1.

Create an FAQ item data page type and an FAQ list page type

The **FAQItemPage** will not have a template because it will just store data, but we want it to be created as a child of a **FAQListPage** that will have a page template that shows a list of its children.

1. Open the **Training** solution with the **AlloyDemo** project.
2. In **Solution Explorer**, in **AlloyDemo**, expand **Models**, right-click **Pages**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **FAQItemPage.cs** for the Name, and click **Add**.
4. Change the **DisplayName** to **FAQ Item**.
5. Add a **Description** of "A data page for an FAQ item (cannot be created by editors)."
6. Add the **AvailableInEditMode** parameter and set it to **false**.
7. Delete the **MainBody** property.
8. Add the following properties with appropriate attributes:
    a. **Question:** XhtmlString
    b. **Answer**: XhtmlString

FAQItemPage does not need an icon because content editors cannot see it in edit view. It does not need to inherit from SitePageData because it will not render as an HTML page with META elements.

Your code should look something like the following:

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

```
using System.ComponentModel.DataAnnotations;

namespace AlloyDemo.Models.Pages
{
    [ContentType(DisplayName = "FAQ Item",
        Description = "A data page for an FAQ item (cannot be created by editors).",
        AvailableInEditMode = false)]
    public class FAQItemPage : PageData
    {
        [Display(Name = "Question", Order = 10)]
        public virtual XhtmlString Question { get; set; }

        [Display(Name = "Answer", Order = 20)]
        public virtual XhtmlString Answer { get; set; }
    }
}
```

9. In **AlloyDemo**, expand **Models**, right-click **Pages**, and click **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
10. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **FAQListPage.cs** for the name, and click **Add**.
11. Change the **DisplayName** to **FAQ List**.
12. Group the page type under **Specialized** by using a constant in **Global.GroupNames**.
13. Add a **Description** of "Use this page for a list of FAQs entered by visitors, answered by editors."
14. Apply the **SiteImageUrl** attribute to the class.
15. Apply an attribute to restrict this page types children to only be **FAQItemPage** instances.
16. Inherit from **SitePageData**.
17. Delete the **MainBody** property.
18. Add a property named **FAQItems** that can contain the child FAQ item pages and apply the **Ignore** attribute to it.

Good practice would be to define a view model to hold this property but marking a content type property with **[Ignore]** means the property will not be registered with the CMS so it can be used for a similar purpose as a view model. Just beware of object caching!

Your code should look something like the following:

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

namespace AlloyDemo.Models.Pages
{
    [ContentType(DisplayName = "FAQ List",
        GroupName = Global.GroupNames.Specialized,
        Description = "Use this page for a list of FAQs entered by visitors,
answered by editors.")]
    [SiteImageUrl]
    [AvailableContentTypes(Include = new[] { typeof(FAQItemPage) },
        IncludeOn = new[] { typeof(StartPage) })]
    public class FAQListPage : SitePageData
    {
        // having an ignored property avoids needing a view model
        // this property will not be stored in CMS so it does not
        // need to be virtual
        [Ignore]
        public IEnumerable<FAQItemPage> FAQItems { get; set; }
    }
```

```
    }
```

Creating an FAQ list page template

1. In **AlloyDemo**, expand **Controllers**, and right-click and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **FAQListPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyDemo.Models.Pages** namespace.
4. Modify the class to derive from **PageControllerBase<T>**.
5. Modify the **Index** action method to use a view model and set the **FAQItems** property using the content loader service.
6. Add a **CreateFAQItem** method, with two parameters: **currentPage** and **question**.
7. Use the content repository service to add a new FAQ item page under the current page, setting its **Question** property to the **question** parameter, and giving it a **Name**.
8. Save and check in the new FAQ item page if the current user has **Read** access rights.

Your controller should look something like the following code:

```
using AlloyDemo.Models.Pages;
using AlloyDemo.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using System.Web.Mvc;

namespace AlloyDemo.Controllers
{
    public class FAQListPageController : PageControllerBase<FAQListPage>
    {
        private readonly IContentRepository repo;

        public FAQListPageController(IContentRepository repo)
        {
            this.repo = repo;
        }

        public ActionResult Index(FAQListPage currentPage)
        {
            var viewmodel = PageViewModel.Create(currentPage);
            var faqs = repo.GetChildren<FAQItemPage>(currentPage.ContentLink);
            viewmodel.CurrentPage.FAQItems = faqs;
            return View(viewmodel);
        }

        public ActionResult CreateFAQItem(FAQListPage currentPage, string question)
        {
            var faqItem = repo.GetDefault<FAQItemPage>(currentPage.ContentLink);

            // if someone is logged in then CreatedBy and ChangedBy will be set,
            // otherwise they will be empty string which is shown as "installer"
            if (string.IsNullOrEmpty(faqItem.CreatedBy))
                 faqItem.CreatedBy = "Anonymous";
            if (string.IsNullOrEmpty(faqItem.ChangedBy))
                faqItem.ChangedBy = "Anonymous";

            faqItem.Question = new XhtmlString(question);
            faqItem.Name = "Q. " + question;
            repo.Save(faqItem,
```

```
                EPiServer.DataAccess.SaveAction.CheckOut,
                EPiServer.Security.AccessLevel.Read);

            return RedirectToAction("Index");
        }
    }
}
```

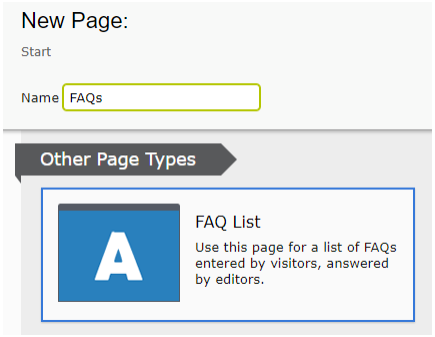
9. In **AlloyDemo**, right-click **Views**, and add a new folder named **FAQListPage**.
10. Right-click **FAQListPage**, and choose **Add** | **New Item...**, or press *Ctrl* + *Shift* + *A*.
11. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
12. Modify the view, as shown in the following markup, and note the following:
    a. The question form submits back to the **CreateFAQItem** action method.
    b. Content editors see a warning message.
    c. Each FAQ item outputs its: question and when it was asked, and the answer and who and when it was answered.

```
@model AlloyDemo.Models.ViewModels.PageViewModel<FAQListPage>
<h2>Alloy FAQs</h2>
<div class="navbar">
    <h3>Enter your question</h3>
    @using (Html.BeginForm(actionName: "CreateFAQItem", controllerName: null))
    {
        <input type="text" tabindex="1" name="question"
               placeholder="Enter your question" />
        <input type="submit" tabindex="2" class="btn" value="Submit" />
    }
</div>
@if (EPiServer.Editor.PageEditing.PageIsInEditMode)
{
    <div class="alert alert-info">
        Questions do not appear until they have been answered by a
        content editor and published.
    </div>
}
<div class="nav-stacked">
    <h2>Questions with answers</h2>
    @foreach (var faqItem in Model.CurrentPage.FAQItems)
    {
        @Html.Raw(faqItem.Question)
        <p>Asked at @faqItem.Created.ToShortTimeString() on
            @faqItem.Created.ToShortDateString()</p>
        <p>Answered by @faqItem.ChangedBy at
            @faqItem.StartPublish.Value.ToShortTimeString() on
            @faqItem.StartPublish.Value.ToShortDateString()</p>
        @Html.Raw(faqItem.Answer)
    }
</div>
```

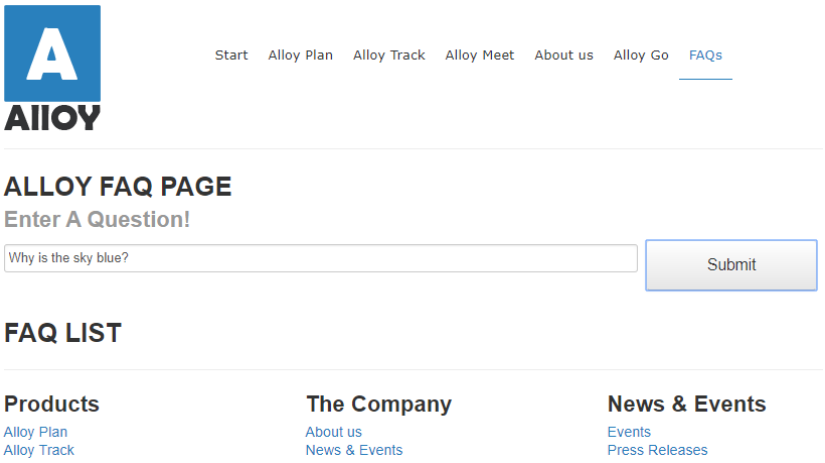
Test the website FAQ functionality

1. Start the **AlloyDemo** website, and log in as **Admin**.

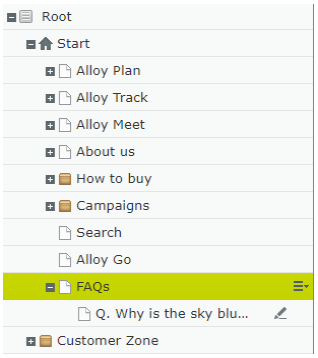
2. Create an FAQ list page named **FAQs** under the **Start** page, as shown in the following screenshot:



3. **Publish** the page.
4. Switch to **Live** view to experience the website as a visitor.
5. Navigate to the **FAQs** page.
6. Enter and submit a question, for example, “Why is the sky blue?”, and note that the FAQ does NOT appear, as shown in the following screenshot:



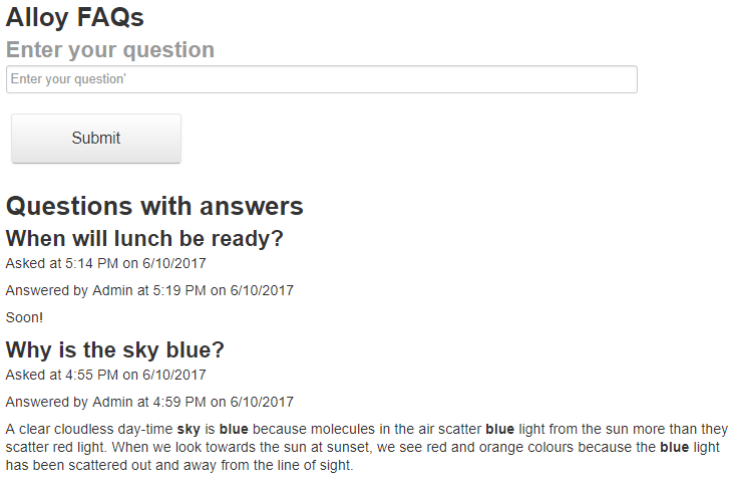
7. Log in as **Admin**., view the **Pages** tree, and note the **FAQs** page has a child with a pencil icon to indicate that it has a saved draft but has not been published yet, as shown in the following screenshot:



If the browser was already running, press F5 to refresh the page to see the newly created FAQ item page.

8. Edit the question, answer it, and publish it.

9. Switch to Live view, and note the question and its answer now appear, as shown in the following screenshot:



Exercise F3 – Optional: Listening for events and customizing services with initialization modules

In this exercise, you will create initialization modules that (1) prevents creating a page if the parent page already has more than a maximum number of children, and (2) removes the service that suggests most recent content types.

Prerequisites: complete Exercise A1.

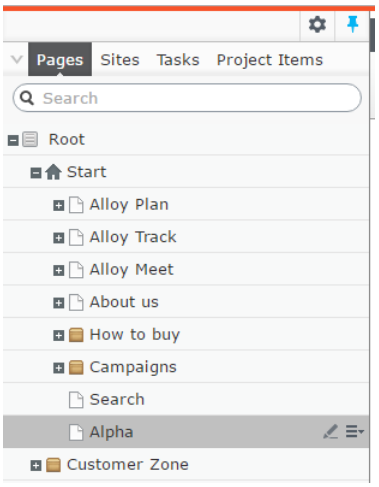
Creating an initialization module to listen for events

1. In **~\Business\Initialization**, add an **Episerver** | **Initialization Module** named **PreventMoreThanMaxChildrenInitializationModule.cs**.
2. Modify the statements, as shown in the following code:

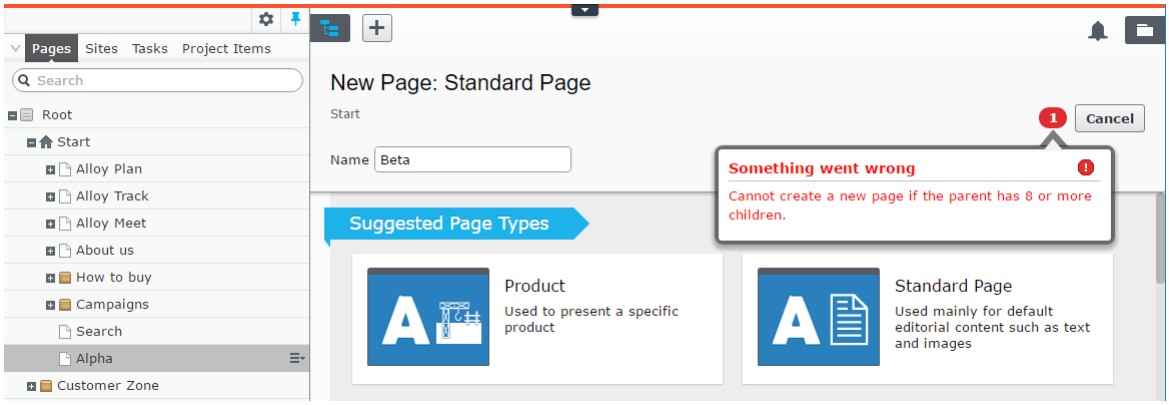
```
using AlloyDemo.Models.Pages;
using EPiServer;
using EPiServer.Core;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using System.Linq;

namespace AlloyDemo.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class PreventMoreThanMaxChildrenInitializationModule : IInitializableModule
    {
        private bool initialized = false;
        private IContentEvents events;
        private IContentLoader loader;
        private const int maxChildren = 8;

        public void Initialize(InitializationEngine context)
        {
            if (!initialized)
            {
                loader = context.Locate.ContentLoader();
                events = context.Locate.ContentEvents();
                events.CreatingContent += Events_CreatingContent;
                initialized = true;
            }
        }

        private void Events_CreatingContent(object sender, ContentEventArgs e)
        {
            var sitepage = e.Content as SitePageData;
            if (sitepage != null)
            {
                var children = loader.GetChildren<IContent>(sitepage.ParentLink);
                if (children.Count() >= maxChildren)
                {
                    e.CancelAction = true;
                    e.CancelReason =
        $"Cannot create a new page if the parent has {maxChildren} or more children.";
                }
            }
        }

        public void Uninitialize(InitializationEngine context)
        {
```

```
            if (initialized)
            {
                events.CreatingContent -= Events_CreatingContent;
            }
        }
    }
}
```

3. Start the site, and log in as **Admin**.
4. Navigate to **CMS** | **Edit**.
5. Add a **Standard** page underneath the **Start** page named **Alpha**. You will be able to create the page, as shown in the following screenshot:



6. **Publish** the Alpha page.
7. Add a **Standard** page underneath the **Start** page named **Beta**. This time, you will not be able to create the page, because Start now has eight children, as shown in the following screenshot:



8. Click **Cancel**.
9. Close the browser.

Limiting the number of children to eight will cause you annoyance later, so change the limit in the initialization module to something like 100, as shown in the following code:

```
private const int maxChildren = 100;
```

Creating an initialization module to remove a service

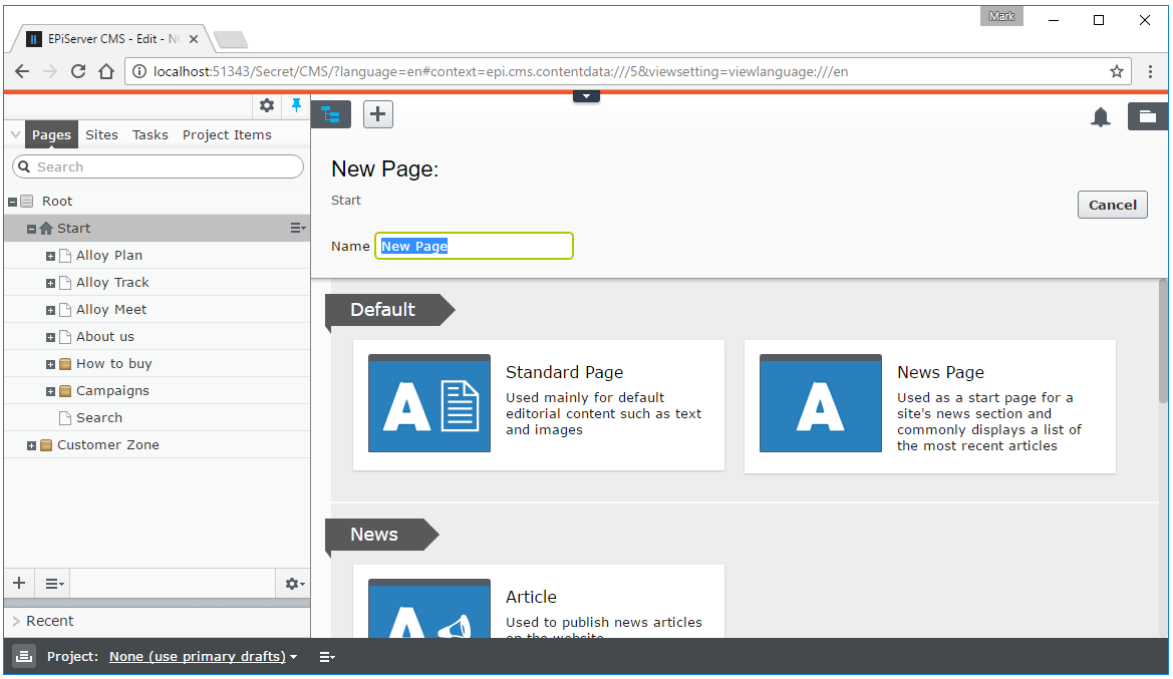
1. In the previous screenshot, note the **Suggested Page Types** group. This group is automatically created based on the most recently used page types.

2. In **~\Business\Initialization**, add an **Episerver** | **Initialization Module** named **RemoveSuggestedPageTypesInitializationModule.cs**.
3. Modify the statements, as shown in the following code:

Note the class implements **IConfigurableModule** (that itself implements **IInitializationModule**), and that the method we implement to remove the service is named **ConfigureContainer**. Both the methods **Initialize** and **Uninitialize** must exist, but in this scenario, they do nothing.

```
using EPiServer.Cms.Shell.UI.Rest;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.ServiceLocation;

namespace AlloyDemo.Business.Initialization
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class RemoveSuggestedPageTypesInitializationModule : IConfigurableModule
    {
        public void ConfigureContainer(ServiceConfigurationContext context)
        {
            context.StructureMap().EjectAllInstancesOf<IContentTypeAdvisor>();
        }

        public void Initialize(InitializationEngine context) { }

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

4. Start the site, and log in as **Admin**.
5. Navigate to **CMS** | **Edit**.
6. Add a **Standard** page underneath the **Start** page, and note the **Suggested Page Types** group has gone, as shown in the following screenshot:

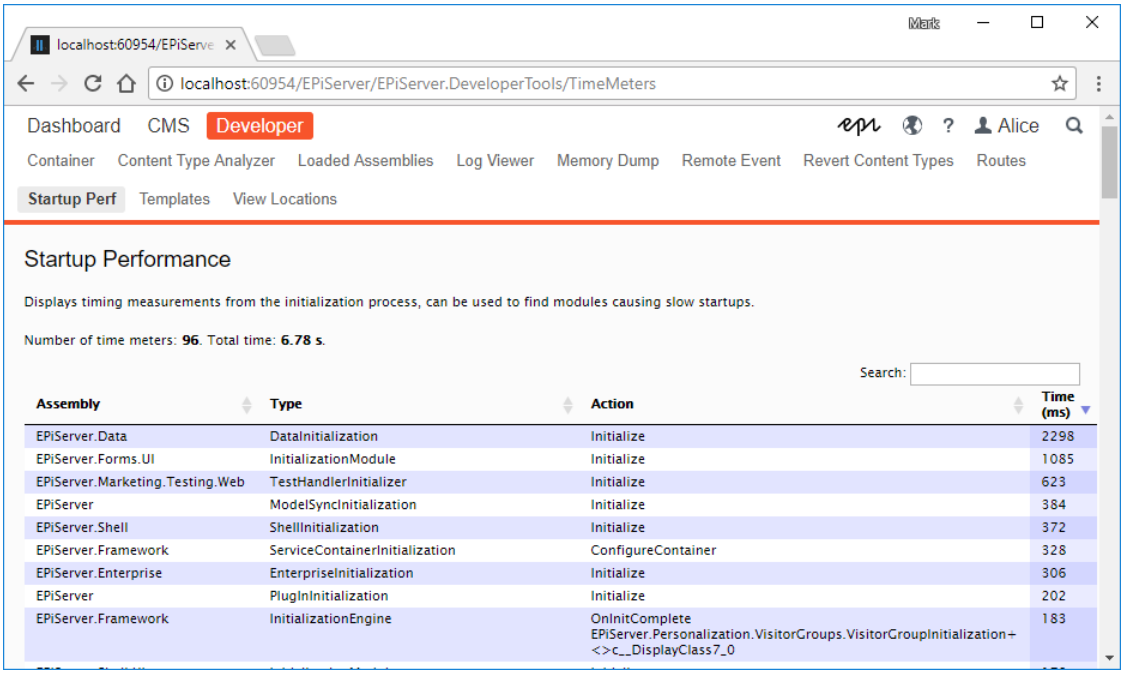


Install Episerver Developer Tools to monitor initialization modules

1. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Console**.
2. Enter the following command:

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

3. Start the website, and log in as **Admin**.
4. Navigate to **Developer** | **Startup Perf**, as shown in the following screenshot:



Exercise F4 – Optional: Implementing scheduled jobs

In this exercise, you will implement a scheduled job to modify page names, titles, and descriptions that contain bad words.

Prerequisites: complete Exercise A1.

Create a scheduled job

1. Open the solution that includes the **AlloyDemo** project.
2. In **Solution Explorer**, right-click **~/Business**, add a folder named **ScheduledJobs**, and add an Episerver **Scheduled Job** named **BadWordScannerScheduledJob**.
3. Modify its code as shown below:

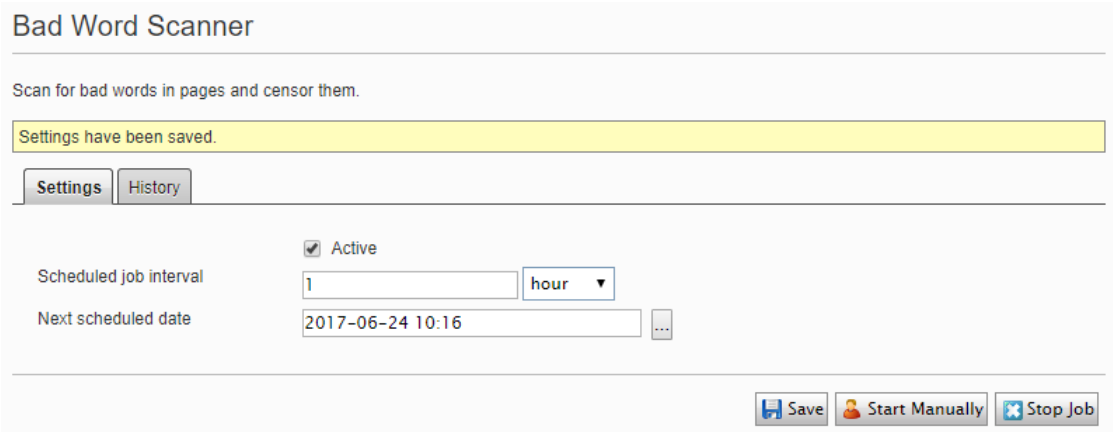
```
using AlloyDemo.Models.Pages;
using EPiServer;
using EPiServer.Core;
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using EPiServer.ServiceLocation;

namespace AlloyDemo.Business.ScheduledJobs
{
    [ScheduledPlugIn(DisplayName = "Bad Word Scanner",
        Description = "Scan for bad words in pages and censor them.")]
    public class BadWordScannerScheduledJob : ScheduledJobBase
    {
        private bool _stopSignaled;

        // you could load this from a file or service
        private string[] badWords = new[] { "frak" };

        public BadWordScannerScheduledJob()
        {
            IsStoppable = true;
        }

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

        public override string Execute()
        {
            OnStatusChanged(string.Format(
                "Starting execution of {0}", this.GetType()));

            var repo = ServiceLocator.Current.GetInstance<IContentRepository>();

            var finder = ServiceLocator.Current
                .GetInstance<IPageCriteriaQueryService>();

            int pageCount = 0;

            foreach (string word in badWords)
            {
                var criteria = new PropertyCriteriaCollection();

                criteria.Add(new PropertyCriteria
                {
                    Type = PropertyDataType.LongString,
```

```
                    Name = "PageName",
                    Condition = EPiServer.Filters.CompareCondition.Contained,
                    Value = word
                });

                criteria.Add(new PropertyCriteria
                {
                    Type = PropertyDataType.LongString,
                    Name = "MetaTitle",
                    Condition = EPiServer.Filters.CompareCondition.Contained,
                    Value = word
                });

                criteria.Add(new PropertyCriteria
                {
                    Type = PropertyDataType.LongString,
                    Name = "MetaDescription",
                    Condition = EPiServer.Filters.CompareCondition.Contained,
                    Value = word
                });

                PageDataCollection results = finder.FindPagesWithCriteria(
                    ContentReference.RootPage as PageReference, criteria);

                foreach (SitePageData page in results)
                {
                    var clone = page.CreateWritableClone() as SitePageData;

                    clone.Name = page.Name.Replace(word, "[censored]");
                    clone.MetaTitle = page.MetaTitle.Replace(word, "[censored]");
                    clone.MetaDescription =
                        page.MetaDescription.Replace(word, "[censored]");

                    repo.Save(clone,
                        EPiServer.DataAccess.SaveAction.CheckIn,
                        EPiServer.Security.AccessLevel.NoAccess);

                    pageCount++;
                }

                //For long running jobs periodically check if stop is signaled and if
so stop execution
                if (_stopSignaled)
                {
                    return "Stop of job was called";
                }
            }

            return $"{pageCount} pages containing one of the following bad words:
'{string.Join("' or '", badWords)}' have been censored.";
        }
    }
}
```

Testing the scheduled job

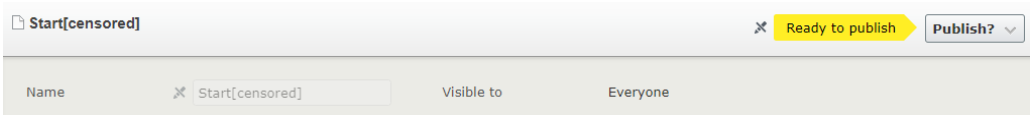
1. Start the **AlloyDemo** site, and log in as **Admin**.
2. Edit **Start**, add **frak** into its **Name**, and publish the change.
3. Edit **About us**, add **frak** into its **Title** (MetaTitle), and publish the change.
4. Edit **Alloy Track**, add **frak** into its **Page description** (MetaDescription), and publish the change.

5. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Bad Word Scanner**.
6. Set the job to be active, and set it to run every hour, and change the next schedule date to two minutes in the future, and click **Save**, as shown in the following screenshot:



7. Navigate to **CMS** | **Edit**, and wait a few minutes for the time you set to pass.
8. Navigate to **CMS** | **Admin** | **Admin** | **Scheduled Jobs** | **Bad Word Scanner**.
9. Click **History**, and note the job ran succesfully and censored three pages, as shown in the following screenshot:

10. Navigate to **CMS** | **Edit**, and note the Start page has been censored and marked as eing ready to publish, as shown in the following screenshot:



11. **Publish** the censored change.
12. Check the **About us** and **Alloy Track** pages are similiarly censored and ready to publish.

Exercise F5 – Optional: Implementing soft and hard deletes

In this exercise, you will create a page type with template to allow anonymous visitors to enter a content reference of an item on content to delete.

Prerequisites: complete Exercise A1.

Creating a DeleteContent page type

1. Open the **Training** solution with the **AlloyDemo** project.
2. In **Solution Explorer**, in **AlloyDemo**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
3. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **DeleteContentPage.cs** for the Name, and click **Add**.
4. Change the **DisplayName** to **Delete Content**.
5. Add a **Description** of "Use this to soft or hard delete content."
6. Add a **GroupName** and set to **Global.GroupNames.Specialized**.
7. Allow an instance of this page type under a Start page.
8. Inherit from **SitePageData**.
9. Delete the **MainBody** property.

Your code should look something like the following:

```
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;

namespace AlloyDemo.Models.Pages
{
    [ContentType(DisplayName = "Delete Content",
        GUID = "0f01522d-fa66-4dff-92f3-e395f2ed4f36",
        GroupName = Global.GroupNames.Specialized,
        Description = "Use this to soft or hard delete content.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class DeleteContentPage : SitePageData
    {
    }
}
```

Creating a DeleteContent page template

1. In **Solution Explorer**, in **AlloyDemo**, expand **Controllers**, and right-click and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **DeleteContentPageController.cs** for the Name, and click **Add**.
3. Fix the compilation error by clicking the light bulb, and choose the option to import the **AlloyDemo.Models.Pages** namespace.
4. Modify the class to derive from **PageControllerBase<T>**.
5. Add a constructor with **IContentRepository** passed as a parameter and store it in a private readonly field.
6. Modify the **Index** action method to use a view model.
7. Add a **Delete** method, with three parameters: **currentPage**, **contentReference**, and **hardDelete**.
8. Use the content repository service to either delete (if hard delete is "on") or move the content reference to the wastebasket.

9. Set a message in **ViewData** to tell the visitor what happened.

Your controller should look something like the following code:

```
using AlloyDemo.Models.Pages;
using AlloyDemo.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using EPiServer.Security;
using System.Web.Mvc;

namespace AlloyDemo.Controllers
{
    public class DeleteContentPageController : PageControllerBase<DeleteContentPage>
    {
        private readonly IContentRepository repo = null;

        public DeleteContentPageController(IContentRepository repo)
        {
            this.repo = repo;
        }

        public ActionResult Index(DeleteContentPage currentPage)
        {
            var viewmodel = PageViewModel.Create(currentPage);
            return View(viewmodel);
        }

        [HttpPost]
        public ActionResult Delete(DeleteContentPage currentPage,
            ContentReference contentReference, string hardDelete)
        {
            string name = repo.Get<IContent>(contentReference).Name;

            if (hardDelete == "on")
            {
                repo.Delete(contentReference, forceDelete: true,
                    access: AccessLevel.NoAccess);

                ViewData["message"] = $"'{name}' was deleted permanently.";
            }
            else
            {
                repo.Move(contentReference, destination: ContentReference.WasteBasket,
                    requiredSourceAccess: AccessLevel.NoAccess,
                    requiredDestinationAccess: AccessLevel.NoAccess);

                ViewData["message"] = $"'{name}' was moved to trash.";
            }
            var viewmodel = PageViewModel.Create(currentPage);
            return View("Index", viewmodel);
        }
    }
}
```

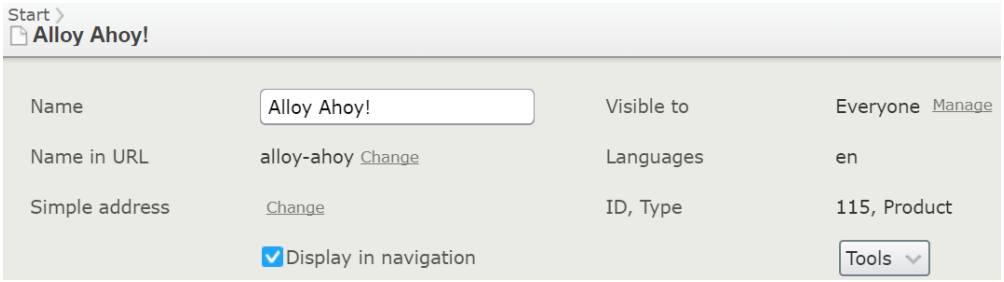
10. In **AlloyDemo**, right-click **Views**, and add a new folder named **DeleteContentPage**.
11. Right-click **DeleteContentPage**, and choose **Add** | **New Item**..., or press *Ctrl* + *Shift* + *A*.
12. In **Add New Item - AlloyDemo**, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Partial View (MVC Razor)**, enter **Index.cshtml** for the Name, and click **Add**.
13. Modify the view, as shown in the following markup, and note the following:
    a. If a message is passed using **ViewData**, it is shown to the visitor.

b. A form that allows a visitor to enter a content reference, select a check box for hard delete, and click a **Delete** button.

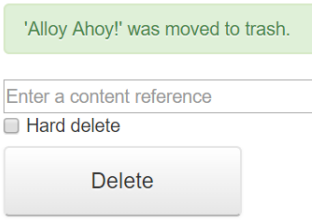
```
@using AlloyDemo.Models.Pages
@model PageViewModel<DeleteContentPage>

@if(ViewData["message"] != null)
{
    <div class="alert alert-success">
        @ViewData["message"]
    </div>
}
<div>
    <form action="@Url.ContentUrl(Model.CurrentPage.ContentLink)delete" method="post">
        <input name="contentReference" placeholder="Enter a content reference" />
        <label for="hardDelete"><input name="hardDelete" id="hardDelete"
type="checkbox" /> Hard delete</label>
        <input type="submit" value="Delete" />
    </form>
</div>
```

Test the website delete content functionality

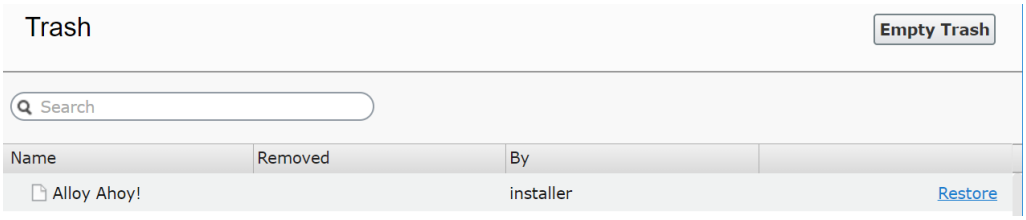
1. Start the **AlloyDemo** website, and log in as **Admin**.
2. Create a **Delete Content** page named **Deleter** under the **Start** page.
3. **Publish** the page.
4. Create a **Standard** page named **Alloy Ahoy!** under the **Start** page.
5. **Publish** the page, and make a note of its ID, for example 115, as shown in the following screenshot:



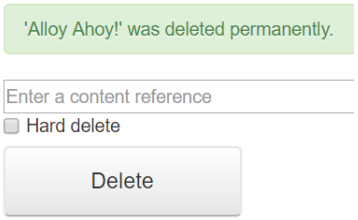
6. Switch to **Live** view, and log out so that you are anonymous.
7. Navigate to the **Deleter** page.
8. Enter the ID of the Alloy Ahoy! page, click **Delete**, and note the message, as shown in the following screenshot:



9. Log in as **Admin**, and view the **Trash**, as shown in the following screenshot:



10. **Restore** the Alloy Ahoy! page.
11. Switch to **Live** view, and log out so that you are anonymous.
12. Navigate to the **Deleter** page.
13. Enter the ID of the Alloy Ahoy! page, select the **Hard delete** check box, click **Delete**, and note the message, as shown in the following screenshot:



14. Log in as Admin, and confirm that the page is not in the Trash, or the Pages tree, and has been permanently deleted.

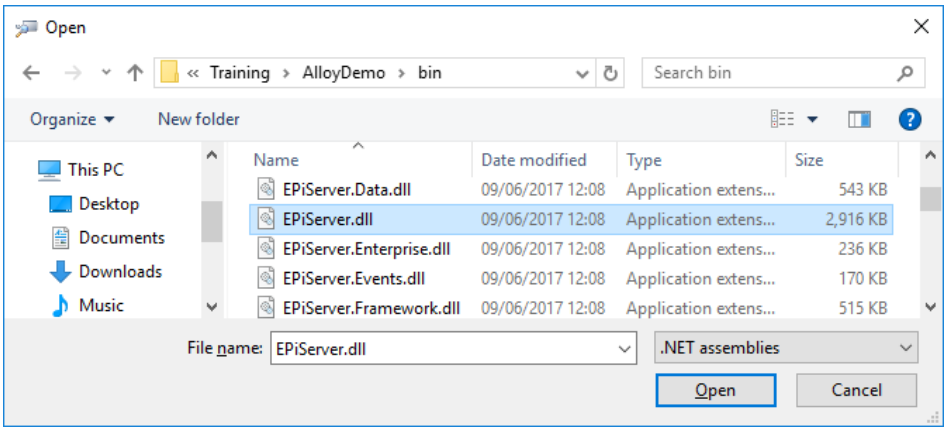
Exercise F6 – Optional: Learning from Episerver's assemblies

In this exercise, you will use ILSpy to decompile and view Episerver assembly code to learn from it.

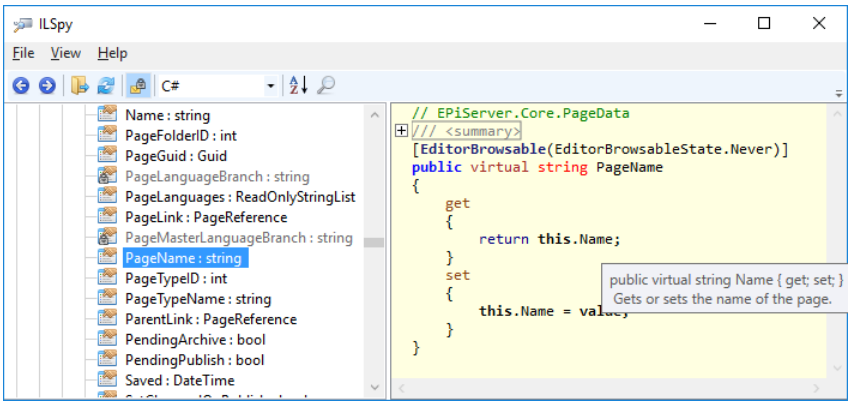
Prerequisites: complete Exercise A1.

Decompiling Episerver assemblies to understand our APIs

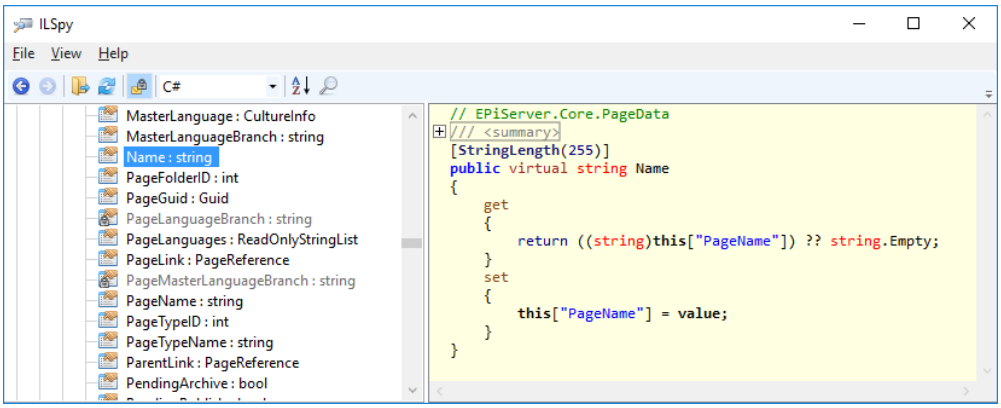
10. Download and install **ILSpy** from: http://ilspy.net/
11. Choose **File** | **Open**, browse to **AlloyDemo** site's **bin** folder, select **EPiServer.dll**, and click **Open**, as shown in the following screenshot:



12. Expand **EPiServer (11.n.n.0)**.
13. Expand **EPiServer.Core**.
14. Expand **PageData**.
15. Select **PageName**, and note that property is decompiled, as shown in the following screenshot:

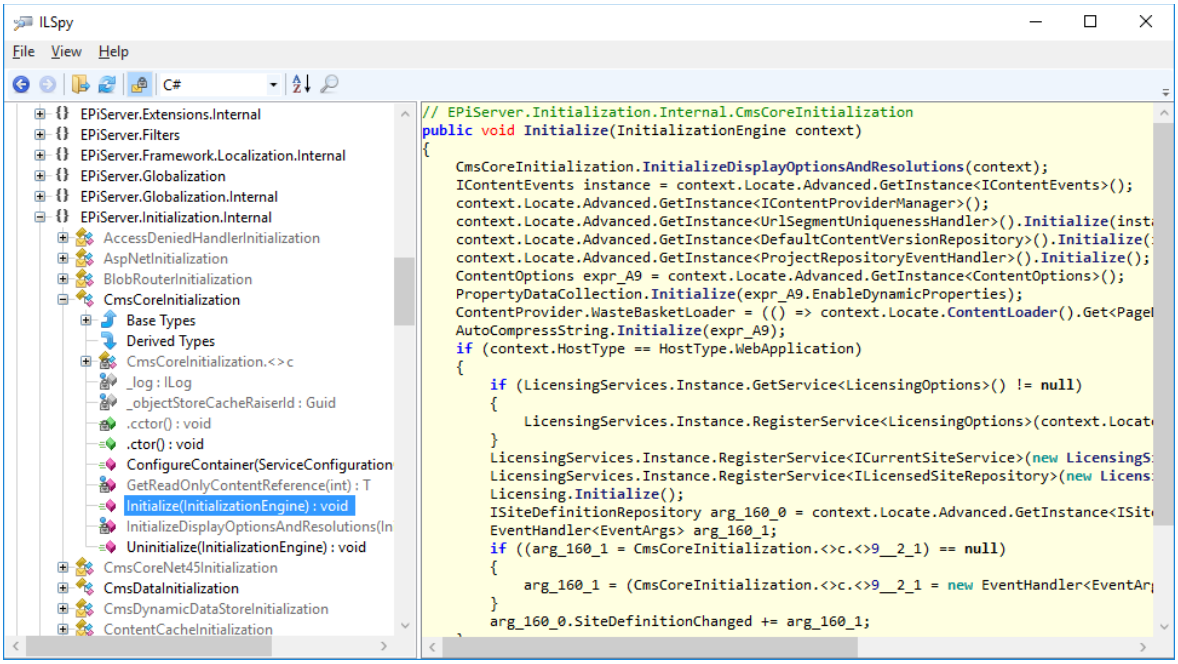


16. In the decompiled source code, click **this.Name**, and note that property is decompiled, as shown in the following screenshot:



Reviewing Episerver initialization

1. In the left list, collapse **EPiServer.Core**.
2. Scroll down the tree view, expand **EPiServer.Initialization.Internal**, expand **CmsCoreInitialization**, and click **Initialize(InitializationEngine) : void**, and note some of the things that Episerver CMS does when it starts, as shown in the following screenshot:



You have now learned how to use tools like ILSpy to see how Episerver's own developer implement functionality. Follow their good practice, for example, in an initialization module use **context.Locate.Advanced.GetInstance<T>** to retrieve services using a dependency resolver.

Module G – Optimizing, Securing, and Deploying

Goal

The overall goal of the exercises in this module is to learn how to prepare an Episerver CMS website before deployment. You will:

1. Control the caching of responses in CDN and browser.
2. Implement logging.
3. Secure the AlloyDemo website.

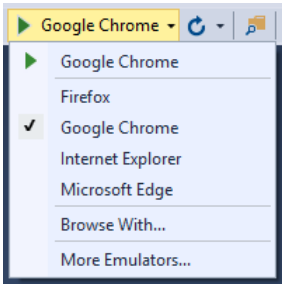
Exercise G1 – Optional: Controlling the caching of responses

In this exercise, you will set cache-control headers and confirm that they were sent in the HTTP response to be read by a CDN and browser.

Prerequisites: complete Exercise A1.

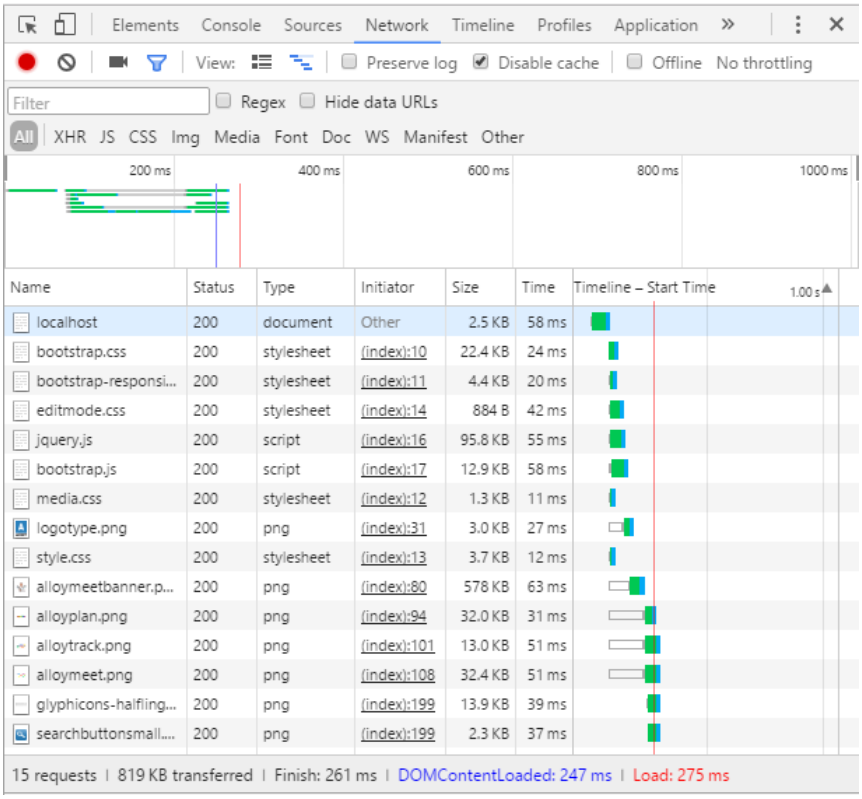
The following instructions assume you are using Google Chrome. Other browsers have similar features.

1. Open the **AlloyDemo** project.
2. In Visual Studio, ensure that **Google Chrome** is set as the default browser for running your website.



3. Start the site by pressing *Ctrl + F5* or navigate to **Debug** | **Start Without Debugging**.
4. In **Google Chrome**, press *F12* to show the developer tools pane.
5. Click the **Network** tab.
6. Check the **Disable cache** check box to ensure requests will not be read from the browser's local cache, as shown in the following screenshot:

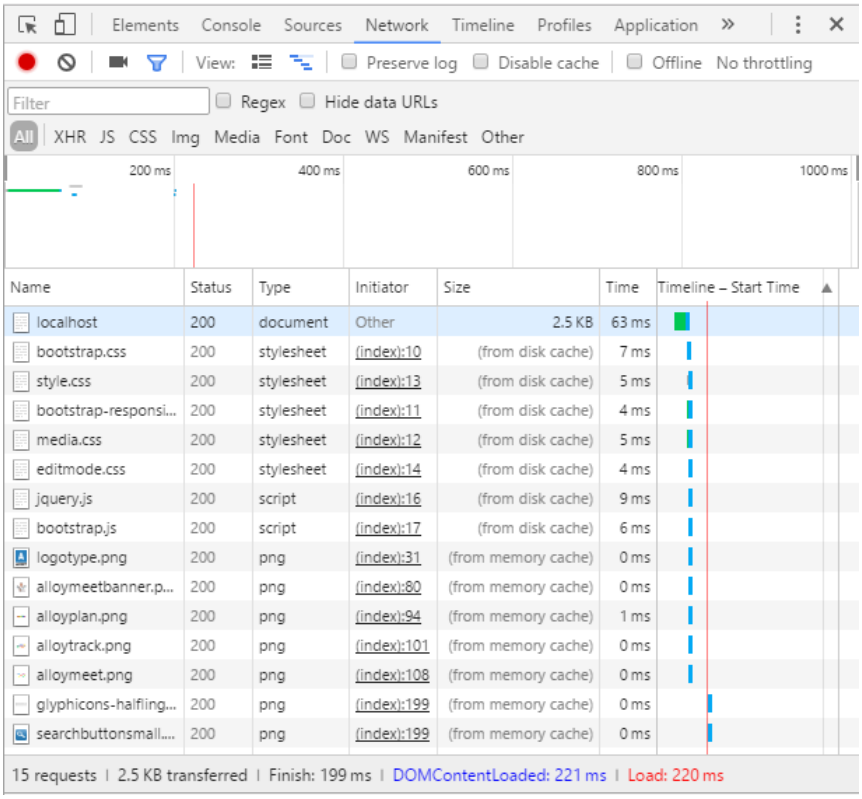
7. Press *F5* to refresh the site's Start page. You will see all the HTTP requests recorded in the developer tools pane, as shown in the following screenshot:



Note that the Status code returned is **200** OK for all requests. Note the load time was 275 ms and 819 KB was transferred over 15 requests for resources.

8. Uncheck the **Disable cache** check box to allow requests to be read from the browser's local cache.

9. Press *F5* to make the same request, and note the differences in the developer tools pane, as shown in the following screenshot:



Note that the Status code returned is **200** OK for all requests. Note the load time was 220 ms and 2.5 KB was transferred over 15 requests for resources. Only the request for **localhost** was sent from the web server, all other requests were handled from either disk cache (for stylesheets and scripts) or memory cache (for images) in the browser.

Viewing the HTTP headers for three types of response

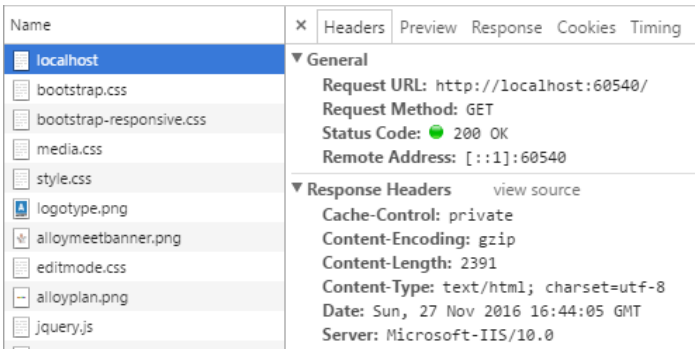
The 15 resources used on the Start page can be divided into three types:

- **Dynamically-generated content**: localhost
- **Static application files**: bootstrap.css, style.css, bootstrap-responsive.css, media.css, editmode.css, jquery.js, bootstrap.js, glyphicons-halflings.png, searchbuttonssmall.png
- **Static asset content**: logotype.png, alloymeetbanner.png, alloyplan.png, alloymeet.png, alloytrack.png

By default, each of the three types is treated differently for caching purposes. This is something you will likely want to configure to optimize for your site.

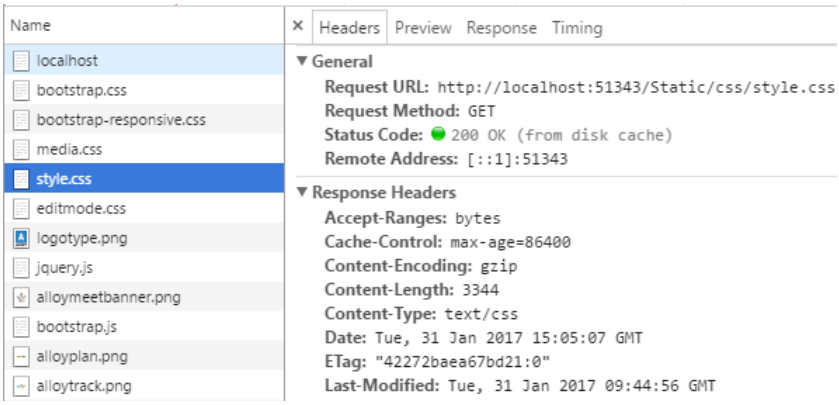
1. In the list of requests, click the first one, named **localhost**, and ensure that the **Headers** tab is selected. This response is dynamically generated at runtime from an MVC view. Note the **Cache-Control** header is set to **private**, meaning that only the browser could store it in its cache, but CDNs

would not cache it, and there is no **Expires** value after the **Date** header, so the browser won't cache it either, as shown in the following screenshot:



Good practice would be to change the default from private to public and consider if and for how long to cache based on the frequency that the dynamically-generated content changes.

2. In the list of requests, click the one named **style.css**, and ensure that the **Headers** tab is selected. This response is static and loaded from the web server's file system. Note that the **Cache-Control** header is set to **max-age=86400**, meaning that the browser or any intermediaries like CDNs can store it for up to one day (the value is in seconds), as shown in the following screenshot:



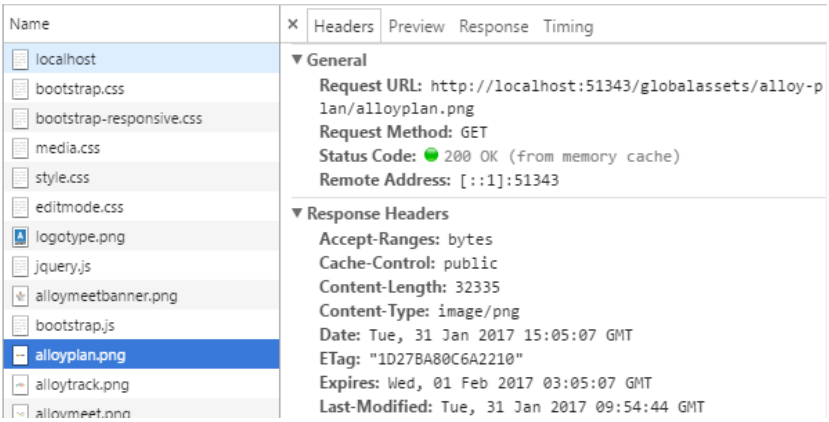
Common **max-age** values are:

- One minute: max-age=60
- One hour: max-age=3600
- One day: max-age=86400
- One week: max-age=604800
- One month: max-age=2628000
- One year: max-age=31536000

Good practice would be to change the default from one day to one year.

3. In the list of requests, click the one named **alloyplan.png**, and ensure that the **Headers** tab is selected. Note the **Cache-Control** header is set to **public**, meaning that the browser and any

intermediaries can store it until it expires. Its **Expires** header is 12 hours after the **Date** header, as shown in the following screenshot:



4. Close the browser.

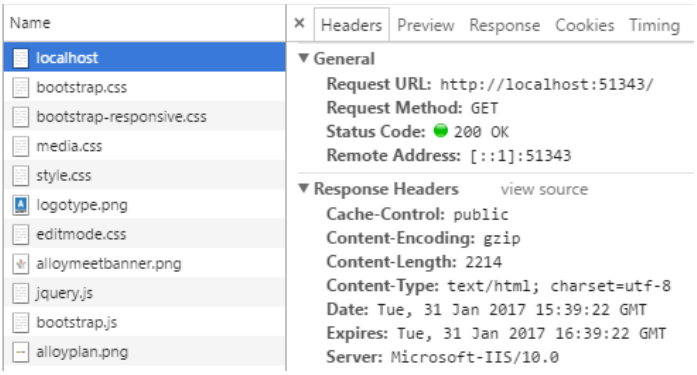
Controlling caching of dynamically-generated content with code

For total control of how dynamically-generated content is cached, use code.

1. Open **~\Controllers\StartPageController.cs**.
2. Modify its **Index** method to add the following three statements *before* returning the view:

```
Response.Cache.SetCacheability(System.Web.HttpCacheability.Public);
Response.Cache.SetExpires(System.DateTime.Now.AddHours(1));
Response.Cache.SetSlidingExpiration(true);
```

3. Start the site.
4. View the recorded network requests in the developer tools pane. Note that the **Cache-Control** header is set to **public**, meaning that both the browser *and* CDNs can cache it, and there is an **Expires** value set one hour after the **Date** controlling how long the resource will be cached, as shown in the following screenshot:



5. Close the browser.
6. To avoid confusion in later exercises, open **~\Controllers\StartPageController.cs**, and comment out the three statements that enabled caching.

Controlling caching of dynamic content with attributes and configuration

A simpler, but less flexible and powerful, alternative to writing code is to use a combination of configuration and attributes.

1. In the **~\Controllers** folder, open **DefaultPageController.cs**.

This controller is used for all page types in Alloy that do not have their own specific controller.

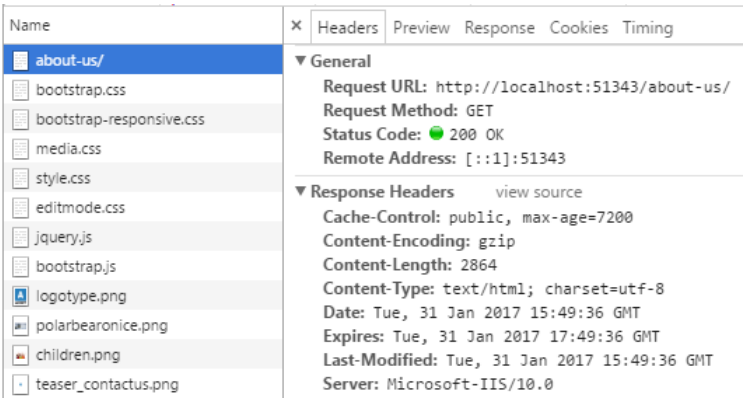
2. Apply the `[ContentOutputCache]` attribute *before* the **Index** method:

```
[EPiServer.Web.Mvc.ContentOutputCache]
public ViewResult Index(SitePageData currentPage)
```

3. Open **~\Web.config**.
4. Find the `<episerver>` section. Inside the `<applicationSettings>` element, add the attribute **httpCacheExpiration** and set its value to a time span of two hours (02:00:00). Note the **httpCacheabillity** attribute value is already **Public**, but this is ignored unless the `[ContentOutputCache]` attribute is applied as an action filter. The defaults set in configuration can be overridden in a `[ContentOutputCache]` attribute if necessary.

```
<episerver>
  <applicationSettings
    httpCacheability="Public"
    httpCacheExpiration="02:00:00"
    ...
```

5. Start the site.
6. View the recorded network requests in the developer tools pane for the **About us** page and note the HTTP response headers have been affected, for example, **Expires** is two hours after **Date**, and **max-age** is **7200** seconds (2 hours), as shown in the following screenshot:



7. Close the browser.
8. To avoid confusion in later exercises, open **~\Controllers\DefaultPageController.cs**, and comment out the `[ContentOutputCache]` that enabled caching.

Controlling caching of static content

1. Open **~\Web.config**.
2. At the top of the file, add the following `<section>` element inside the `<configSections>` element:

```
    <section name="staticFile" allowLocation="true" type=
"EPiServer.Framework.Configuration.StaticFileSection, EPiServer.Framework.AspNet" />
```

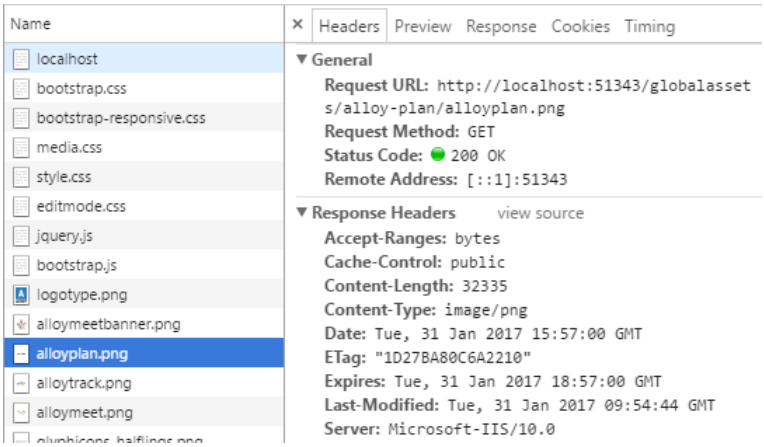
> **Breaking change in CMS 11**
> Assembly name has changed to EPiServer.Framework.AspNet. For CMS 10, use EPiServer.Framework.

3. Add the following element after the `</configSections>` element. It will set the expiration to 3 hours of any static media assets managed by the CMS, such as images, videos, and so on.

```
<staticFile expirationTime="03:00:00" />
```

> In this exercise, you set the expiration to three hours, but a year would be better practice in real life.

4. Start the site.
5. Show the developer tools by pressing *F12*.
6. Check the **Disable cache** check box, and then press *F5* to refresh the previously cached for 12 hours static content.
7. Uncheck the **Disable cache** check box, and then press *F5*.
8. View the recorded network requests in the developer tools pane for the **alloyplan.png** image on the Start page and note the HTTP response headers have been affected. The difference between the **Date** and **Expires** will now be 3 hours instead of 12 hours, as shown in the following screenshot:



9. Close the browser.

Controlling caching of static application files

1. Find the `<system.webServer>` element.
2. Inside it, find the `<staticContent>` element that contains a `<clientCache>` element with **cacheControlMode** set to **UseMaxAge** and a **cacheControlMaxAge** set to a time span of 1 day, as shown in the following configuration:

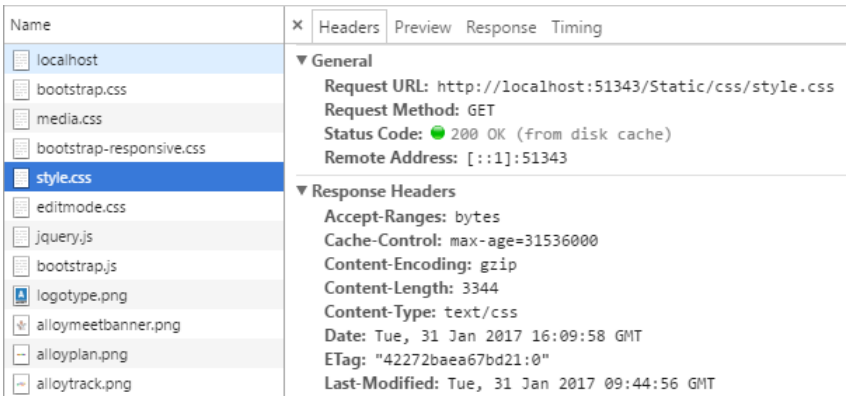
```
<staticContent>
  <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="1.00:00:00" />
```

3. Change the number of days to **365**. This will set the expiration to one year for any static application files that are part of your site, such as images, JavaScript files, CSS stylesheets, and so on.

```
<staticContent>
  <clientCache cacheControlMode="UseMaxAge"
               cacheControlMaxAge="365.00:00:00" />
```

When you set a "far future" max-age value for static application resources, you should include a version number or date as part of the filename, e.g. change **style.css** to **style-2017-01-31.css** or **jquery.js** to **jquery-3.1.1.js**.

4. Start the site.
5. Show the developer tools by pressing *F12*.
6. Check the **Disable cache** check box, and then press *F5* to refresh the previously cached for 1 day static application files.
7. Uncheck the **Disable cache** check box, and then press *F5*.
8. View the recorded network requests in the developer tools pane for the **style.css** file on the Start page and check that the HTTP response headers have been affected. The **max-age** is now **31536000** seconds (one year) instead of 86400 seconds (one day), as shown in the following screenshot:



9. Close the browser.

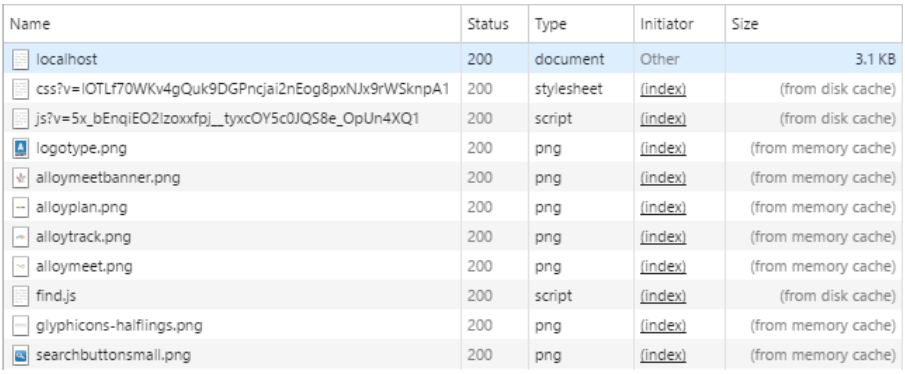
Bundling, minimization, and cache busting

In the screenshot above, note that the CSS files needed for the page are five separate HTTP requests.

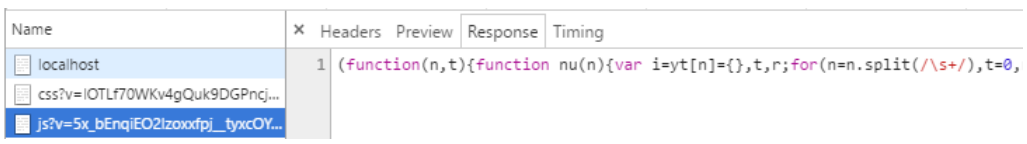
1. Find the `<system.web>` element.
2. Inside it, find the `<compilation>` element that contains a **debug** attribute set to **true**, as shown in the following configuration:

```
<system.web>
  <httpRuntime targetFramework="4.6.1" requestValidationMode="2.0" />
  <compilation debug="true" targetFramework="4.6.1"
               optimizeCompilations="true" />
```

3. Change the **debug** attribute to **false**.
4. Start the site.
5. Show the developer tools by pressing *F12*.
6. Press *F5*.
7. View the recorded network requests in the developer tools pane and note the CSS files (and JavaScript files) have been bundled into single requests, as shown in the following screenshot:



8. Click the JavaScript bundle, **js?v=...**, and then click the **Response** tab, and note that as well as bundling multiple requests for JavaScript together, the files have been minimized, as shown in the following screenshot:



If any of the files in a bundle changes, ASP.NET automatically changes the hash value (v=…) so that the cached bundle is “busted” and the change downloaded. You could implement a similar “cache busting” technique in combination with “far future” cache control headers for static content.

9. Close the browser.
10. Open **~\Business\Initialization\BundleConfig.cs**, and note the **RegisterBundles** method that determines which static application files are included in the two bundles, as shown in the following code:

```
bundles.Add(new ScriptBundle("~/bundles/js").Include(
            "~/Static/js/jquery.js",
            "~/Static/js/bootstrap.js"));

bundles.Add(new StyleBundle("~/bundles/css")
    .Include("~/Static/css/bootstrap.css", new CssRewriteUrlTransform())
    .Include("~/Static/css/bootstrap-responsive.css")
    .Include("~/Static/css/media.css")
    .Include("~/Static/css/style.css", new CssRewriteUrlTransform())
    .Include("~/Static/css/editmode.css"));
```

Creating a page type and view model for the object cache

You will create a page type with a page template that will show the contents of the Episerver object cache.

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

1. In **~\Models\Pages**, add an **Episerver** | **Page Type** named **ObjectCachePage.cs**.
2. Modify the contents, as shown in the following code:

```
namespace AlloyDemo.Models.Pages
{
    [SiteContentType(DisplayName = "Object Cache",
        GroupName = Global.GroupNames.Specialized,
        Description = "View the contents of the object cache.")]
    [SiteImageUrl]
    [AvailableContentTypes(IncludeOn = new[] { typeof(StartPage) })]
    public class ObjectCachePage : SitePageData
    {
    }
}
```

3. In **~\Models\ViewModels**, add a class named **ObjectCachePageViewModel.cs**.
4. Modify the contents, as shown in the following code:

```
using AlloyDemo.Models.Pages;
using System.Collections;
using System.Collections.Generic;

namespace AlloyDemo.Models.ViewModels
{
    public class ObjectCachePageViewModel
        : PageViewModel<ObjectCachePage>
    {
        public IEnumerable<DictionaryEntry> CachedItems { get; set; }

        public string FilteredBy { get; set; }

        public ObjectCachePageViewModel(
            ObjectCachePage currentPage) : base(currentPage)
```

```
            {
            }
        }
    }
```

Creating a page template for the object cache

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 **~\Controllers**, add an **Episerver | Page Controller (MVC)** named **ObjectCachePageController.cs**.
2. Modify the contents, as shown in the following code:

```
using AlloyDemo.Models.Pages;
using AlloyDemo.Models.ViewModels;
using EPiServer;
using EPiServer.Core;
using EPiServer.Web.Mvc;
using System.Collections;
using System.Linq;
using System.Web.Mvc;

namespace AlloyDemo.Controllers
{
    public class ObjectCachePageController
        : PageControllerBase<ObjectCachePage>
    {
        public ActionResult Index(ObjectCachePage currentPage, string filterBy)
        {
            var viewmodel = new ObjectCachePageViewModel(currentPage);

            var cachedEntries = HttpContext.Cache.Cast<DictionaryEntry>();

            switch (filterBy)
            {
                case "pages":
                    viewmodel.CachedItems = cachedEntries
                        .Where(item => item.Value is PageData);
                    break;
                case "content":
                    viewmodel.CachedItems = cachedEntries
                        .Where(item => item.Value is IContent);
                    break;
                default:
                    viewmodel.CachedItems = cachedEntries;
                    break;
            }

            viewmodel.FilteredBy = filterBy;

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

3. In **~\Views**, create a new folder named **ObjectCachePage**.
4. In **~\Views\ObjectCachePage**, create a file named **Index.cshtml**.
5. Modify its contents, as shown in the following markup:

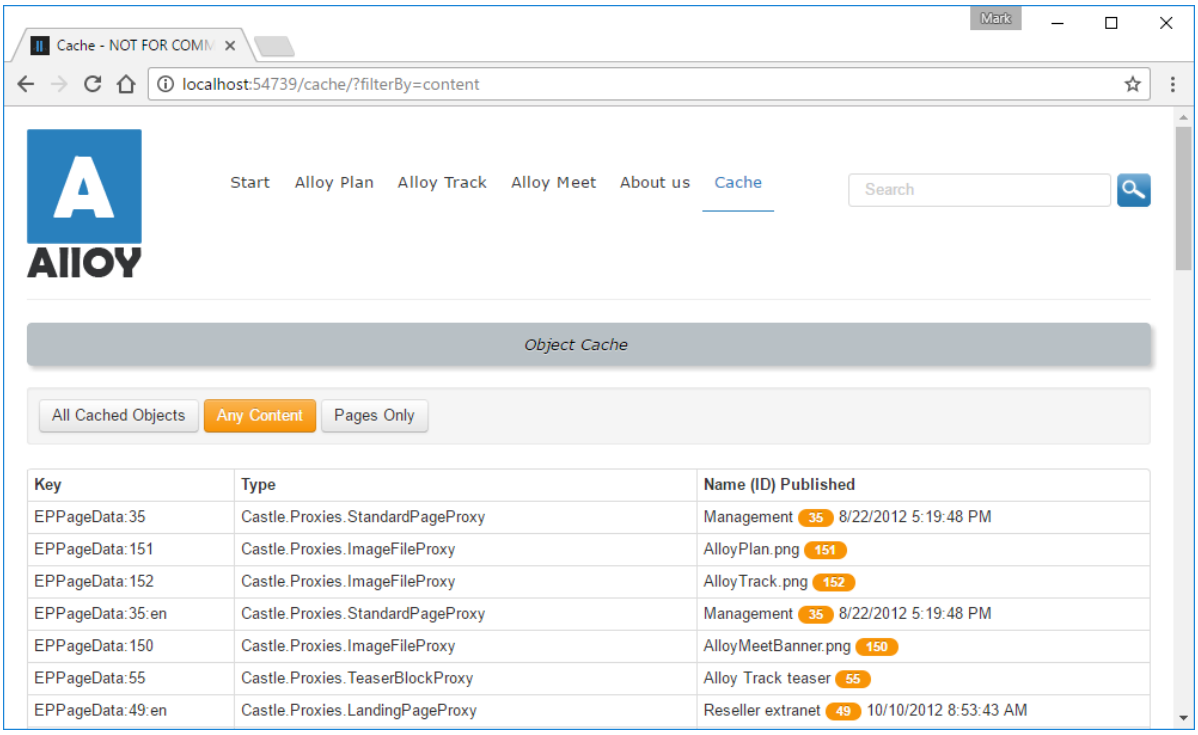
```
@using System.Collections
@using EPiServer.Core
```

```
@using EPiServer.Web.Mvc.Html
@model AlloyDemo.Models.ViewModels.ObjectCachePageViewModel
<div>
    <div class="alert alert-info">Object Cache</div>
    <div class="well well-small">
        @Html.ContentLink("All Cached Objects", Model.CurrentPage.ContentLink, null,
           htmlAttributes: new { @class = string.IsNullOrWhiteSpace(Model.FilteredBy)
? "btn btn-warning" : "btn" })

        @Html.ContentLink("Any Content", Model.CurrentPage.ContentLink,
           routeValues: new { filterBy = "content" },
           htmlAttributes: new { @class = Model.FilteredBy == "content"
                                ? "btn btn-warning" : "btn" })


        @Html.ContentLink("Pages Only", Model.CurrentPage.ContentLink,
           routeValues: new { filterBy = "pages" },
           htmlAttributes: new { @class = Model.FilteredBy == "pages"
                                ? "btn btn-warning" : "btn" })
    </div>
    <table class="table table-condensed table-bordered table-condensed">
        <tr>
            <th>Key</th>
            <th>Type</th>
            <th>@(string.IsNullOrWhiteSpace(Model.FilteredBy)
                        ? "Value" : "Name (ID) Published")</th>
        </tr>
        @foreach (DictionaryEntry item in Model.CachedItems)
        {
            <tr>
                <td>@item.Key</td>
                <td>@item.Value.GetType()</td>
                <td>
                    @if (item.Value is IContent)
                    {
                        @((item.Value as IContent).Name)
                        <span class="badge badge-warning">
                            @((item.Value as IContent).ContentLink.ID)</span>
                    }
                    @if (item.Value is PageData)
                    {
                        @((item.Value as PageData).StartPublish)
                    }
                </td>
            </tr>
        }
    </table>
</div>
```

Creating an object cache page in the page tree

1. Start the site, and log in as **Admin**.
2. Underneath the **Start** page, add an **Object Cache** page named **Cache**.
3. **Publish** the page.
4. View the site as a visitor.

5. Click **Any Content**, as shown in the following screenshot, and note the key values that Episerver uses for content, for example, the **Alloy Track teaser** block has a key of **EPPageData:55**:



Exercise G2 – Optional: Implementing logging

In this exercise, you will implement logging an invalid property setting by using Log4net.

Prerequisites: complete Exercise A1.

Reviewing the default logging configuration

1. Open the **Training** solution with the **AlloyDemo** project.
2. Open the **~/EPiServerLog.config** file.
3. Note the configuration of the first **<appender>** element:
   - It uses the **RollingFileAppender** type with a **rollingStyle** or **Date** and a pattern of **yyyyMMdd** so that each day a new log file will be created with the date as part of the filename.
   - The files will be stored in the **App_Data** folder.
   - Each entry will include the date, thread, level, and message.

```
<appender name="errorFileLogAppender" type="log4net.Appender.RollingFileAppender" >
    <!-- Consider moving the log files to a location outside the web application -->
    <file value="App_Data\EPiServerErrors.log" />
    <encoding value="utf-8" />
    <staticLogFileName value="true"/>
    <datePattern value=".yyyyMMdd.'log'" />
    <rollingStyle value="Date" />
    <threshold value="Error" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <appendToFile value="true" />
    <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %level %logger: %message%n" />
    </layout>
</appender>
```

4. Note the enabling of logging at **Error** level.

```
<root>
    <!-- setting this value to All, Debug or Info will affect performance.-->
    <level value="Error" />
    <!--Enabled file logging-->
    <appender-ref ref="errorFileLogAppender" />
```

Writing to the logger

1. Open the **~/Controllers/StartPageController.cs** file.
2. Import namespaces for logging, by adding the following statements to the top of the file:

```
using EPiServer.Core;
using EPiServer.Logging;
```

3. Inside the class, declare a field to store a reference to the registered **ILogger** service, as shown in the following code:

```
private readonly ILogger logger = LogManager.GetLogger();
```

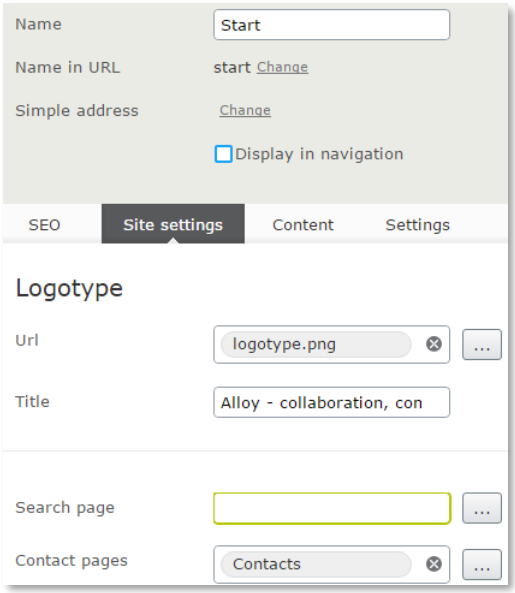
4. Inside the **Index** method, before creating the page view model, add a statement to write to the log if a search page link has not been set for the StartPage, as shown in the following code:

```
if (PageReference.IsNullOrEmpty(currentPage.SearchPageLink))
{
    logger.Error("No 'Search page' is specified in 'Site settings'.");
}
```

5. In the **Solution Explorer**, click **Show All Files**, and then delete the **~/App_Data/EPiServerErrors.log** file to clear any existing logs for today.

Testing the logging

1. Start the site, and log in as **Admin**.
2. Edit the **Start** page, and switch to **All Properties** view.
3. Click the **Site settings** tab.
4. Clear the **Search page** property, as shown in the following screenshot:

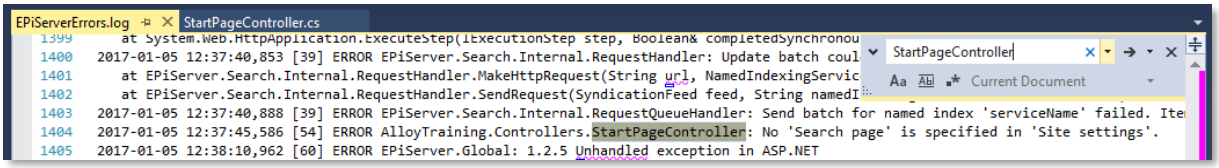


5. **Publish** the changes to the Start page.
6. View the Start page as a visitor.
7. Close the browser.

Reviewing the log

1. In **Solution Explorer**, refresh the **~\App_Data** folder, and then open the **EPiServerErrors.log** file.
2. If the file contains lots of log entries, press *Ctrl* + *F* to find, type **StartPageController** and press ENTER to find the log entry.
3. Note the log entry, as shown in the following screenshot:

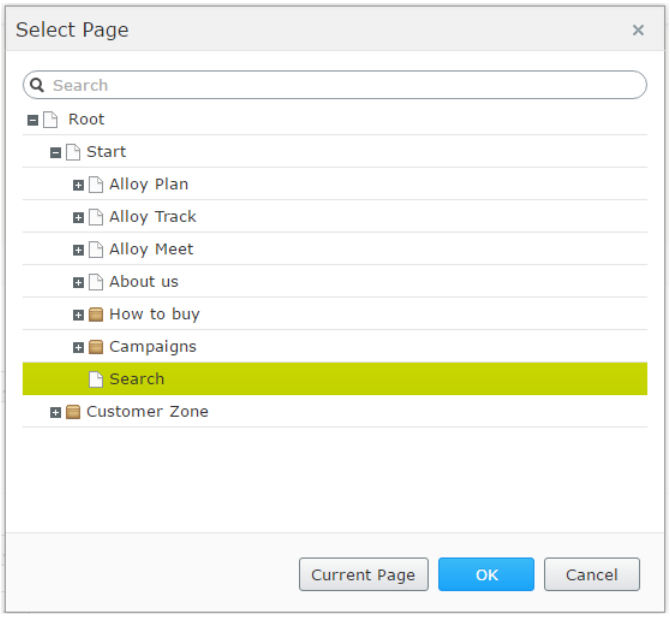
```
2017-01-05 12:37:45,586 [54] ERROR
AlloyDemo.Controllers.StartPageController: No 'Search page' is specified in
'Site settings'.
```



Restore the Search page

1. Start the site, and log in as **Admin**.
2. Edit the **Start** page, and switch to **All Properties** view.

3. Click the **Site settings** tab.
4. Set the **Search page** property to the Search page, as shown in the following screenshot:



5. **Publish** the changes to the Start page.
6. Close the browser.

You've now added basic logging to one page type used by the site.

Exercise G3 – Optional: Securing an Episerver site

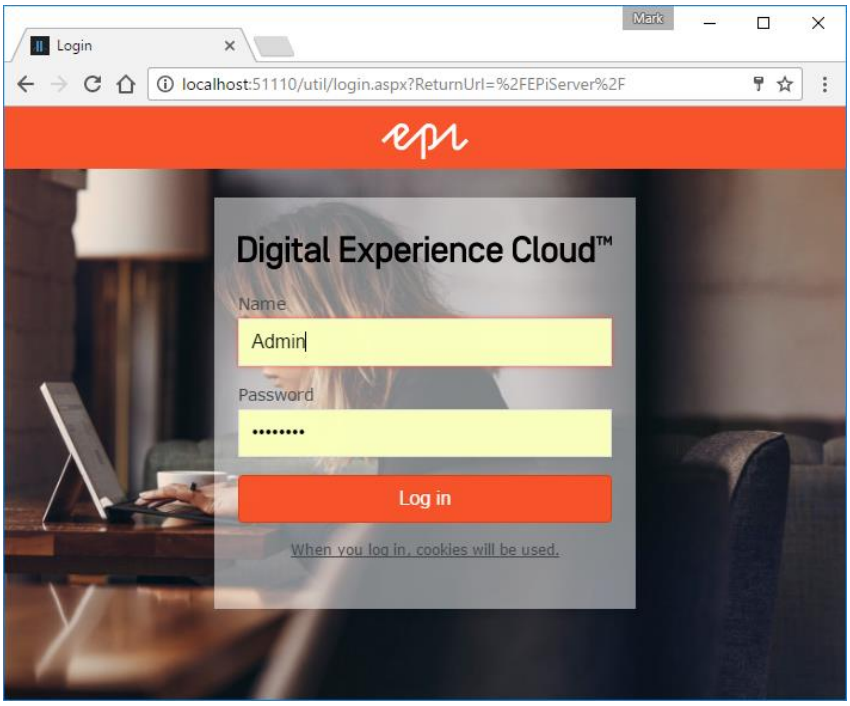
In this exercise, you will change the Episerver URL path to implement security through obscurity.

Prerequisites: complete Exercise A1.

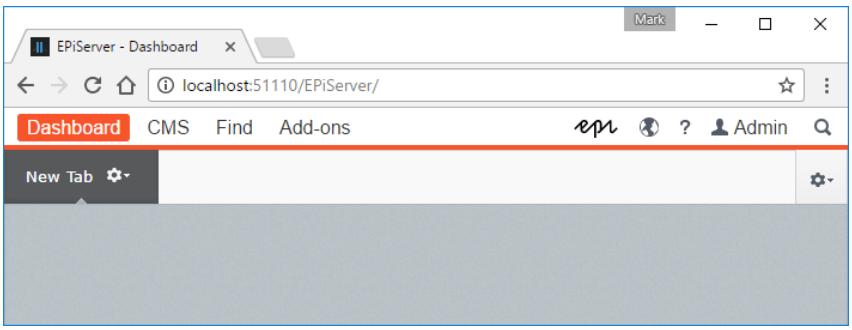
Reviewing the default Episerver URL path

1. Open the **AlloyDemo** project.
2. Start the site.
3. In the browser's address bar, enter: **`EPiServer/`** at the end of the URL, as shown in the following screenshot:

4. You will be redirected to the log in page, as shown in the following screenshot:



5. Enter a name of **Admin** and a password of **Pa$$w0rd**, click **Log in**, and you will see the **Global** menu and **Dashboard**, as shown in the following screenshot:



6. Close the browser.

You will now modify the site's configuration to change the URL path.

Changing the Episerver URL path

1. Open **~/Web.config**.
2. Find the `<applicationSettings>` element, and modify the **uiUrl** attribute to **~/Secret/CMS**, as shown in the following partial configuration:

```
<episerver>
  <applicationSettings
    ...
    uiSafeHtmlTags="b,i,u,br,em,strong,p,a,img,ol,ul,li"
    httpCacheability="Public"
    uiEditorCssPaths="~/Static/css/Editor.css"
    uiShowGlobalizationUserInterface="true"
    uiMaxVersions="20"
    uiUrl="~/Secret/CMS/" />
```

3. Find the `<location path="EPiServer">` element, and modify the **path** attribute to **Secret**, as shown in the following configuration:

```
<location path="Secret">
```

4. Find the `<location path="EPiServer/CMS/admin">` element, and modify the **path** attribute to **Secret/CMS/Admin**, as shown in the following partial configuration:

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

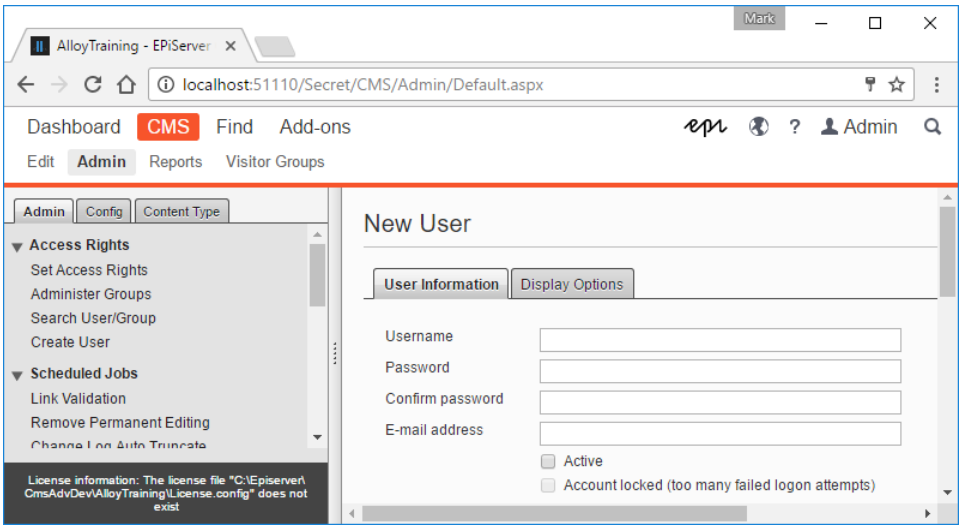
5. Use Visual Studio's Find feature (press *Ctrl* + *F*) to search for **~/EPiServer**.
6. Modify the two **~/EPiServer** entries to **~/Secret**, as shown in the following configuration:

```
<virtualPathProviders>
  <clear />
  <add name="ProtectedModules"
       virtualPath="~/Secret/"
       physicalPath="Modules\_Protected"
       type="EPiServer.Web.Hosting.VirtualPathNonUnifiedProvider,
             EPiServer.Framework.AspNet" />

<episerver.shell>
  <publicModules rootPath="~/modules/" autoDiscovery="Modules" />
  <protectedModules rootPath="~/Secret/">
```

Breaking change in CMS 11
Assembly name has changed to EPiServer.Framework.AspNet. For CMS 10, use EPiServer.Framework.

7. Start the site, and log in as **Admin**.

8. All the menus in the **Global** menu should now use **Secret** as the URL path, as shown in the following screenshot:



This is the end of the exercises book.

Summary of Attributes in Episerver

Attributes for Content Types

Name	Parameters	Description
ContentType	DisplayName Description GroupName Order GUID AvailableInEditMode	Controls the registration of a content type in the CMS database.
Access	Roles Users ...others	Controls who can create an instance of this type.
ImageUrl	Path	Sets a 120 x 90 preview icon.
AvailableContentTypes	Availability Include IncludeOn Exclude ExcludeOn	Controls which content types can be this content type's parent and children.
MediaDescriptor	Extensions ExtensionString	Controls which file extensions are associated with this media type.

Attributes for Properties

Name	Parameters	Description
Display	Name Description GroupName Order	Controls the registration of the property in the CMS database.
AllowedTypes	AllowedTypes RestrictedTypes ...others	Controls which content types can be referenced by the property.
CultureSpecific		Allows the property to have language branches.
Seachable		Includes the property in the search index.
Editable	true	
Ignore		Prevents property from being stored in CMS database.
ScaffoldColumn	false	Hides the property from Editors.
UIHint	uiHint presentationLayer	For visitors: selects a DisplayTemplate. For editors: selects a custom property editor.
SelectOne	SelectionFactory	Editors can select one value from a drop-down listbox.
SelectMany	SelectionFactory	Editors can select many values from a list of check boxes.
Required, Range, StringLength, RegularExpression, and other ValidationAttribute-derived attributes	...various	Validation rules are applied when saving and publishing property.

Attributes for Controllers

Name	Parameters	Description
TemplateDescriptor	Name Description Inherited Default Tags, TagString AvailableWithoutTag ModelType Path TemplateTypeCategory	Controls the registration of a content template in the CMS database.
ContentOutputCache	Duration (seconds) Location ...others	Controls how long and where the response is cached. Can also be applied to action method(s).

Attributes for Groups and Tabs

Name	Parameters	Description
GroupDefinitions	n/a	Apply to a static class with string constants to define Tabs in code.
Display	Name Order	Controls the registration of the tab names in the CMS database.
RequiredAccess	AccessLevel	Controls what access rights an Editor must have to see a tab and its properties.

Attributes for Initialization Modules

Name	Parameters	Description
InitializableModule	UninitializeOnShutdown	Apply to a class to register it as an initialization module. It must implement `IInitializableModule` or `IConfigurableModule`.
ModuleDependency	One or more types.	Controls the order in which initialization modules execute by defining dependencies.

Attributes for Extensions

Name	Parameters	Description
EditorDescriptorRegistration	TargetType UIHint EditorDescriptorBehavior	Registers a class as a custom editor for a data type.
UIDescriptorRegistration	n/a	Registers a class to customize UI elements for a content type.
Component	Title Description PlugInAreas SortOrder AllowedRoles Categories	Registers a class as a gadget for an Editor or Admin to add to their Dashboard or Edit view Navigation and Assets panes.
GuiPlugIn	Area Category SortIndex DisplayName Description DefaultEnabled RequiredAccess	Registers a class as a plug-in. Category is only used by Visitor Group Criteria
ScheduledPlugIn	DisplayName Description InitialTime IntervalLength IntervalType Restartable SortIndex GUID	Registers a class as a scheduled job. It can optionally inherit from ScheduleJobBase or have a static method named Execute that returns a string.