



Umbraco Training

Umbraco Fundamentals

Your Name: _____

Date Attended: _____

Trainer: _____

Agenda

We have a great day ahead of us, here is a list of the things we will be working on today:

Solution Setup

- Getting started
- Umbraco Cloud

Basic building blocks

- Defining content with document types
- Templating
- Navigation and lists with Razor

Feature iterations

- Enhancing the site with the image cropper
- Building a news section
- A macro-powered gallery

Strategies for nested content

- Patterns for complex content structures

Rich content pages

- Grid editor
- Nested content
- Forms

Appendix: Razor introduction

Training site setup

Your course site is a standard Umbraco Cloud site, giving you an easy way to get started with Umbraco, without worrying about setting up a local environment. Your trainer will provide you with access to the site.

Environment

- Umbraco 7
- Umbraco Cloud

Bundled items:

- Html, css and javascript files
- Razor snippets



Umbraco Cloud

For this course, we use Umbraco Cloud - and so you'll need to know a little about how this service actually works.

Using Umbraco Cloud

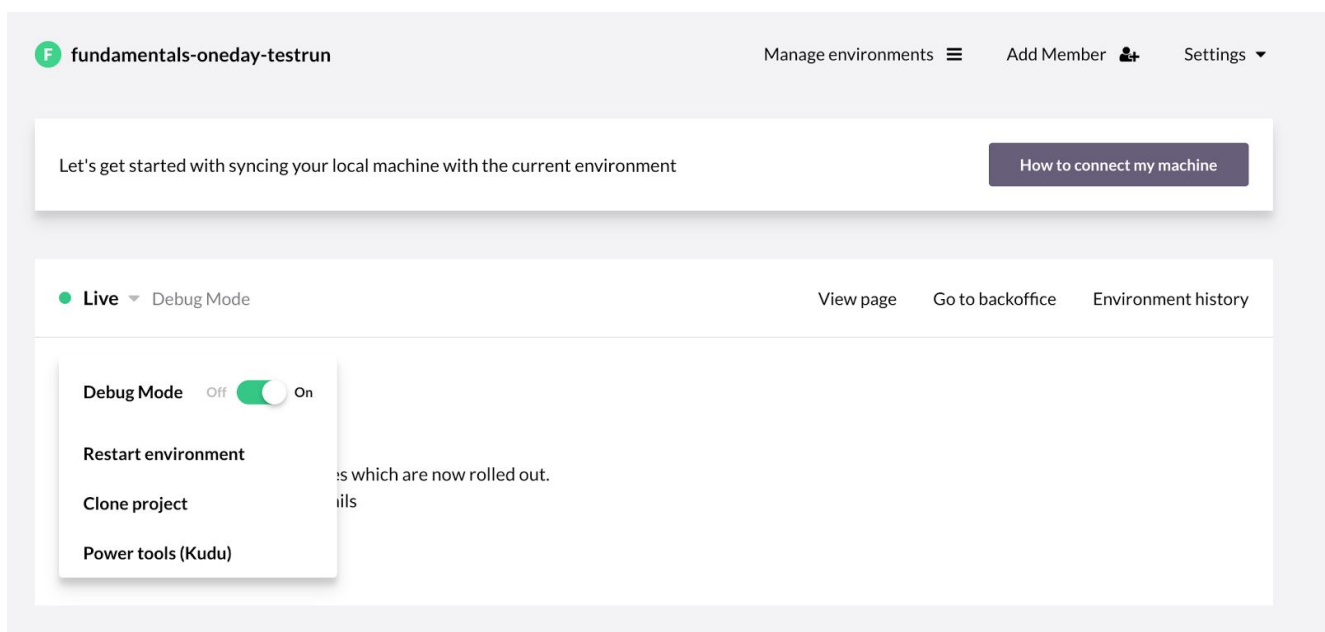
As a course attendee, you will already have access to an Umbraco Cloud site, a welcome email should already have found its way into your inbox - follow the instructions in the email to setup your account on Umbraco Cloud - Cloud access is free during the course.

Getting access to your site:

1. Go to Umbraco.io
2. Login with the credentials provided
3. Navigate to your assigned site
4. Click “Go to backoffice”
 - a. You will now see the standard Umbraco Login screen
 - b. Enter the **same** credentials once more to gain access to Umbraco.io

Getting the site in debug mode

As you develop and work with your site, you will need to get some feedback on the mistakes you make, to get detailed information from the site, you need to configure it to run in debug mode, do this by clicking the arrow next to the site name:





Basic building blocks

For the first part of this course we will setup a basic website, which will have a simple content structure, a number of document types which define what content we can store and how we can edit it, and finally, we will add some templates which handle how the site will look and feel.

Exercise 1 - Content

In this first exercise, we will construct the basic scaffold for our standard pages and construct the editor experience for managing these pages from the backoffice. At the end of this exercise, we will have a basic content structure with a homepage and 3 child pages.

In this exercise, we will:

1. Create document types
2. Setup and configure properties on our document types
3. Configure permissions for our content
4. Create a basic content structure

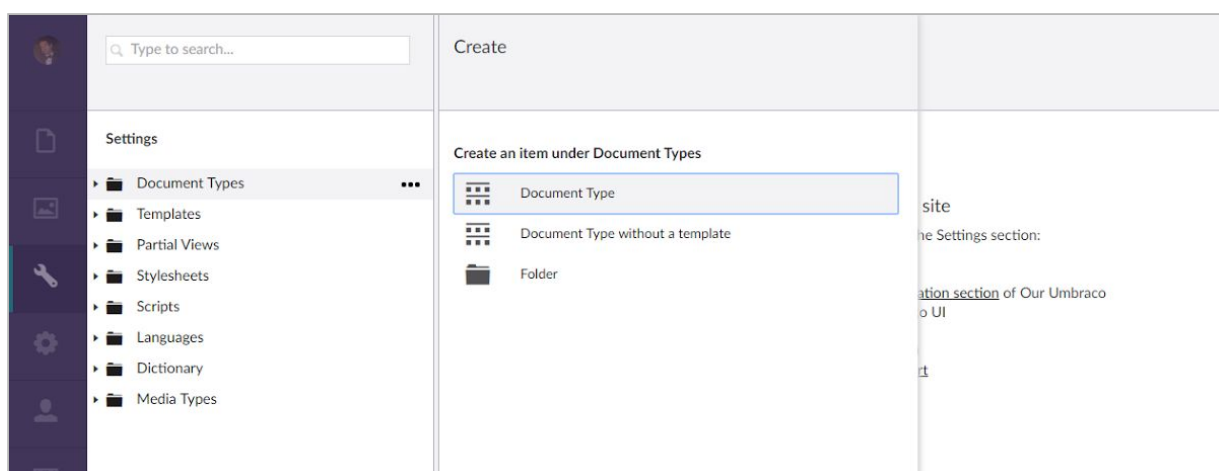
Create a document type

The very first step of any Umbraco site is to define what content our editors can create. We use document types for this definition. For this exercise, we will create 2 document types.

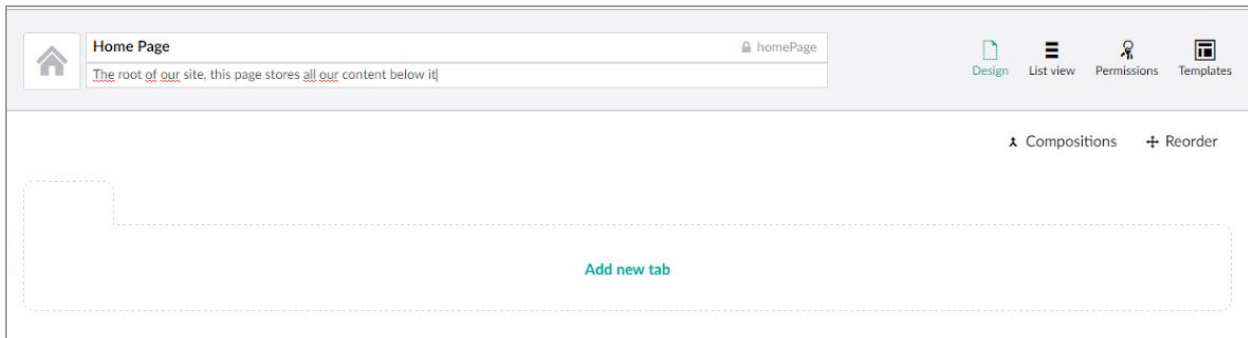


Go to settings: For this part of the exercise, we will use the settings section of the backoffice

Click the options icon **...** on the document types tree and select “Document type”, this opens the document type editor:



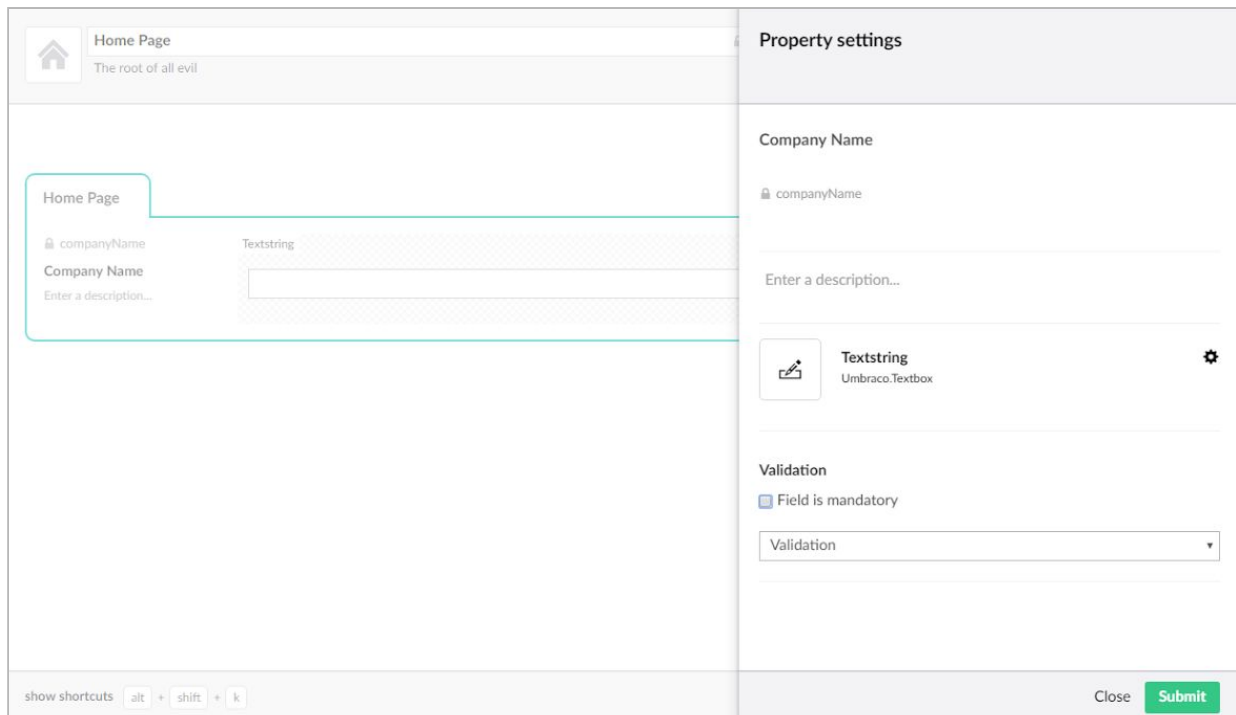
Enter the name “Home Page” and choose an appropriate icon, also ensure this document has a suitable description:



Add document type properties

Add a new property to the Home Page document type:

1. Click add new tab, enter the tab name “Home Page”
2. Click **“Add property”**
 - a. Enter the name **“Company Name”**
 - b. Enter an instruction description such as “Used in the title and footer of the website”
 - c. Click **“Add Editor”**
 - d. Choose the editor **“Textstring”**
 - e. Click “Submit”



Repeat this process 2 more times, for the 2 other properties that should be on the **Home Page** document type, consistent with the specification below:



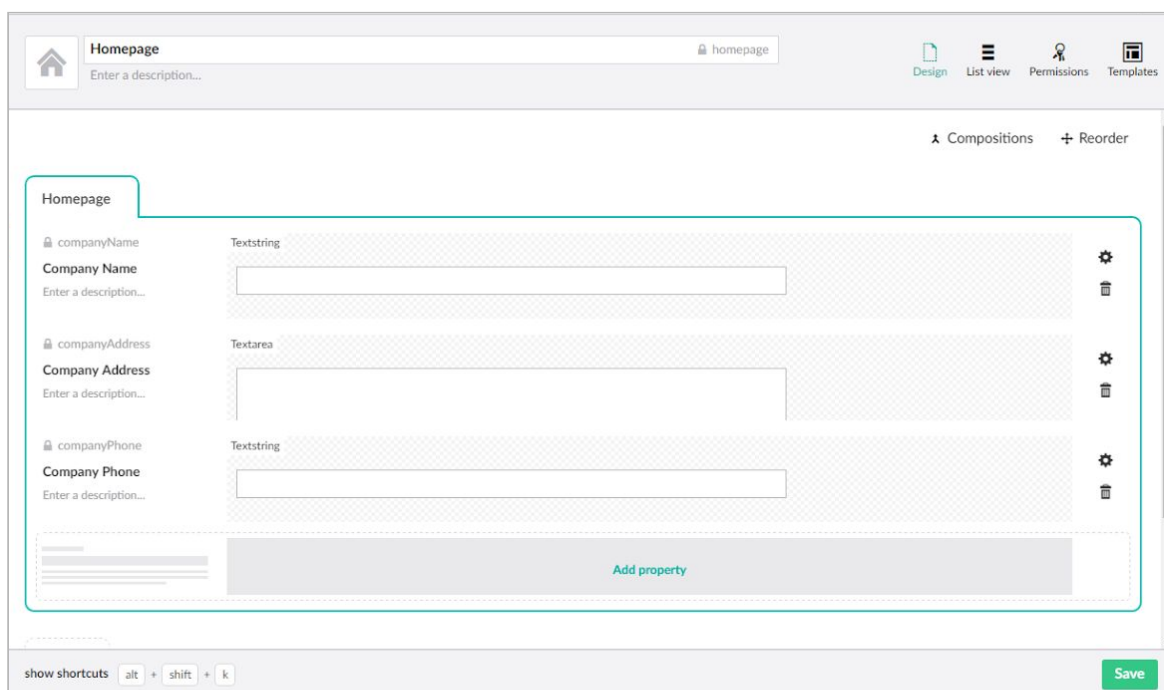
Home Page (homePage)

Property Name	Editor	Alias
Company Name	Textstring	companyName
Company Address	Textarea	companyAddress
Company Phone	Textstring	companyPhone



Notice : The above is a specification of the properties a given document type should have. These will appear in the workbook every time you need to create a document type. Follow it as a checklist as you define new document types.

When the Home Page document type is complete, it will look like this:



Notice : For this initial chapter, the process is described in great detail. In future chapters, we will provide less detailed information, but provide all the information you need in the specification boxes provided.

Create another document type

Following the same process as before, create another document type, using the specification below, and ensure it has an editor-friendly description and icon.

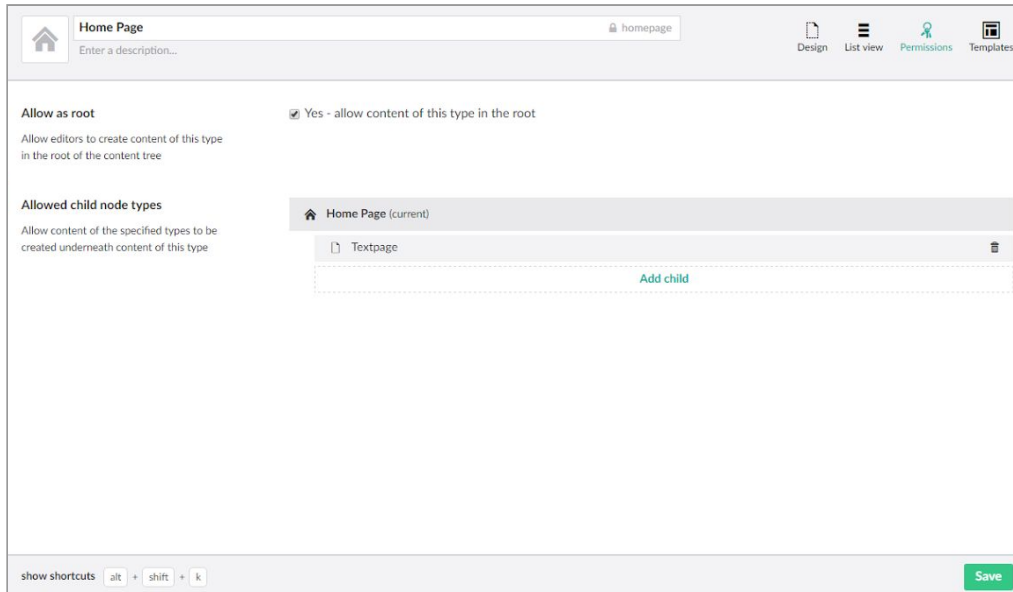
 Text Page (textPage)		
Property Name	Editor	Alias
Body Text	Richtext Editor	bodyText

Ensure you have entered all of the names, alias' and appropriate editors correctly, then continue to configuring permissions.

Configure document type permissions

Configure the **Home Page** document type to be allowed as a 'root node', and to allow content of type **Text Page** to be created beneath it:

1. Open the **Home Page** document type
2. Click the "**Permissions**" Tab
3. Check the "**Yes - Allow content of this type in the root**"
4. Click "**Add Child**"
5. Choose "**Text Page**"
6. Save the document type



Configure the **Text Page** document type to allow further content of type **Text Page** to be created below it.

1. Open the **Text Page** document type
2. Click the “**Permissions**” Tab
3. Click “**Add Child**”
4. Choose “**Text Page**”
5. Save the document type



Tip : Whenever you create a new document type, you explicitly need to allow it as content **below** other types (even itself). You define this on the **parent** type, not the child.

Basic content structure

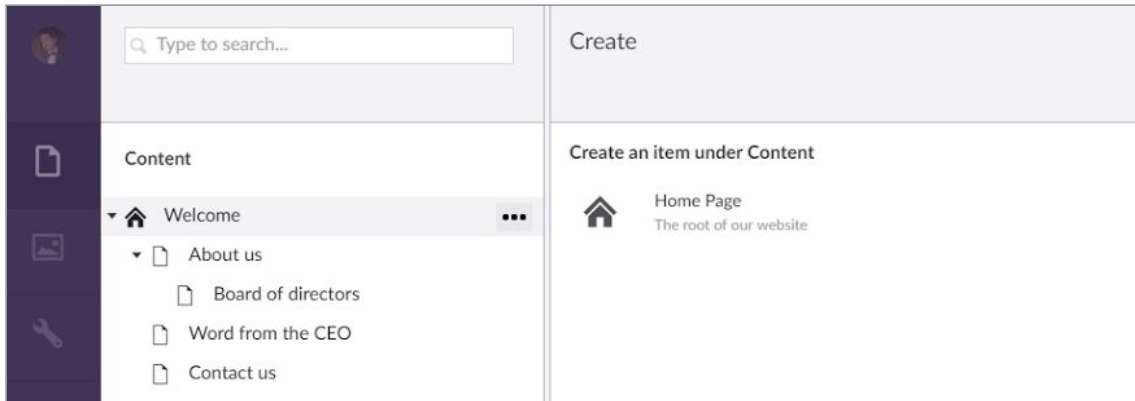
With the document types configured, a basic content structure can be created:



Go to content: For this part of the exercise, we will use the content section of the backoffice

1. Click the ... next to the “Content” headline in the tree and choose “**Home Page**”. This opens the content editor. Name the page “Welcome” and fill in the 3 fields “Company Name”, “Company Address”, and “Company Phone” with the appropriate company information of your choice, save and publish the new page.

2. Click the ... next to the "Welcome" node in the tree and choose **"Text Page"** - give the page a name and save and publish the page.
3. Create 2 more child pages below the welcome page
4. Create an additional child below one of the child pages:



The assignment is complete when a simple content structure with 3-4 pages is saved and published in a similar arrangement to this example:

- ❑ **Welcome** (homePage)
 - ❑ **About Us** (textPage)
 - ❑ **Board of Directors** (textPage)
 - ❑ **Word from the CEO** (textPage)
 - ❑ **Contact Us** (textPage)

The content cannot yet be viewed in the browser by our site visitors, this will be handled in the next exercise.



Go further : If you need to move or sort the content, you can do so by clicking the options icon, choose "do something else", and then select either sort, move or one of the other content options from the options menu.

Exercise 2 - Presentation

Now that we have a basic content structure to view and navigate, we will add some html to the currently empty templates. We will also add basic navigation which will update automatically as we add more content.

In this exercise, we will:

1. Update templates with new html
2. Add a stylesheet
3. Insert dynamic content data in our templates
4. Build navigation for our site

Update templates with new html

To get something to look at in the browser, we will copy pre-made html into our 3 template files. This gives a good baseline for displaying dynamic content later.



Go to settings: For this part of the exercise, we will use the settings section of the backoffice

1. Expand the templates folder, you will see 2 templates. **“Home Page”** and **“Text Page”**
2. Click the options ... icon on the templates tree, choose “create”
3. Name the template **“Layout”**
4. Open the html folder inside your Course files and copy the contents of **“layout.html”**
5. Paste the contents into the template below **“The fun starts here”**

- Open the **“Home Page”** template and copy paste the content of **“homepage.html”** and set its master template to **Layout**.
- Do the same with the **“Text Page”** template and the file **“textpage.html”**, and set the master template to **“Layout”**



Notice : If you view any page on your site now, you see a common error when working with templates, which is: **“The RenderBody method has not been called for layout page”**. This is due to the templates knowing that they should be connected, but not where they should be connected.

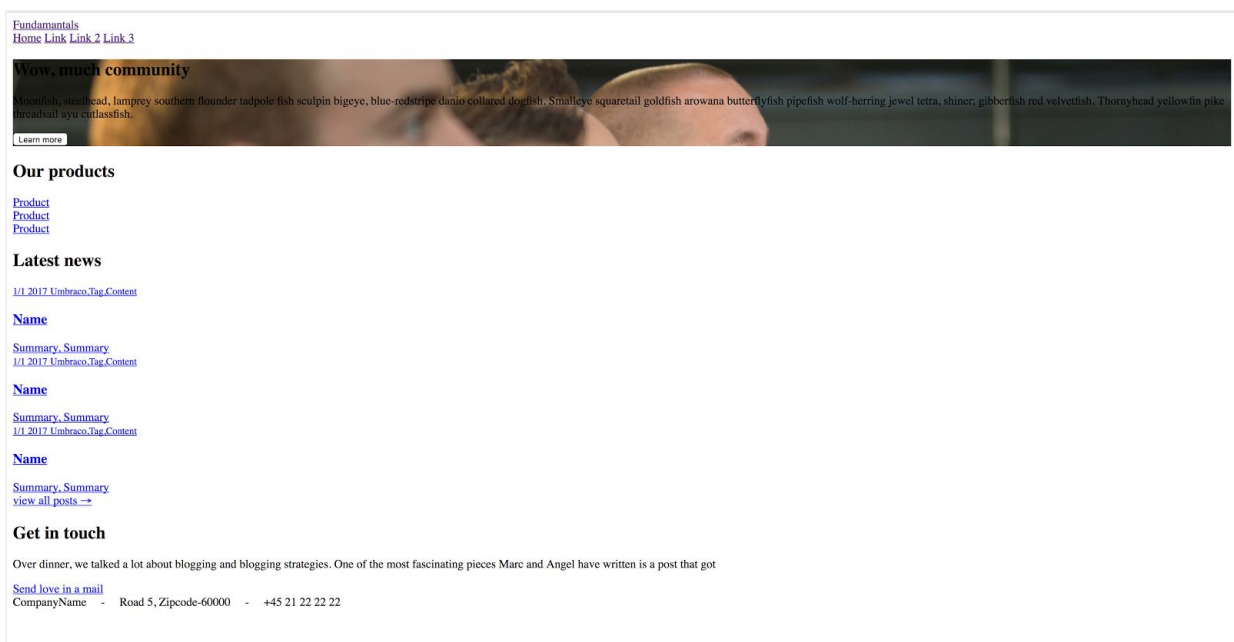
- Open the Layout template, place the caret on **line 54** and click the **“Sections”** button, choose **“Render Child Template”** and click **“Insert”**, this inserts the following:

```
<!-- End of top navigation -->
```

```
@RenderBody()
```

```
<!-- Beginning of footer -->
```

- Open your website and confirm your template html works and is correctly combined.



Add a stylesheet

To style your html output, you need to add a stylesheet to the website, this is achieved via the Umbraco backoffice.

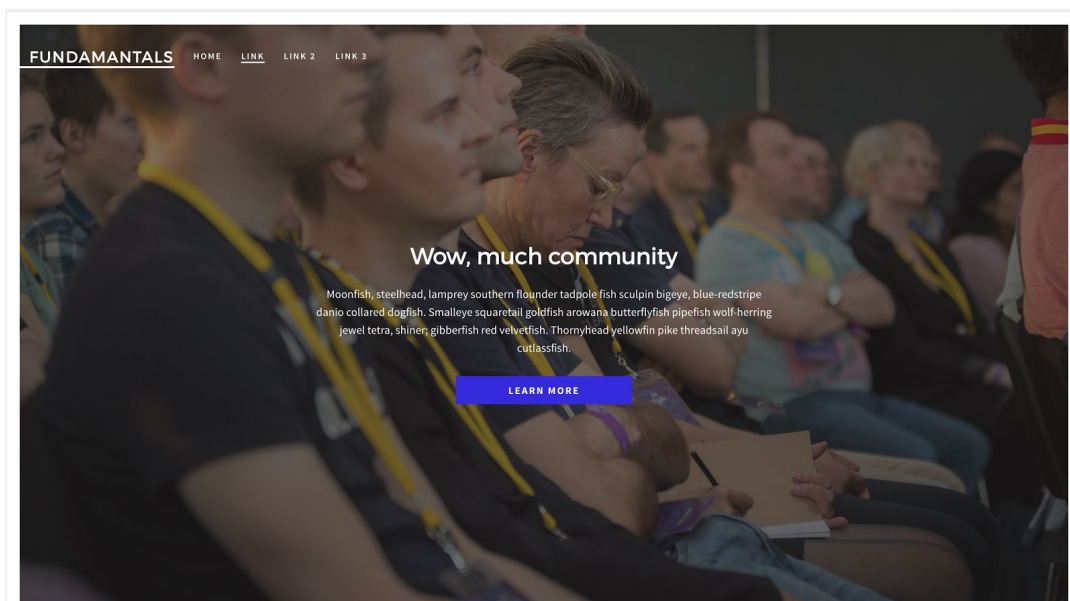


Go to settings: For this part of the exercise, we will use the settings section of the backoffice

1. Click the ... options icon next to “Stylesheets” and create a new stylesheet, name it “**Style**”
2. Copy the contents of the **Style.css** file located in the Course Files folder into the stylesheet and click “**Save**”
3. Refresh the site in the browser and ensure the stylesheet is applied



Tip : It is not magic that the stylesheet is applied. When a stylesheet named “Style” is created, Umbraco automatically stores a file at “/css/style.css”. This file is referenced in the html you pasted in earlier on **line 20**. - so you can also add a stylesheet by just copying the css file to /style



Go further : You can assign a stylesheet to any Richtext editor to get a more accurate preview of your content in the backoffice. Open the BodyText field on the Text Page document type and modify the settings on the editor - at the bottom of the settings is a list of available stylesheets you can assign - try it out - or ask your trainer for a demonstration.

Insert dynamic content data in our templates

Now that we have pages with static html, we need to enhance it with content entered into the backoffice. We will replace the static text with references to Umbraco content, using the properties that we added to our document types in the previous exercise.



Go to settings: For this part of the exercise, we will use the settings section of the backoffice

1. Open the “**Text Page**” template and look for the html comments indicating the static text to be replaced:

```
<section class="section section--themed section--header
section--content-center-bottom">
  <div class="section__hero-content">
    <!-- Replace this with "pageName" value of the page -->
    <h1 class="no-air">Name of the page</h1>
  </div>
</section>
```

2. Highlight the “Name of the page” text and click Insert -> Value, choose “pageName” from the list of values and click **insert**.

```
<section class="section section--themed section--header
section--content-center-bottom">
  <div class="section__hero-content">
    <!-- Replace this with "pageName" value of the page -->
    <h1 class="no-air">@Umbraco.Field("pageName")</h1>
  </div>
</section>
```

3. Do the same, further down the template, but replace the static text with the “**bodyText**” value

FUNDAMENTALS

HOME

LINK

LINK 2

LINK 3

About us

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam vitae sollicitudin lacus, et lobortis dolor. Vestibulum tincidunt magna id lorem feugiat sollicitudin in eleifend nisi. Ut eget purus quis quam aliquam hendrerit. Curabitur blandit nulla ac venenatis consectetur. Proin ac enim ac diam semper lacinia. Duis facilisis urna sit amet luctus viverra. Suspendisse quis tincidunt turpis, quis imperdiet augue. In leo est, commodo id felis hendrerit, maximus sagittis justo. Vestibulum bibendum erat ultrices leo venenatis, eu lacinia justo viverra. Nullam eleifend posuere nunc, non consequat justo dictum a. Integer gravida, purus et finibus tincidunt, lectus ipsum fringilla diam, eget tempus turpis felis non magna. Aenean in risus lacinia, maximus augue eu, fermentum dui. Duis tortor urna, molestie at nibh eget, laoreet mattis erat. Morbi fermentum turpis neque, vel sodales arcu commodo a. In tristique magna feugiat tincidunt convallis. In libero libero, viverra sed est in, ullamcorper porta erat.

CHILD PAGE

CHILD PAGE

CHILD PAGE

Insert Recursive Values

1. Open the **Layout** template and scroll to the bottom of it.
2. Replace the static text values in the footer area with the **companyName**, **companyAddress** and **companyPhone** property values, with the **recursive** checkbox checked

```

<div class="col-md-12 ta-center">
  <!-- Use the companyName value here -->
  @Umbraco.Field("companyName", recursive: true)

```



Tip: recursive values are a great way to store values in a root page and have them available to use in the templates throughout the site. Usually used for footers, section titles and meta data content.

Build a basic top navigation

1. In the **Layout** template scroll to the <header> area of the template at **line 40**
2. Inside the <nav> element, replace the static list of links with a foreach loop:

```
<nav class="nav-bar top-nav">
  <a class="nav-link" href="/">Home</a>
  @foreach(var child in Model.Content.Site().Children() ){
    <a class="nav-link" href="@child.Url">@child.Name</a>
  }
</nav>
```

Build a secondary navigation

1. In the Text Page template, find the <nav> element and modify it in the same way as the previous exercise, so the static list of links is replaced with a foreach loop:

```
<nav class="nav-bar nav-bar--list">
  @foreach(var child in Model.Content.AncestorOrSelf(2).Children() ){
    <a class="nav-link nav-link--black nav-link--air-bottom"
      href="@child.Url">@child.Name</a>
  }
</nav>
```



Go further : In some cases, you need your navigation to reflect which section of the site is active, by applying an extra css class. These checks can be made with the helpers generally known as IsHelpers:

Example: Output the class “nav-link--active” if the item being rendered is equal to, or a parent of the current page:

```
<a href="@child.Url" class="nav-link nav-link--black
  @child.IsAncestorOrSelf(Model.Content, "nav-link--active")">
  @child.Name
</a>
```

Try the above with the top and secondary navigation - consult the razor cheatsheets for more info.



Feature iterations

Now that we have the basics in place, we will look at the structured process of adding new functionality to a website. For most feature iterations, this process will roughly follow the same steps:

We configure document types to define the data required for the feature.

We then set up the necessary content structure to have a workable test setup.

We then design the different templates required to display the feature correctly.

Finally, we use razor to implement more complex iteration through our content.

Exercise 3 - News area

In this exercise, we will create a complete news area on our site, using Document types, templates and razor to build and design the feature. This exercise will also nicely tie together all the different building blocks in Umbraco, providing a coherent process for developing features.

In this exercise, we will:

1. Create two new document types
2. Use a listview to display many items
3. Create a handful of articles to display
4. Create a dynamic article listing for the news page
5. Use ImageCropper to handle article images

Create Document types

To build this news area feature, we will need two document types: one to store all the articles in a single location and another type to define the data a News article will contain.



Go to settings: For this part of the exercise, we will use the settings section of the backoffice

Create the following 2 document types:



News Area (newsArea)

No properties

Permissions:

Allow as child under **Home Page**

List view

Enable default listview



News Article (newsArticle)

Property Name	Editor	Alias
Summary	Textarea	summary
Body Text	Richtext Editor	bodyText
Categories	Tags	categories

Permissions:

Allow as child under **News Area**

Create content

Before we can go any further, we need to create a basic content structure, so we have some content to render.



Go to content: For this part of the exercise, we will use the content section of the backoffice

Create the following content structure below Welcome:

- ❑ **Welcome** (homePage)
 - ❑ **News** (newsArea)
 - ❑ **News Article 1** (newsArticle)
 - ❑ **News Article 2** (newsArticle)
 - ❑ **News Article 3** (newsArticle)

Ensure that the created content has `categories`, `summary`, and `bodyText` filled out with test content.

Building the news listing with razor

With content in place, we will work on displaying the created content by modifying the **News Area** and **News Article** Templates.

1. Copy the contents of the files **NewsArea.html** and **NewsArticle.html** into their corresponding templates
2. Ensure both templates use “**Layout**” as their Master template.

3. Replace the static text in both templates with `Umbraco.Field` references by following the html comments
4. In the **News Area** template define a query for all articles below the news area, just below the layout declaration on **line 4** :

```
@{
    Layout = "Layout.cshtml";
    var articles = Model.Content
        .Children<NewsArticle>()
        .OrderByDescending(x => x.CreateDate);
}
```

5. In the **News Area** template, replace the static news list with a for-each loop that will step through each News article in the `articles` collection and output its date, categories, title and summary.

```
@foreach(var article in articles){
    <a href="@article.Url" class="blogpost blogpost-with-image">
        <div class="blogpost-info">

            <div class="blogpost-meta">
                <small class="blogpost-date">@article.CreateDate</small>
                <small class="blogpost-cat">
                    @String.Join(", ", article.Categories)
                </small>
            </div>

            <h3 class="blogpost-title">@article.Name</h3>
            <div class="blogpost-excerpt">@article.Summary</div>

        </div>
    </a>
}
```

News

Our corporate news outlet, get the latest corp news..

5/11/2017 9:13:25 AM aadasd,news,mush

News Article 3

alsdj asjdjakhsdk jaskdjha skdjhakjsdhkajhs kjhdsa

5/11/2017 9:13:03 AM news,mash,meh

News Article 2

alsd haksjdkjashd kjasdjhas kdjhaskjdh aksjhd kjasdkjash dkjaskdjhaskj hkasjhasd



Notice : We will revisit this part of the site later on as we work with media and image croppers and the expectations of the Marketing department.

Go further: present news on the home page.

Currently, we only fetch articles as children of the current page. To re-use article content, we can render them on the **Home Page** of the website, by altering our articles query just ever so slightly:



Go further : This exercise is nearly identical to the last one, the only difference is the query being made, the rest of the technique is the same, which is why it is not included here. Go back and see what you did, and replicate it here for this exercise.

Add the following to the head of the **Home Page** template.

```
@{
    Layout = "Layout.cshtml";

    var site = Model.Content.Site();
    var articles = site.FirstChild<NewsArea>()
        .Children<NewsArticle>()
        .OrderByDescending(x => x.CreateDate)
        .Take(3);
}
```

Use the `articles` variable in a foreach-loop the same way as you did in the **News Area** template to output a list of the three articles on the **Home Page** template

Alternative: curated news on the home page.

Maybe, you do not want to just list every news article you create on the front page - maybe it's only for exclusively picked content - in that case, we can make use of a content picker!

1. Add a new property to the **Home Page** document type, with the name "Selected News" and choose **Multinode Treepicker** as the editor..
2. Modify the code from the optional exercise above so it looks like:

```
@{
    Layout = "Layout.cshtml";

    var articles = Model.Content.SelectedNews.OfType<NewsArticle>();
}
```

Use the `articles` variable in a foreach-loop the same way as you did in the previous exercise.

Go even further: List articles by tag

To only show articles with a specific tag, we can use `TagQuery` and a simple URL-based filtering, so when a user visits `your-domain/news?tag=umbraco` - only News tagged with “umbraco” will be listed:

1. On the **News area** template, modify the article query at the top of the template like so:

```
var tag = Request.QueryString["tag"];
var articles = (tag != null)
    ? Umbraco.TagQuery.GetContentByTag(tag).OfType<NewsArticle>()
    : Model.Content
        .Children<NewsArticle>()
        .OrderByDescending(x => x.CreateDate);
```

2. Below the articles query, add a query for all tags on the website:

```
var tags = Umbraco.TagQuery.GetAllTags();
```

3. Directly below the news listing `<div>` add a sidebar `<div>` to list out all the tags of the News section, and how often they have been used:

```
<div class="col-md-3">
    <nav class="nav-bar nav-bar--list">
        @foreach(var t in tags){
            <a class="nav-link nav-link--black nav-link--air-bottom"
                href="?tag=@t.Text">@t.Text (@t.NodeCount)</a>
        }
    </nav>
</div>
```

The final result should resemble this:

News

Our corporate news outlet, get the latest corp news..

5/11/2017 9:13:25 AM aadasd,news,mush

News Article 3

alsdj asjdjakhsdk jaskdjha skdjhakjsdhkajhs kjhdsa

5/11/2017 9:13:03 AM news,mash,meh

News Article 2

alsd haksjdkjashd kjasdjhas kdjhasdkdh aksjhd kjasdkjash dkjaskdjhasjk hkasjhasd

5/11/2017 9:12:36 AM news,umrbaco,mush,mash,bot

News article 1

asd,kjas kdjalskjdklkjasldkjaslkjdklk asjdkkjas lkdlas kjdlkasj lkajslkjaskd

[AADASD \(1\)](#)

[BOT \(1\)](#)

[MASH \(2\)](#)

[MEH \(1\)](#)

[MUSH \(2\)](#)

[NEWS \(3\)](#)

[UMRBACO \(1\)](#)



Go further : Try taking a look at the News Article page - how about making the Categories listed for the article, link to the News listing page, so a user can easily find other articles tagged with the same category?

Exercise 4 - Media Gallery

This exercise will guide you through the media library and how you can use media items and macros to create a reusable and configurable gallery component, that website editors can use.

In this exercise, we will:

1. Upload media to the media library
2. Create a macro + macro partial view
3. Insert a macro on our content pages

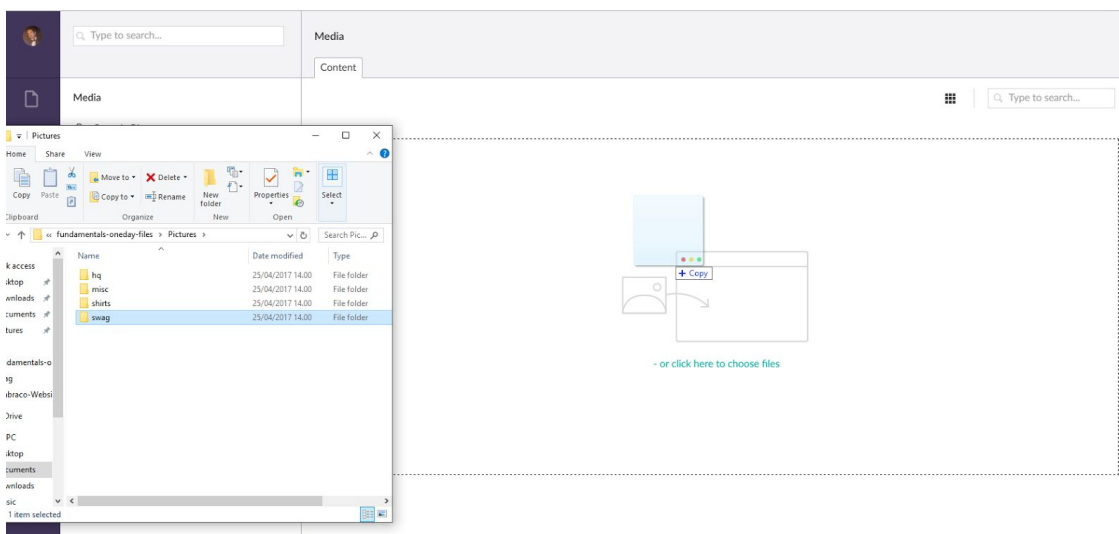
Upload media to the media library

To have some content to display with the media gallery we are building we will need to upload some pictures to our site.



Go to media: For this part of the exercise, we will use the media section of the backoffice

1. From the course files folder, inside the pictures folder, drag the **swag** folder onto the image upload area



2. This will create a media folder called “swag” containing all of the uploaded images

Create a macro

To process the pictures in the **swag** folder, and to make it accessible to our editors, we will need to add a macro.



Go to developer: For this part of the exercise, we will use the developer section of the backoffice

1. Click the ... options icon next to “**Partial View Macro Files**” folder and choose “**New partial view macro from snippet**”
2. Choose “**Gallery**” from the list of snippets
3. Name the partial view “**Gallery**” and save it
4. Expand the **Macros** tree and choose the **Gallery** macro, modify its parameters, to match those listed in the specification below:



Gallery (gallery)

Parameter Alias	Name	Type
mediaIds	Select Images	Multiple Media Picker

Editor Settings

Use in rich text editor and the grid: **yes**

Render in rich text editor: **no**

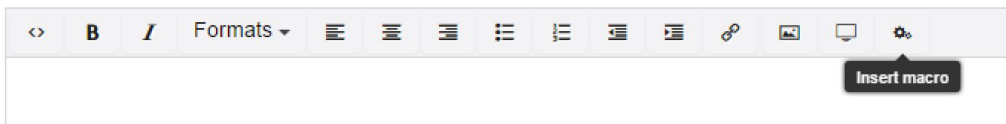
Insert macro in a news article

With the gallery macro in place, our editors can now insert a gallery into any content page which uses a rich text editor.

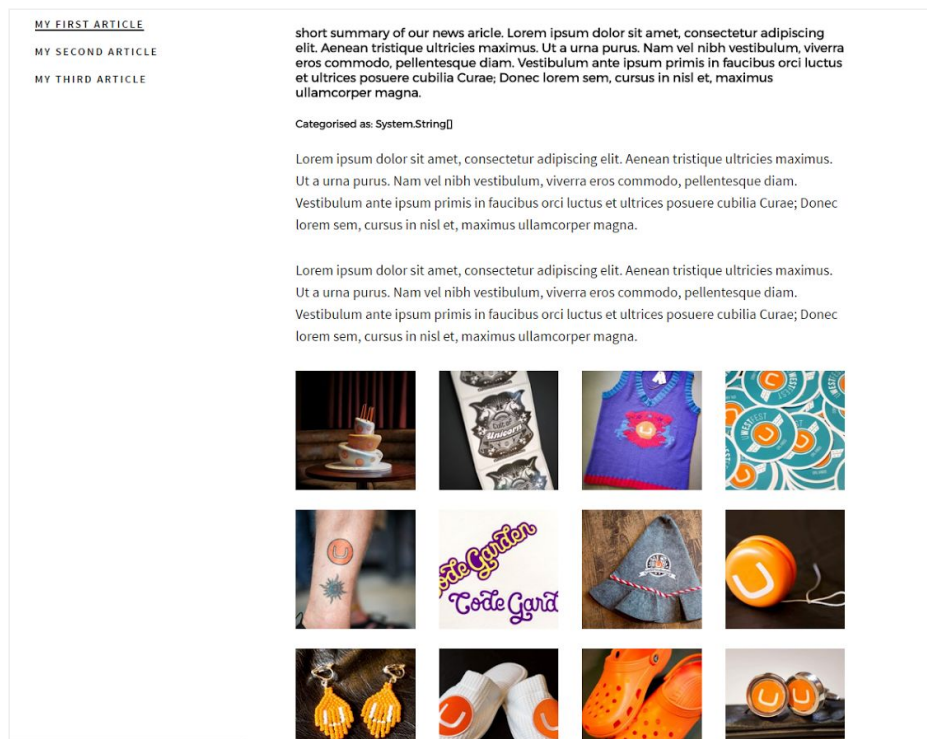


Go to content: For this part of the exercise, we will use the content section of the backoffice

1. Open a News Article and click the “Insert Macro” button on the rich text editor



2. Choose the Gallery macro from the list of macros, click select
3. Select the “Swag” folder in the media library.
4. Publish the article and view it in the browser.



Tip: If you want to see how the markup for the gallery macro is generated, head back to the **Developer** section and expand **Partial View Macro Files** and select **Gallery.cshtml**

This is “just” a view file so you can modify it to fit the design of any site. Important thing to note here is the way we get the **mediaIds** parameter:

```
@{ var mediaIds = Model.MacroParameters["mediaIds"] as string; }
```

You can do the same for your own custom partial views or if you add additional parameters.



Content Structures

Content can live in many forms, and be cross-referenced, reused and iterated through on our website in several different ways. For this part of the course, we will examine the different strategies for creating complex content structures and the pros and cons of using them with different scenarios.

Exercise 5 - Header images

This exercise will go through how we can, using picked content nodes, build a simple image carousel that we can add to multiple types of pages, using a document type composition.

In this exercise, we will:

1. Refactor templates
2. Create a header document type
3. Create a header partial view
4. Use razor and compositions to render the picked media items

Refactor templates

Refactoring is the processing of changing how code works internally, without changing the outcome of the code, in order to have a reusable header, we need to refactor our template html.



Go to settings: For this part of the exercise, we will use the settings section of the backoffice

1. Click the ... options on the **Layout** template and create a new template name “**Child Page**”
2. Move the html <section> representing the header from “**Text Page**” to “**Child Page**”
3. Add a @RenderBody() call below the pasted section, so looks like the code below:

```
@inherits Umbraco.Web.Mvc.UmbracoTemplatePage
@{
    Layout = "Layout.cshtml";
}


<section class="section section--themed section--header
section--content-center-bottom">
    <div class="section__hero-content">
        <h1 class="no-air">@Umbraco.Field("pageName")</h1>
    </div>
</section>

@RenderBody()
```

4. Also, remove the header <section> html from **News Area** and **News Article** templates
5. Set the master template on **Text Page**, **News Area** and **News Article** to “**Child Page**”

Create a Header component

1. Under Document types, create a folder named “**Components**”
2. Create a **Header** document type without a template:

 **Components / Header** (header)

Property Name	Editor	Alias
Header Image	Multiple Media Picker	headerImage

Template:
No template needed

3. Add the header as a composition on the “**News Article**” document type

Modify the Image Cropper settings



Notice : We will now open a Media Type - not a document type - it is the same concept but for media items instead of regular website content pages.

1. Open the **Image** Media type
2. Configure the **Upload Images** field - then click the cog icon to configure the **Image Cropper**:

3. Add 3 crop sizes to the cropper:

Image Cropper (umbracoFile)

Crop Alias	Width	Height
Header	1200	320
List	175	175
Grid	375	375

Create a Header partial view

1. Under Partial Views, create a new partial view “**Header.cshtml**”
2. Copy from “header.cshtml” in the course snippets folder, into the template



Notice : The pasted partial view does not contain a model declaration at the top, so it is using the string based property look up. This is due to the partial view being shared among multiple types of pages. There is a solution to this problem in the go further exercise. Also notice how the GetPropertyValue method declares that it wants a collection of published content returned.

```
@inherits Umbraco.Web.Mvc.UmbracoTemplatePage
@{
    var mediaItems = Model.Content
        .GetPropertyValue< IEnumerable<IPublishedContent> >("headerImage");
}
```

The template can iterate through picked media items (already in the template):

```

@if(mediaItems != null){
    foreach(var item in mediaItems){
        <div class="section__hero-background background-image-full"
            style="background-image: url(
                @Url.GetCropUrl(item, "Header", htmlEncode: false)
            )">

        </div>
    }
}

```

Render the picked media items

1. Open the **Child Page** template and wrap the `<section>` in an `if/else` statement which checks if the current page contains a **headerImage** value:
2. Add a `Html.Partial` call to display the header partial view, otherwise show the current header

```

@if (Model.Content.HasValue("headerImage"))
{
    @Html.Partial("header")
}
else{
    <section class="section section--themed section--header ...">
        ...
    </section>
}

@RenderBody()

```

As the header is added onto different document types, and images are set on each page, the header will update to start presenting an image carousel, instead of the blue header.

Go further: Compositions and types

By having functionally shared between multiple types of pages, our code gets inconsistent with the rest of the site, we depend on strings to find the correct property, and we even have to tell Umbraco what type of data we want back. This can actually all be done by just using the Models-builder.



Tip: A composition document type, has it's own class, the **Header** document type has a class called "Header". When used as a composition it also has what is called an Interface - which is a programmatic signature of what this class contains - these are always prefixed with an I - so if you want to check for a **Header** composition, you check for **IHeader**.

The smart thing about these signatures, is that you use them to check if a composition is present on a page, and also use them to access the Model of this specific part of the composited document type

Check for a composition

Replace the `Model.Content.HasValue(headerImage)` in the if statement with a check to detect if the current page uses a **Header** composition, so the if statement looks like below:

```
@if(Model.Content is IHeader){  
    @Html.Partial("header")  
}
```

Use the composition interface as the model

As the partial view only cares about values that exists on the Header composition, we can just use the model of the **Header** document type in the partial view - this makes our model and queries much easier to read, and ensures that the correct type of value is returned.

In the **header.cshtml** partial view file, modify the first 4 lines of the file to look like this:

```
@inherits Umbraco.Web.Mvc.UmbracoTemplatePage<IHeader>  
@{  
    var mediaItems = Model.Content.HeaderImage;  
}
```

Save the file and reload the page, everything will work as before, but with much cleaner code.

Exercise 6 - Product page

In this final exercise, we will use all of the basic building blocks of the course, and introduce two new ones: Grid Editor and Nested Content. The end result of this exercise is a rich product page, complete with a grid-powered layout, testimonials and a lead-generation form.

In this exercise, we will:

1. Create testimonial, product area, and product document types
2. Configure the grid editor
3. Configure the nested content editor
4. Create the needed content for our page
5. Create a template to display our product
6. Create a lead generating form

Create the required document types

As always, our first step is to define what data we want to store and edit.



Go to settings: For this part of the exercise, we will use the settings section of the backoffice

For this feature, **3 document types** are needed:



Components / Testimonial (testimonial)

Property Name	Editor	Alias
Title	Textstring	title
Quote	Textarea	quote
Image	Media Picker	image

Template

No template needed



Product Area (productArea)

Property Name	Editor	Alias
No properties needed		

Compositions:

Use Header composition

Permissions

Allow as child under **Homepage**



Product (product)

Property Name	Editor	Alias
Body Text	Grid Layout	bodyText
Testimonials	Nested Content	testimonials

Compositions:

Use Header composition

Permissions

Allow as child under **Product Area**

Configure the Nested Content editor


When adding the Nested Content editor to the “**Testimonials**” field, we need to configure it to use the “**Testimonial**” document type :

1. Click “Add” and add the “**Testimonial**” document type, and choose only the single tab on the document type (likely named “**Testimonial**”)
2. For the Name template, enter: `{{title}}` - to display the Title field as the title of the nested item

Configure the grid editor

The Grid editor on the **Product** document type needs to be configured, so we present as fewer options to editors as possible.

1. On the **Product** document type, configure the **Body Text** field
2. Configure the rows and cells on the grid editor to match the specification below
3. Delete the unneeded 2-column layout

 **Grid Editor**

Grid Layouts: 1 Column Layout

Ensure the row configuration for the editor matches the below

Row Name	Configuration
Headline	<div>1 cell allowing “Rich text editor” and “Headline” editors</div> <div>Richtext + Headline</div>
2 Columns	<div>2 Cells allowing “Rich text editor” and “Image” editors</div> <div><div>Richtext + image</div><div>Richtext + image</div></div>
3 Images	<div>3 Cells allowing an “image” editor</div> <div><div>Image</div><div>Image</div><div>Image</div></div>
Form	<div>2 Cells allowing a “Form” editor and a Rich Text editor</div> <div><div>Form</div><div>Richtext</div></div>

Create sample product content

We now have all the structural parts in place to start building content, so let's start building our new product page.



Go to content: For this part of the exercise, we will use the content section of the backoffice

1. Ensure the following content structure is created:

- ❑ **Welcome** (homePage)
 - ❑ **Products** (productArea)
 - ❑ **Umbraco Cloud** (product)

1. On the **Umbraco Cloud** page build up the **grid** layout so it resembles this outline:

Headline

Umbraco Cloud is the future

Rich text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris ultrices, nisi sit amet gravida rutrum

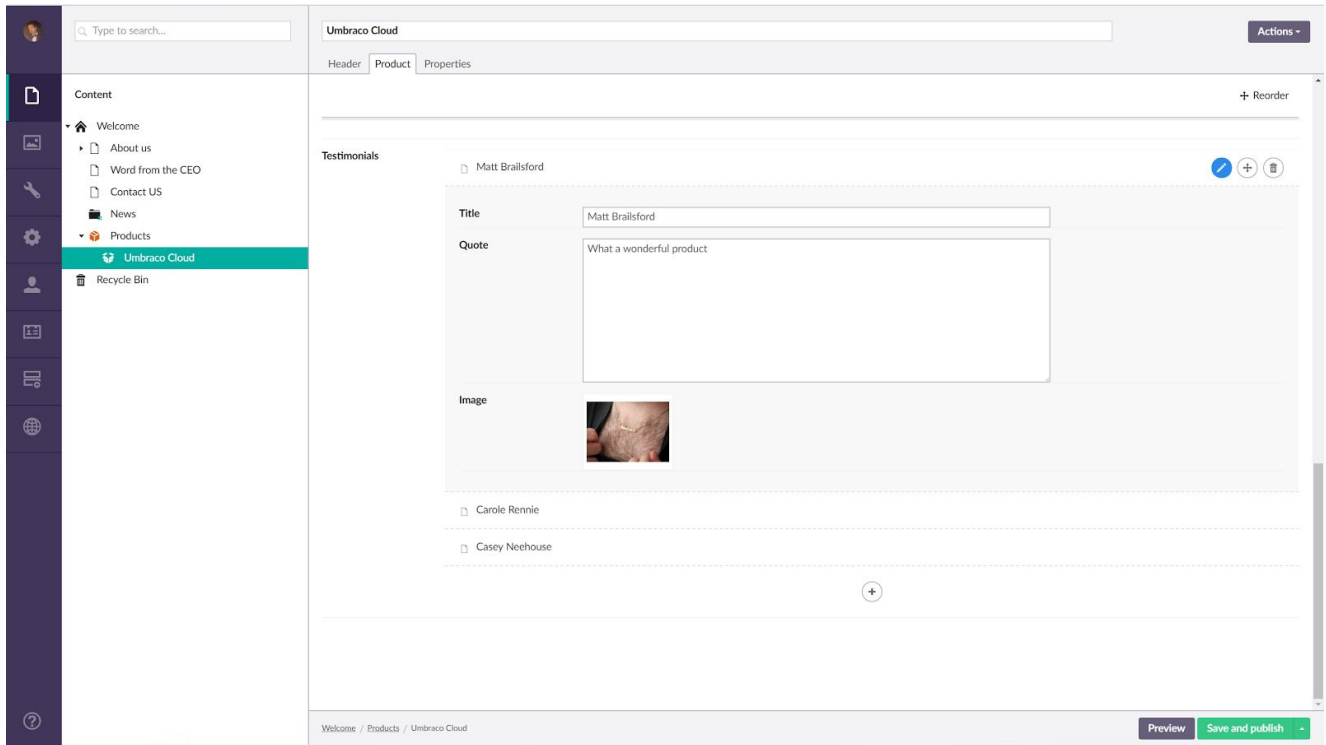
- Lorem ipsum
- Lorem ipsum
- Lorem ipsum



Image Caption



2. On the **Umbraco Cloud** page, add **3 testimonials** to the nested content editor.



Add html and razor to the product template

With the structure and content in place, edit the product template to display the content.

1. Inherit from the **Child Page** template
2. Add a basic structure and a call to render the grid body text:

```
<section class="section">
  <div class="container">
    @Html.GetGridHtml(Model.Content, "bodyText")
  </div>
</section>
```

3. Add another `<section>` element and render testimonials:

```
@{
    var testimonials = Model.Content
        .Testimonials
        .OfType<Testimonial>();
}
```


Instead of typing the entire thing below, you can copy the testimonial html from the **Home Page** template and fill out the blanks.

```
<section class="section">
  <div class="container">
    <div class="row">

      @foreach(var testimonial in testimonials){
        <div class="col-sm-6 col-lg-4">
          <div class="testimonial">
            <div class="testimonial-image"
              style="background-image: url('@testimonial.Image.GetCropUrl("grid") ')">
            </div>
            <div class="testimonial-content">
              <h3 class="testimonial-title">@testimonial.Title</h3>
              <p class="testimonial-text no-air">@testimonial.Quote</p>
            </div>
          </div>
        </div>
      }

    </div>
  </div>
</section>
```

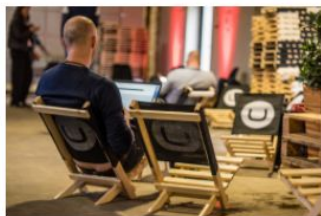
Our page should now look like so:

Umbraco Cloud

Umbraco Cloud is the future

Donec sollicitudin molestie malesuada. Cras ultricies ligula sed magna dictum porta. Vivamus magna justo, lacinia eget consectetur sed, convallis at tellus. Proin eget tortor risus. Nulla quis lorem ut libero malesuada feugiat. Donec rutrum congue leo eget malesuada. Donec sollicitudin molestie malesuada. Donec sollicitudin molestie malesuada. Sed porttitor lectus nibh. Vivamus magna justo, lacinia eget consectetur sed, convallis at tellus.

Donec sollicitudin molestie malesuada. Cras ultricies ligula sed magna dictum porta. Vivamus magna justo, lacinia eget consectetur sed, convallis at tellus. Proin eget tortor risus. Nulla quis lorem ut libero malesuada feugiat. Donec rutrum congue leo eget malesuada. Donec sollicitudin molestie malesuada. Donec sollicitudin molestie malesuada. Sed porttitor lectus nibh. Vivamus magna justo, lacinia eget consectetur sed, convallis at tellus.



Automatic Upgrades

Sed porttitor lectus nibh. Curabitur arcu erat, accumsan id imperdiet et, porttitor at sem. Quisque vel it nisi, pretium ut lacinia in, elementum id



Team Development

Sed porttitor lectus nibh. Curabitur arcu erat, accumsan id imperdiet et, porttitor at sem. Quisque vel it nisi, pretium ut lacinia in, elementum id



Content Flow

Sed porttitor lectus nibh. Curabitur arcu erat, accumsan id imperdiet et, porttitor at sem. Quisque vel it nisi, pretium ut lacinia in, elementum id

It will of course vary based on the images and content entered by you

Create a lead-generating form

With content in place, the last thing we need is to create a basic form to collect data from interested visitors - for that we will use Umbraco forms.



Go to forms: For this part of the exercise, we will use the media section of the backoffice

1. Click the options icon next to the forms folder and create a new form
2. Ensure the form follows the specification below:



Contact Us

Field Name	Editor	Alias
Your Name	Short Answer	yourName
Your Email	Show Answer	yourEmail
Your Message	Long Answer	yourMessage

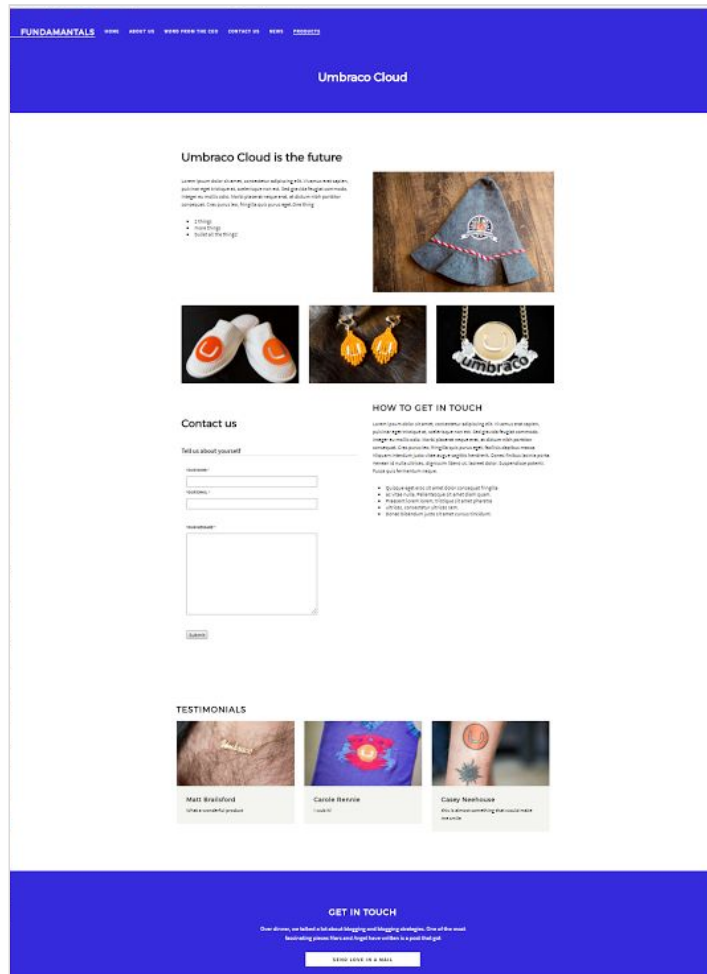
3. Save the form



Go to content: For this part of the exercise, we will use the content section of the backoffice

Go back to the Umbraco Cloud product page and insert a form row - if configured correctly - it will now prompt you to select a form and enter text into a rich text editor field.

When completed, the full page should look like the following image:



Go further

Finally, we can polish the grid and the editor experience for each row configuration



Go further : This is only scratching the surface of what is possible with the grid - to enhance the editor experience even further, you can include custom css in the rich text editor and apply custom styles to the rows and cells of the grid

Add custom css to the rich text editor

1. Go to the settings section and create a new stylesheet named “rich text editor”
2. Right-click the stylesheet and click “create”...
3. Add a new style by giving it a name and css selector like:
 - a. **Name:** Header
Alias: h2
 - b. **Name:** Sub header
Alias: h3
4. Open the grid property editor inside the “Product Page” document type editor - scroll to the bottom of the settings and select the newly created “Rich text editor” stylesheet

Add a custom css class to the 3-images row

1. Click the row containing the 3 images
2. Click the Cog-icon for accessing the settings
3. In the css-class field type “**image-component**”
4. Refresh the Umbraco Cloud page and see how 40px of padding has been added
5. This is due to the included stylesheet containing a definition for an “**image-component**” class - you just simply applied it to this row

Umbraco Partnership

Congratulations on completing your training course! Consider taking your Umbraco career to the top level and enjoy 19 powerful benefits

Developers working at Umbraco Certified Gold Partners, enjoy a number of benefits which makes working professionally with Umbraco an even more delightful experience. In total, the Gold Partnership comes with a whopping 19 delightful benefits, but we've hand-picked the three very best benefits for you as a developer:

Support and Advising directly from Umbraco HQ

A Gold Partnership automatically gives you a full Umbraco Support Agreement. This means you can use specialists from Umbraco HQ to give you Architectural Advising on a new project, ensuring you have the best possible foundation and peace-of-mind - or simply get advice and support should you ever get stuck during a project.

Free training and certification

We offer free training twice a year for one of your in-house developers at our special "Gold Partner Academy". A great opportunity for you, or your colleague, to stay up to date.

A financial high-five!

As both an incentive and an appreciation, we offer our Gold partners the opportunity to join the Cloud Referral Program which rewards you with a recurring referral fee every time you create a project on Umbraco Cloud. So now, working with Umbraco Cloud won't just make your everyday job simpler, it will also earn you money.

Sign up today!

The three benefits above are only a small taste of what you get with an Umbraco Gold Partnership. You can read more about the three available types of Umbraco partnership and their corresponding benefits and requirements at this page <https://umbraco.com/partners/become-a-partner/>

Sign up online or get in touch with Anders T. Sørensen anders@umbraco.com

Umbraco Training Courses

Become more visible in the Umbraco marketplace. Show off your Umbraco expertise as a certified developer by attending our official Umbraco courses

Fundamentals

Fundamentals is suitable for both developers, and content editors who want to learn about the newest features of Umbraco.

Earns you 100 certification points

Extending the Backoffice

Learn how to take full advantage of all the Umbraco backoffice can offer.

Earns you 50 certification points

MVC and Visual Studio

Work with the Umbraco core APIs, learn about powerful debugging and performance tuning as well as the many extension points Umbraco offers .net developers.

Earns you 50 certification points

Application Integration

Connect a 3rd party application with Umbraco and learn how you approach the complex integrations of 3rd party data and business logic with Umbraco.

Earns you 50 certification points

Searching and Indexing

Learn how to optimise the search function on your website using Examine, making it a breeze for your visitors to find relevant content.

Earns you 50 certification points