Front-end Web Development in Sitebuilder

Exercise Book

Table of Contents

[Getting started 5](#_Toc512249870)

[Installing Visual Studio Code 5](#_Toc512249871)

[Optional: Download and install Git 6](#_Toc512249872)

[Download the Sitebuilder template project 6](#_Toc512249873)

[Option 1: Using Git 6](#_Toc512249874)

[Option 2: Download manually 6](#_Toc512249875)

[Install the Live Server extension 7](#_Toc512249876)

[Section 1: Anatomy of a Sitebuilder page 9](#_Toc512249877)

[Exercise 1.1: Use a .less file to change the style of a page 9](#_Toc512249878)

[Exercise 1.2: Using standard ID7 variables in styling 10](#_Toc512249879)

[Exercise 1.3: Using jQuery to add JavaScript functionality to a page 11](#_Toc512249880)

[Section 2: Using built-in ID7 components 12](#_Toc512249881)

[Exercise 2.1: Add a jumbotron to your page with a button in it 12](#_Toc512249882)

[Exercise 2.2: Add a modal with a YouTube video 13](#_Toc512249883)

[Exercise 2.3: Implement some Bootstrap clichés in a page design 16](#_Toc512249884)

[Section 3: Testing and QA 19](#_Toc512249885)

[Exercise 3.1: Analysing colour contrast 19](#_Toc512249886)

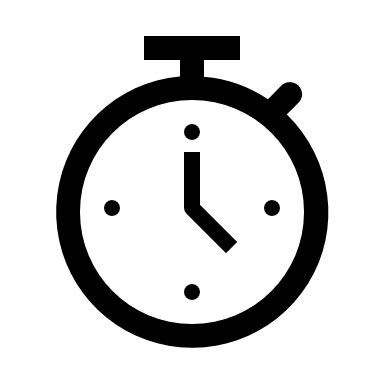
[Section 4: Advanced CSS layouts 21](#_Toc512249887)

[Exercise 4.1: Using flexbox for equal height columns 21](#_Toc512249888)

[Exercise 4.2: Using CSS grid to lay out contents 24](#_Toc512249889)

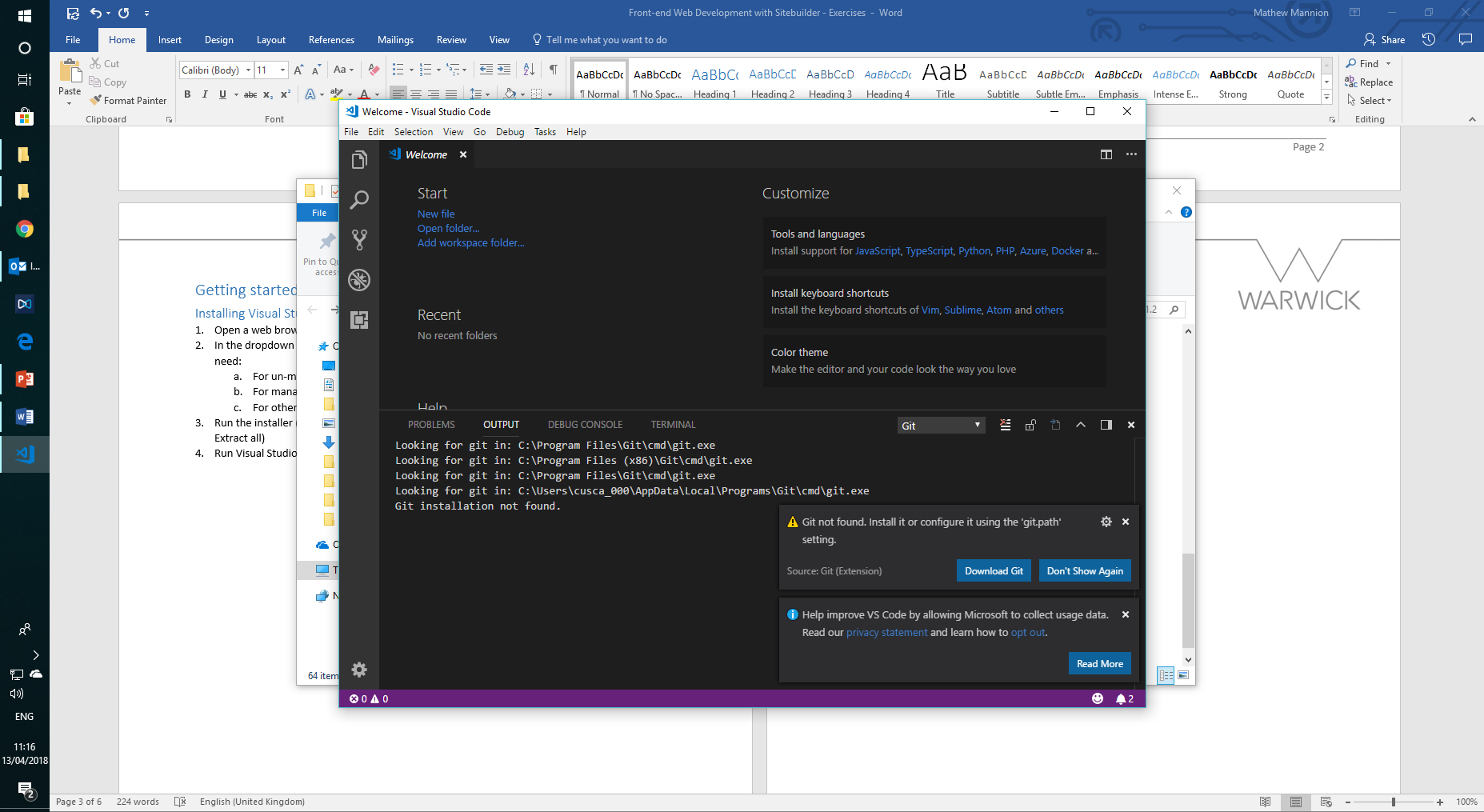
# Getting started

## Installing Visual Studio Code

 15 minutes

1. Open a web browser and visit <https://code.visualstudio.com>
2. In the dropdown next to the **Download** button, click the arrow and download the version you need:
   1. For un-managed Windows, use “Windows x64 – Installer”
   2. For managed Windows, use “Windows x64 – .zip”
   3. For others, use the package for your operating system
3. Run the installer (or extract the zip file somewhere if you’re using the .zip version – right click -> Extract all)
4. Run **Visual Studio Code** (or double click on “Code” in the extracted folder for the .zip version)

You should see a screen something like this:



## Optional: Download and install Git

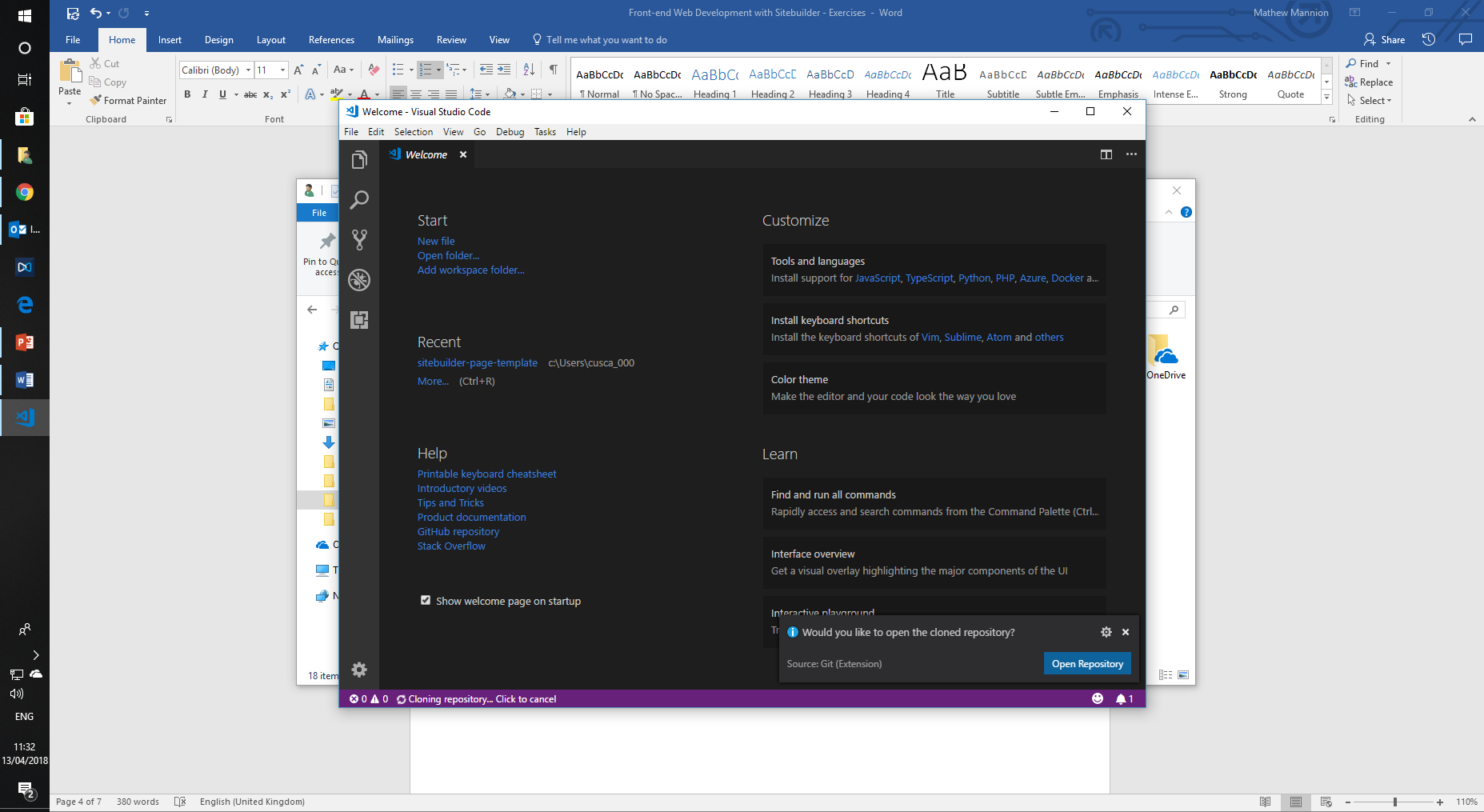
1. Click the “**Download Git**” button in the pop-up in the bottom right – this will open <https://git-scm.com>
2. Click the Download link on the right and run the downloaded installer – accept the defaults for everything (but uncheck “View release notes” at the end)
3. Close and re-open Visual Studio Code – you shouldn’t get the notification about “Git not found” any more

Click the X at the right-hand side of the panel starting PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL to close it.

## Download the Sitebuilder template project

### Option 1: Using Git

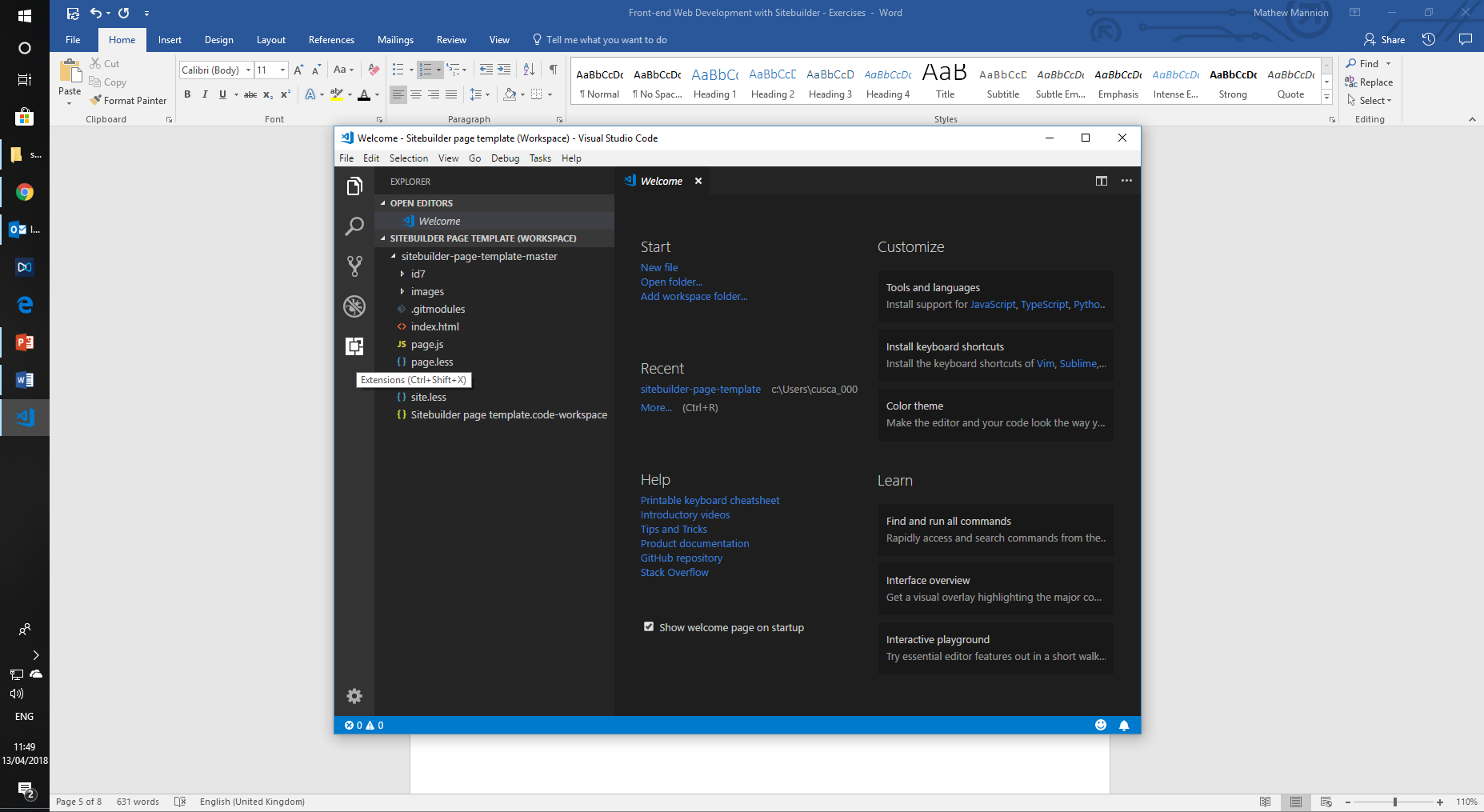
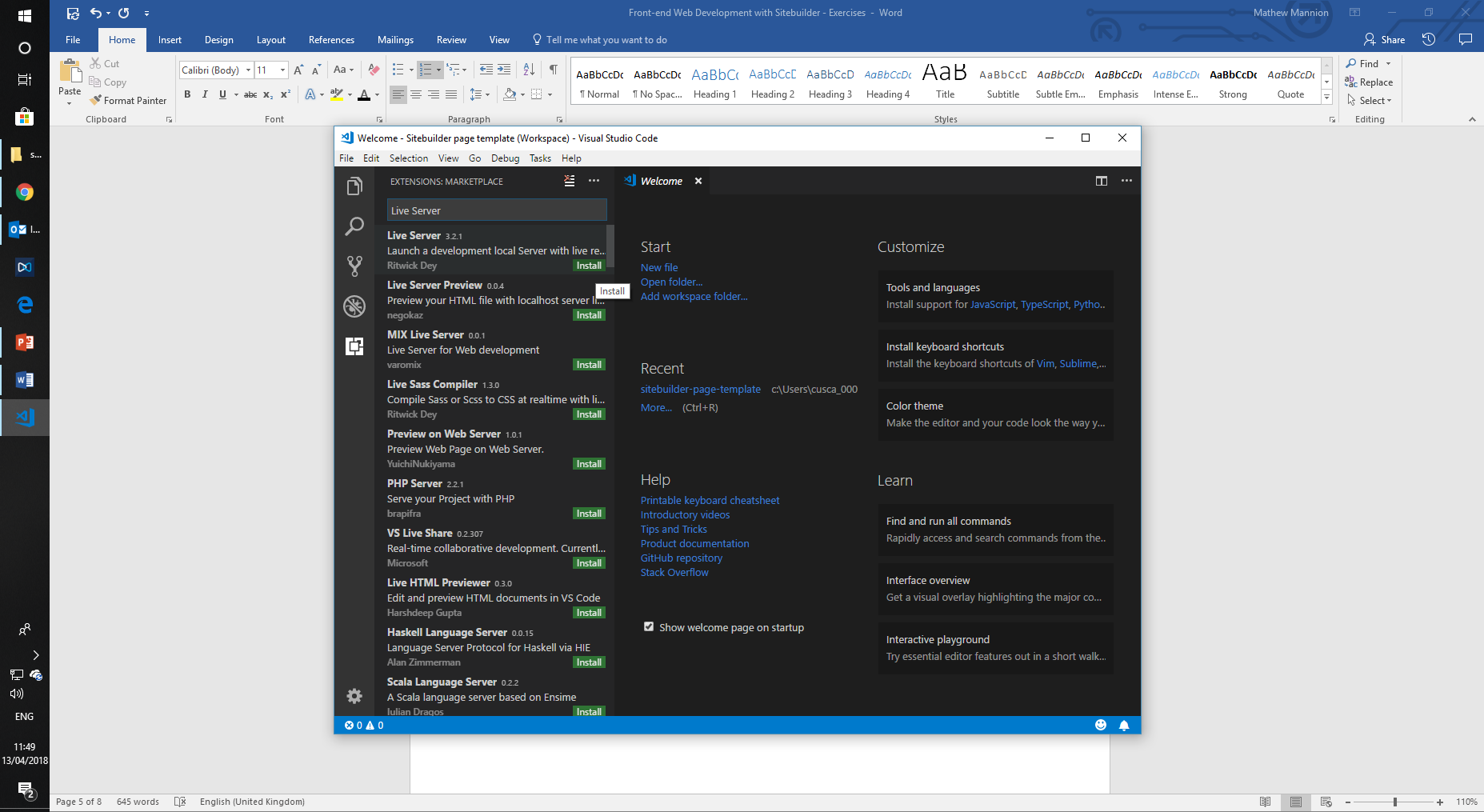
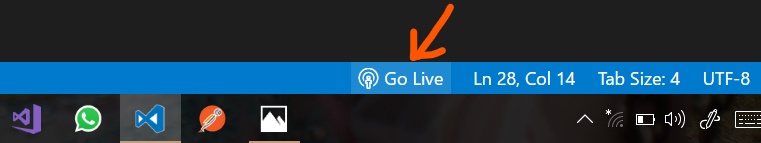
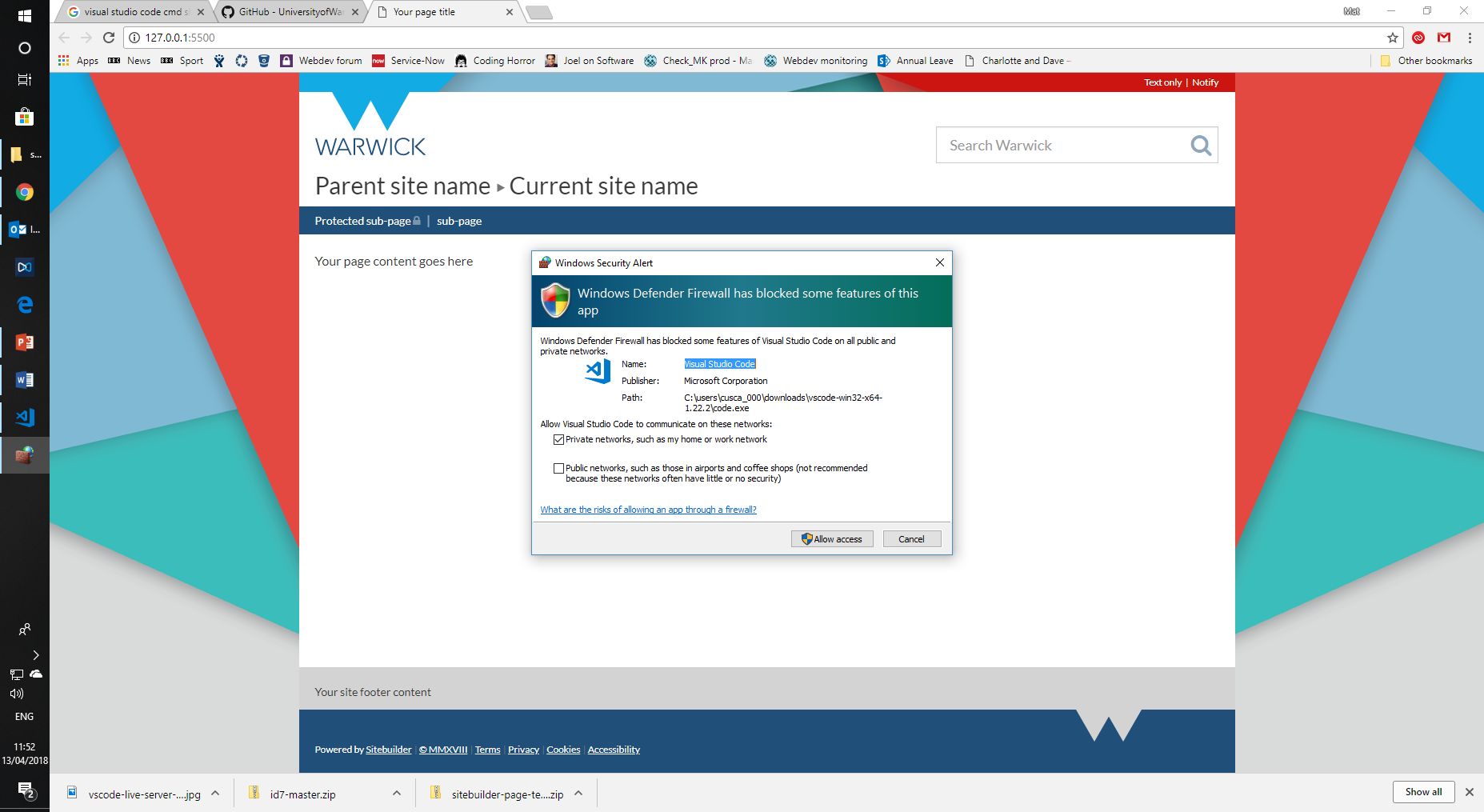
**Advanced users:** If you’re already familiar with GitHub, you might want to fork the repository at <https://github.com/UniversityofWarwick/sitebuilder-page-template> to your own account to save your work back to later (you can skip to “Install the Live Server extension”)

1. Open **Visual Studio Code** and press Ctrl+Shift+P (⇧⌘P on macOS) to open the **command palette**
2. Type git clone and select the **Git: Clone** option
3. For the repository URL, use   
   https://github.com/UniversityofWarwick/sitebuilder-page-template.git   
   and press enter
4. Use the default parent directory (or change it to one where you want the project to be downloaded to)
5. In the bottom right, click “**Open Repository**” when prompted  
   
6. Right click on the id7 folder in the directory structure on the left, and select “**Open in Command Prompt**”
7. In the prompt that opens, type git submodule update --init to populate the id7 folder. You’ll only have to do this once.
8. The id7 folder should turn blue and have an S to the right of it.

### Option 2: Download manually

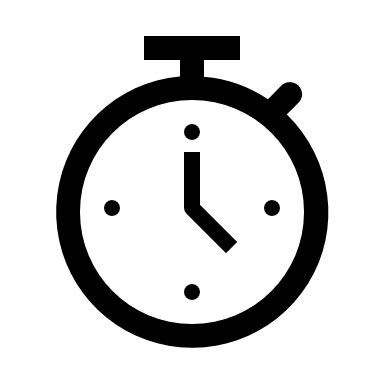
1. Go to <https://github.com/UniversityofWarwick/sitebuilder-page-template>
2. Click the green “**Clone or download**” button on the right and select “Download zip”. It will download a file called sitebuilder-page-template-master.zip
3. Extract the zip file to where you want to keep the project (right click > Extract all)
4. Go to <https://github.com/UniversityofWarwick/id7>
5. Click the green “**Clone or download**” button on the right and select “Download zip”. It will download a file called id7-master.zip
6. Extract the zip file somewhere and open the id7-master folder. **Select all the files inside it and move them to the id7 folder underneath your sitebuilder-page-template project**
7. In Visual Studio Code, go to **File > Open Workspace…**, navigate to your sitebuilder-page-template project and click on the Sitebuilder page template.code-workspace file

## Install the Live Server extension

1. In **Visual Studio Code**, click the **Extensions** button on the far-left bar or go to **View > Extensions** in the menu.  
   
2. In the search bar at the top, search for “**Live Server**” and click on the green Install button on the top result  
   
3. The button will change to “**Installing**” and then to “**Reload**”. Click Reload and then click the Explorer tab in the top left
4. You should now see a “**Go Live**” button in the blue bar at the bottom
5. Click on the **Go Live** button. If you get a Windows Defender Firewall prompt you can click Cancel on it – you don’t need to allow access  
   
6. A web browser will open with the template Sitebuilder page in. When you make changes to the project in Visual Studio and save the file, this will automatically reload with the changes.

# Section 1: Anatomy of a Sitebuilder page

## Exercise 1.1: Use a .less file to change the style of a page

 15 minutes

**Goal: Use a .less file to modify the styles of a Sitebuilder page. This is mostly an exercise to ensure you have all the components set up correctly.**

1. Open **Visual Studio Code** and **Go Live** if you haven’t already, so you can see the Sitebuilder page template in a web browser
2. Open index.html in the Explorer page so it displays in the editor window. This is the HTML of the template page.
3. Find where the page content is (it should say <p>Your page content goes here</p>, on around line 130), and replace it with some HTML that includes a link, then save it:

<h2>A heading</h2>

<p>Here is some text with <a href="warwick.ac.uk">a link</a></p>

1. Open the page.less file in the Explorer pane so it displays in the editor window (it should start with // Import some handy ID7 files to provide variables etc.
2. At the bottom of the file, add some CSS that makes the main content text bigger and red (you probably don’t want to do this normally…)

.id7-main-content {

color: red;

font-size: 20px;

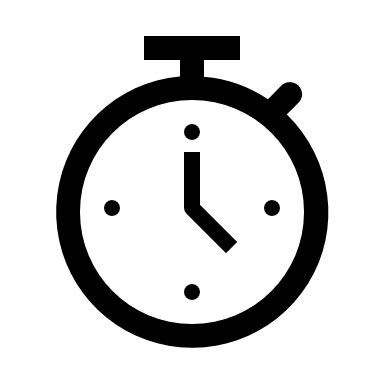
}

1. When you save the page.less file, the template will reload in the web browser and you will be able to see the text on the page is bigger and red.

**Extra time exercise:** Alongside the page.less file is a site.less file that controls the ID7 “brand” of the site itself. This sets the brand colour (i.e. navigation, links, the colour behind WARWICK in the masthead, etc.) and the imagery that’s used in the masthead and the left and right borders (stored under the images/ folder here).

Try changing the site.less file to change the brand colour, or use different images. We don’t allow Sitebuilder editors to do this normally (at the moment) as we run checks against the colours and imagery when we apply this in Sitebuilder (e.g. contrast checks)

## Exercise 1.2: Using standard ID7 variables in styling

 15 minutes

**Goal: Using .less variables means you don’t have to repeat yourself and you can see all your style declarations in one place. You only have to change it in one place rather than in many places.**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Open page.less in the Explorer pane so it’s in the edit window.
3. In your existing declaration to change the colour of text in .id7-main-content, change the colour declaration to the @id7-brand-red-bright variable:

.id7-main-content {

color: @id7-brand-red-bright;

font-size: 20px;

}

1. You should see that the colour of text is a slightly different red.
2. Open index.html in the Explorer page so it displays in the editor window.
3. Add a <div> below the current content of the page (but still inside <div class="column-1-content">) that we can style:

<div class="my-custom-box">

<h2>What’s in the box?</h2>

<p>Here is <a href="warwick.ac.uk">a link</a> inside the box</p>

</div>

1. Back in page.less, style the box by giving it a background and some padding:

.my-custom-box {

background: @id7-brand-red-bright;

color: white;

padding: @grid-gutter-width;

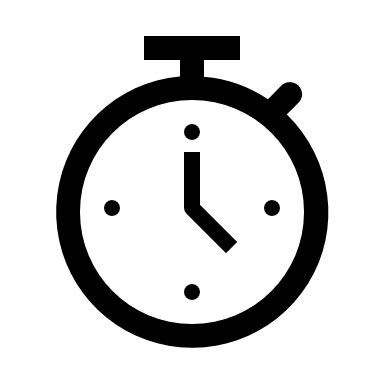
}

**Extra time exercise:** Changing the link colours inside an element is one of the most common styling changes, and one of the fiddliest – you have to set hover colours etc. ID7 provides a mixin (function) that you can call in your .less to do this automatically:

.link-colour(@colour, [@hover]); (the hover colour is optional, it will default to 20% darker than @colour).

Set the link colour inside your custom box to white. Try changing it to inherit as well, what’s the difference?

## Exercise 1.3: Using jQuery to add JavaScript functionality to a page

 15 minutes

**Goal: Add a “close” link to our box and make the box disappear when it’s clicked**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Open index.html in the Explorer page so it displays in the editor window.
3. Inside the <div class="my-custom-box"> that you created in the previous exercise, add a Close link. We set the link’s href to "#" to indicate it doesn’t go anywhere, and a class that we can refer to later.

<p><a class="close-link" href="#">Close</a></p>

1. Open up page.js (it should be empty) and add the following code:

jQuery(function($) {

$('.close-link').on('click', function () {

$('.my-custom-box').addClass('hidden');

});

});

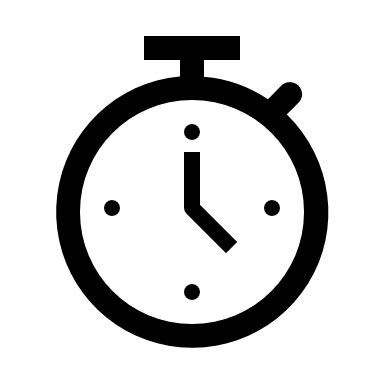
1. **You will have to refresh the page in the browser for the JavaScript to run**. When you click the Close link, the box should disappear.

**Extra time exercise:** Already know a bit of JavaScript? Go ahead and add some animation. You’ll want to add some content below your box so you know that it’s working.

The goal is to have the box fade out, and then for the content below it to slide up once the fade has completed.

# Section 2: Using built-in ID7 components

## Exercise 2.1: Add a jumbotron to your page with a button in it

 15 minutes

**Goal: Add a jumbotron component to your template page that has a call-to-action button in it (don’t worry about the button doing anything)**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Edit your page content in index.html and add some HTML for a jumbotron based on what’s in the Bootstrap documentation at [https://getbootstrap.com/docs/3.3/components/#jumbotron](https://getbootstrap.com/docs/3.3/components/)
3. You’ll need to make a couple of changes for it to display correctly in a Sitebuilder page:
   1. We don’t allow <h1> in page content, so change it to a <h2>
   2. You’ll need to wrap the entire <div class="jumbotron"> in <div class="container-fluid">
4. Switch the link in the documentation to an actual button element.
5. Your complete HTML should look something like this:

<div class="container-fluid">

<div class="jumbotron">

<h2>Hello, world!</h2>

<p>Here is my lead text</p>

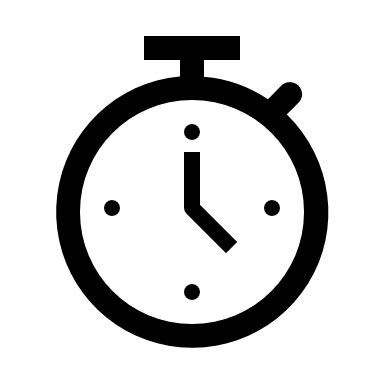
<p><button type="button" class="btn btn-primary">See more</button></p>

</div>

</div>

**Extra time exercise:** Use your page.less file to style the jumbotron on your page.

## Exercise 2.2: Add a modal with a YouTube video

 25 minutes

**Goal: Make your call-to-action button open a modal with a YouTube video in it**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Edit your page content in index.html and **under your jumbotron** add the HTML for a modal from the Bootstrap documentation: <https://getbootstrap.com/docs/3.3/javascript/#modals>  
     
   **GOTCHA: Because of the way modals work, you’ll need to make sure the HTML’s not inside another element or it’ll look odd – make sure it’s beneath your <div class="container-fluid">**
3. Change the modal’s title and remove the “Save changes” button from the example HTML, and empty the body out ready for our video. Your modal HTML should look something like this:

<div class="modal fade" tabindex="-1" role="dialog">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>

<h4 class="modal-title">Check our this video!</h4>

</div>

<div class="modal-body">

</div>

<div class="modal-footer">

<button type="button" class="btn btn-default" data-dismiss="modal">Close</button>

</div>

</div><!-- /.modal-content -->

</div><!-- /.modal-dialog -->

</div><!-- /.modal -->

1. So that we can bind our button to the modal, we’ll need to give it an id. Where you’ve got <div class="modal fade", add an ID attribute so you have   
   <div id="video-modal" class="modal fade"
2. The “usage” part of the Bootstrap modal documentation tells you how to add the right attributes to your button <https://getbootstrap.com/docs/3.3/javascript/#modals-usage> – you’ll need to add both data-toggle and data-target – don’t forget to add a # before the id of your modal.

1. Your jumbotron’s HTML should look something like this:

<div class="container-fluid">

<div class="jumbotron">

<h2>Hello, world!</h2>

<p>Here is my lead text</p>

<p><button type="button" class="btn btn-primary" data-toggle="modal" data-target="#video-modal">See more</button></p>

</div>

</div>

1. In your browser you should now see that your template page opens a modal when you click on the button, and you can close it by either clicking outside, using the Close button in the footer, or the X icon in the top right of the header.
2. We’re now ready to add the code for our YouTube video. Go to any video, right click on it and click “Copy embed code”, then put that inside your modal-body  
     
   **GOTCHA: There is currently a Sitebuilder bug that causes the “allowfullscreen” part of the embed code to be lost on save. It’s fine for your template page, but on a normal Sitebuilder page you’ll have to change it to allowfullscreen="allowfullscreen"**
3. Your video should now be displayed when the modal opens.

**Extra time exercise:** You might have noticed that if you start playing the video and then close it, the video continues to play. There’s a note in the Bootstrap documentation about “Embedding YouTube videos” that links to some resources about this exact thing. You’ll need to use some JavaScript to stop the video playing when the modal is hidden and there’s information in the documentation about the event you can hook into to do this. See if you can get this working.

We implemented this on the Undergraduate Study homepage by using YouTube’s own JavaScript API inside the modal. Can you see how this works?

<div class="modal-body">

<div id="campaign-video"></div>

<script type="text/javascript">

// Load the YouTube IFrame Player API code asynchronously.

var tag = document.createElement('script');

tag.src = "https://www.youtube.com/iframe\_api";

var firstScriptTag = document.getElementsByTagName('script')[0];

firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

// Create an <iframe> (and YouTube player) after the API code downloads.

var player;

function onYouTubeIframeAPIReady() {

player = new YT.Player('campaign-video', {

height: '480',

width: '854',

videoId: 'KtnLEySy4Rg'

});

}

jQuery(function ($) {

$('#campaign-video-modal')

.on('shown.bs.modal', function () {

if (player) player.playVideo();

})

.on('hide.bs.modal', function () {

if (player) player.stopVideo();

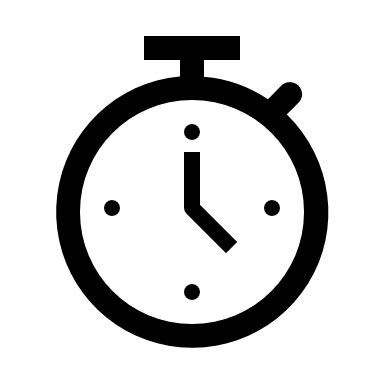
});

});

</script>

</div>

## Exercise 2.3: Implement some Bootstrap clichés in a page design

 30 minutes

**Goal: Ever get a sense of déjà vu when viewing a web page? It’s time for your customers to experience the same**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. At the **top** of your template page’s main content in index.html (i.e. directly inside and at the top of column-1-content) add your hero image. We can use picsum.photos to get a random placeholder image just for this purpose at the size we want (1200x600). We add 3 CSS classes to have it display how we’d like:
   1. img-responsive is a standard Bootstrap class to make an image scale down as the container gets smaller
   2. full-width flushes the image up to the left and right borders of the main content
   3. flush-top removes the gap above the top of the image
3. We wrap the image in a div so the sizing of the container doesn’t affect the image itself:

<div class="full-width flush-top">

<img class="img-responsive" src="https://picsum.photos/1200/600">

</div>

1. Below this, we’ll add a set of large icons that give an idea of what we’re all about. We’ll use Bootstrap’s grid system to add them in three columns, and we’ll use some helper classes to style them:

<div class="full-width container brand-bg text-center">

<div class="row">

<div class="col-sm-4">

<i class="fa fa-5x fa-bullhorn"></i>

<h2>Strength</h2>

<p>Our persistence is unrivalled.</p>

</div>

<div class="col-sm-4">

<i class="fa fa-5x fa-shield"></i>

<h2>Valour</h2>

<p>We will not be swayed.</p>

</div>

<div class="col-sm-4">

<i class="fa fa-5x fa-birthday-cake"></i>

<h2>Cake</h2>

<p>The fuel of champions.</p>

</div>

</div>

</div>

1. The important classes we have here are:
   1. full-width container – this makes the div span the full width
   2. brand-bg – gives the div the same background as the brand colour
   3. text-center – centers all text inside the div
   4. row – specifies a row in the grid
   5. col-sm-4 – specifies a column should take up 4 spaces on sm screen sizes and above
   6. fa fa-5x – a font awesome icon at 5x the text size
   7. fa-bullhorn, fa-shield, fa-birthday-cake – some icons from <https://fontawesome.com/v4.7.0/icons/>
2. We’ll need to add a bit of styling in our less file to fix the heading colour and give it some padding at the top and bottom, as well as some margin below it:

.brand-bg {

.header-colour(white);

padding: @grid-gutter-width 0;

margin-bottom: (@line-height-computed / 2);

}

1. The content boxes are a little easier as they’re just normal content aligned in a grid. We can use img-responsive like before here:

<div class="row">

<div class="col-sm-4">

<img class="img-responsive" src="https://picsum.photos/400/200">

<h2><a href="#">An article title</a></h2>

<p>An abstract from the article.</p>

</div>

<div class="col-sm-4">

<img class="img-responsive" src="https://picsum.photos/400/200">

<h2><a href="#">An article title</a></h2>

<p>An abstract from the article.</p>

</div>

<div class="col-sm-4">

<img class="img-responsive" src="https://picsum.photos/400/200">

<h2><a href="#">An article title</a></h2>

<p>An abstract from the article.</p>

</div>

</div>

1. Finally, we’ll add another coloured box with a quote in it, with the image pulled to the right with a good old-fashioned float:

<div class="full-width container brand-bg">

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

<img class="pull-right" src="https://picsum.photos/200/200">

<blockquote>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>

<footer>Someone famous in <cite title="Source Title">Source Title</cite></footer>

</blockquote>

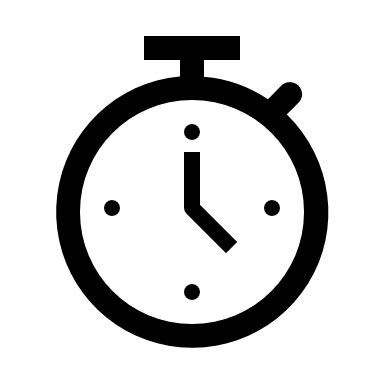
</div>

</div>

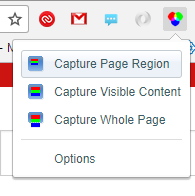
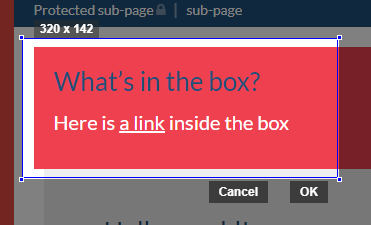
1. While it’s not perfect (we’d need to add extra less to fix contrast problems with the blockquote footer and make our spacing more consistent) we’ve been able to add a lot of common elements.

# Section 3: Testing and QA

## Exercise 3.1: Analysing colour contrast

 15 minutes

**Goal: Make sure your alert box meets WCAG AA colour contrast guidelines**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Make sure you have a my-custom-box div with contents – see the end of exercise 1.2 for the HTML and less if you’ve lost it.
3. If you’re not using Google Chrome, open it and go to <http://127.0.0.1:5500/index.html> - you should see your template page
4. Install the Color Contrast Analyzer extension in Chrome from <https://chrome.google.com/webstore/detail/color-contrast-analyzer/dagdlcijhfbmgkjokkjicnnfimlebcll?hl=en>
5. Select the extension icon from the top right and select “Capture Page Region”
6. Drag the selection box around the headings and text of your coloured box and click OK  
   
7. Make sure “Level AA, Medium Bold and Large Non-Bold Text” is selected from the dropdown at the top (and click Rescan if you had to change it). You should see that the contrast is reasonable for the text and the link, but not for the heading. At level AAA, the contrast for neither is sufficient.
8. Close the tab that was opened by the extension.

1. Change the heading colour to a sufficiently contrasting colour, white. You can either just target h2, or you can use the .header-colour(@colour) mixin from ID7 which is a better choice. Nest the declaration inside the .my-custom-box one:

.my-custom-box {

background: @id7-brand-red-bright;

color: white;

padding: @grid-gutter-width;

.link-colour(white);

.header-colour(white);

}

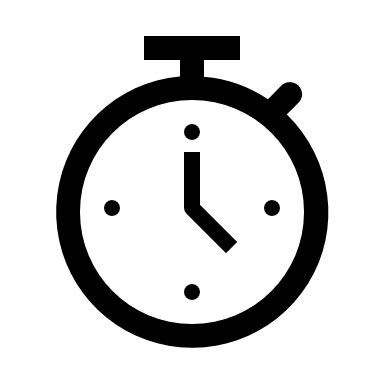
1. Repeat steps 5-7. You should now see that the contrast between the background and the heading text colour is sufficient to pass AA.

**Extra time exercise:** Less has a built-in function called contrast(). The documentation for it is here: <http://lesscss.org/functions/#color-operations-contrast>

Instead of specifying white for the colour, use the contrast() function to base it off @id7-brand-red-bright. Try changing the background colour so contrast() selects a dark colour instead of a light one.

# Section 4: Advanced CSS layouts

## Exercise 4.1: Using flexbox for equal height columns

 30 minutes

**Goal: Use flexbox to design a 3-column layout that lays out 3 boxes so they have the same width and height. Each box has a button at the bottom that should be pinned to the bottom of the box.**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Under any current content in your template page, add a new set of boxes with sample content, wrapped in a container div:

<section class="marketing-boxes">

<div class="box">

<img class="img-responsive" src="https://lorempixel.com/640/360/food/1">

<h3>Lovely bacon</h3>

<p>Bacon ipsum dolor amet tongue kielbasa turkey biltong pastrami leberkas jerky bacon. Pork loin rump pig hamburger filet mignon corned beef. Prosciutto flank tenderloin filet mignon rump.</p>

<a class="btn btn-brand btn-block">Read more</a>

</div>

<div class="box">

<img class="img-responsive" src="https://lorempixel.com/640/360/food/2">

<h3>Cupcakes!</h3>

<p>Cupcake ipsum dolor sit amet bonbon candy bonbon. Cookie caramels toffee sesame snaps chocolate bar.</p>

<p>Cupcake ice cream lollipop tiramisu pastry danish caramels carrot cake. Cupcake halvah chocolate cake powder ice cream.</p>

<p>Danish pastry jelly-o cookie cake marzipan cotton candy. Halvah toffee candy canes lollipop marshmallow pudding.</p>

<a class="btn btn-brand btn-block">Read more</a>

</div>

<div class="box">

<img class="img-responsive" src="https://lorempixel.com/640/360/food/3">

<h3>Bob Ross</h3>

<p>Anyone can paint. Volunteering your time; it pays you and your whole community fantastic dividends.</p>

<a class="btn btn-brand btn-block">Read more</a>

</div>

</section>

1. Your template page should now have the content stacked on top of each other. We need to lay them out horizontally using flexbox. We’ll also give the boxes a light background to make them more distinctive.  
     
     
     
   There are a few things to notice here:

.marketing-boxes {

display: flex;

.box {

flex-grow: 1;

flex-basis: 0;

background: #eeeeee;

padding: @grid-gutter-width;

+ .box { // .box + .box - a box following another box

margin-left: @grid-gutter-width;

}

}

}

* 1. We set flex-grow as 1 (to equally grow each box) but we also need to set flex-basis as 0 to avoid it trying to automatically calculate the width. This is a common use case and the short-hand flex: 1; would achieve the same.
  2. Flexbox doesn’t have a concept of a gutter between items (most hacks are around setting width using calc(), or setting width: 32% or similar, but that’s brittle because it relies on there always being 3 items. Instead, we set a left margin on every item except the first using CSS’s + notation.

1. On our template page we should now see we have 3 boxes equally spaced, but the buttons still don’t display at the bottom of the content. We can use flexbox again on each box to position the button at the “end” of the column (other less is omitted for brevity):

.marketing-boxes {

.box {

display: flex;

flex-direction: column;

.content {

flex: 1;

}

}

}

1. This doesn’t work straight away as it relies on expanding the “content” of each box. Wrap all of the <p> elements in each box in a <div class="content"> and it will push the button to the bottom of each box.

**Extra time exercise:** We haven’t really done mobile-first design here. We need to wrap our layouts in media queries in order to only have them apply when the screen hits certain breakpoints. You can use the following to do this:

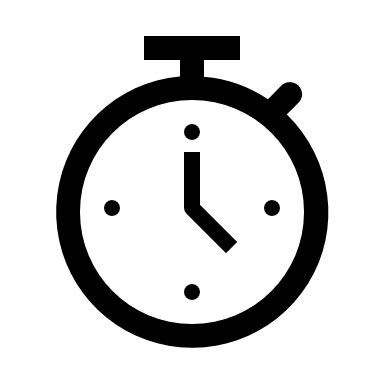
@media (min-width: @screen-sm-min) {} // Tablets in portrait

@media (min-width: @screen-md-min) {} // Some tablets, small desktops

@media (min-width: @screen-lg-min) {} // Most desktops

Experiment with only enabling the flexbox layout at certain breakpoints, and wrapping onto multiple lines.

## Exercise 4.2: Using CSS grid to lay out contents

 30 minutes

**Goal: Insite uses a design with flowing boxes that span across multiple grid tracks. Implement a design that uses grid auto placement to place items on the grid as necessary.**

1. Open **Visual Studio Code** and **Go Live** if you aren’t already.
2. Under any current content in your template page, add a grid container and some content boxes inside it. We’ll give them a background colour in the HTML just to help us see where everything’s being laid out without having to implement things separately:

<div class="grid-layout">

<div class="news-stories" style="background-color: aquamarine;">

Some news stories

</div>

<div class="quick-links" style="background-color: lightsalmon;">

Quick links

</div>

<div class="twitter-feed" style="background-color: lightskyblue;">

Our twitter feed

</div>

<div class="whats-on" style="background-color: lightgreen;">

What's on at the University

</div>

<div class="featured-article" style="background-color: lightseagreen;">

A featured article

</div>

<div class="expert-comment" style="background-color: lightslategray;">

An expert comment link

</div>

<div class="featured-video" style="background-color: lightcoral;">

A video and some other content

</div>

<div class="staff-profile" style="background-color: lightcyan;">

A featured staff profile

</div>

<div class="callout-story" style="background-color: lightyellow;">

A callout story

</div>

</div>

1. In page.less, we’ll define our grid with 4 columns. We won’t define any rows outright but will instead use grid-auto-rows to create rows at the right height (300px), and we’ll specify a gap equivalent to the gutter width to get consistent spacing:

.grid-layout {

display: grid;

grid-template-columns: repeat(4, 1fr);

grid-auto-rows: 300px;

grid-gap: @grid-gutter-width;

}

1. We can already see in our template page that our grid is being laid out:  
     
   
2. We can now specify where our grid items will span multiple rows or columns in our .less file:

.grid-layout {

// Existing properties not shown for brevity

.news-stories, .featured-video, .staff-profile {

grid-column-end: span 3; // 3 cells wide

}

.featured-article {

grid-column-end: span 2; // 2 cells wide

}

}

1. We now have something that looks pretty much exactly like the insite homepage in terms of layout without specifying a single width element. Let’s imagine we wanted quick links to span over two lines instead of one:

.grid-layout {

// Existing properties not shown for brevity

.quick-links {

grid-row-end: span 2; // 2 rows high

}

}

1. That’s not so good, it’s left white spaces in the grid. We can set the grid-auto-flow property to dense to fill in holes earlier in the grid when items are positioned later:

.grid-layout {

// Existing properties not shown for brevity

grid-auto-flow: row dense;

// Existing properties not shown for brevity

}

1. It’s not perfect, but we could easily add more things in there.

**Extra time exercise:** As before, use media queries to have the grid respond at smaller content sizes. You’ll probably want to only enable grid at sm and above; try having fewer columns on sm and md. Note that if you only have two columns, you’ll need to change the cells that span 3 columns to only span 2 as well.