ASP.NET Core MVC Logo: Pablo Iglesias

Building elegant applications with ASP.NET Core MVC 2.1 and CoreUI 2 (Bootstrap 4)

May 21, 2018 POSTS

→ Versión en ESPAÑOL

Share

Follow @mvelosop

This is the **UPDATED** and **REVISED** version of my previous post: Building elegant applications with ASP.NET MVC Core 2 and Bootstrap 4 using CoreUI

In this post I'll explain how to adapt the just released CoreUI template (v2.0.0), based on Bootstrap 4, to use it as a base for ASP.NET MVC Core 2.1 applications.

Actually, "adapting" is mostly just running my first JS program, submitted as a PR to CoreUI's repo.

Although this is an ASP.NET Core MVC 2.1 specific post, the general idea should, at least, serve as a guide for other frameworks outside the .NET world.

> **Key Takeaways**
>
> 1. Key concepts about handling client-side packages with **npm**
> 2. Understanding the relations between main views, layout views and partial views.
> 3. Handling nested layouts
> 4. Customizing Identity 2.1 views from UI packages

Source code

Post: AspNetCore2CoreUI-2.0.zip (release 2.0 from repo)
Repo: https://github.com/mvelosop/AspNetCore2CoreUI

Context

It's been a little while since I wrote the first post on CoreUI and as of these dates (MAY-2018), there are new versions/release-candidates for both ASP.NET MVC Core (v2.1) and CoreUI (v2.0.0) and I've also become to know a little more on front-end subjects, so I thought it would be a good time to publish an updated and revised post.

The process for adapting CoreUI is going to be a bit different from the previous post, to begin with, it's now all centered in **npm**, as **bower** and **gulp** have been removed from both VS and CoreUI and CoreUI is using npm's task execution capabilities.

On the other hand, the process will be much faster thanks to a Node program I developed and submitted as a PR to CoreUI. I also expect to make the process much clearer.

As of MAY-2018, ASP.NET Core 2.1 is only supported by Visual Studio 2017 15.7 or newer.

I don't expect this post to be too affected by the final release of all involved products, but I'll update it in case it's necessary.

Platform and Tools

- Visual Studio 2017 Community Edition v15.7 or later
  (go to Visual Studio's download page for other versions).
- EditorConfig Language Service for Visual Studio
- CoreUI v2.0.0
- .NET Core 2.1.0-rc1 with SDK 2.1.300-rc1 - x64 SDK Installer
  (go to .NET Core's download page for other versions).
- Node 9.11.1
- npm 5.8.0

The adaptation process becomes much easier using a file compare tool. I recommend Beyond Compare, that I've been using since 1998, but there are other similar tools that should work just as well.

- Beyond Compare, from Scooter Software

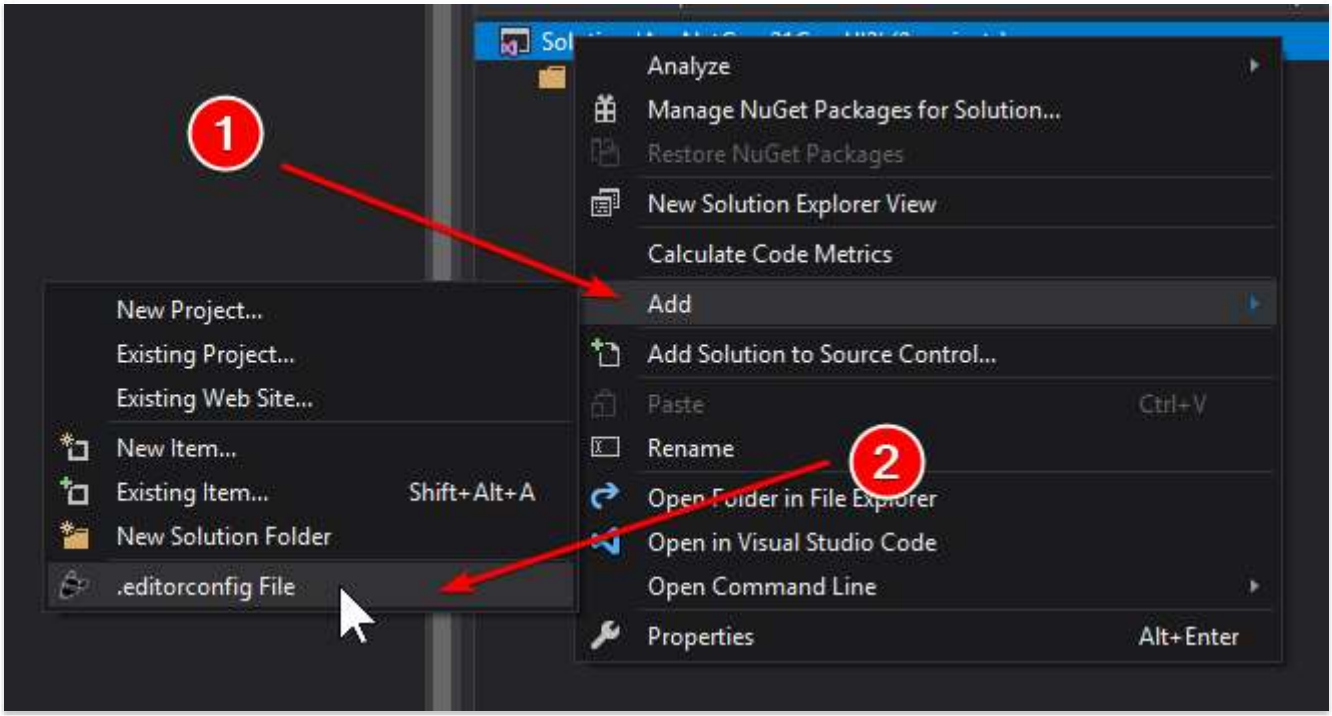
Step by step

1 - Create an ASP.NET MVC Core 2.1 project

Let's start by creating a standard MVC application, using Visual Studio 2017's built-in template.

1.1 - Create a blank solution

1. Create a "blank solution" named **AspNetCore21CoreUI2** with a Git repository
2. Add an **.editorconfig** file to the solution to standardize some formatting options for the project (after installing EditorConfig Language Services)
3. Add a **src** solution folder



.editorconfig
[AspNetCore2CoreUI-2.0]

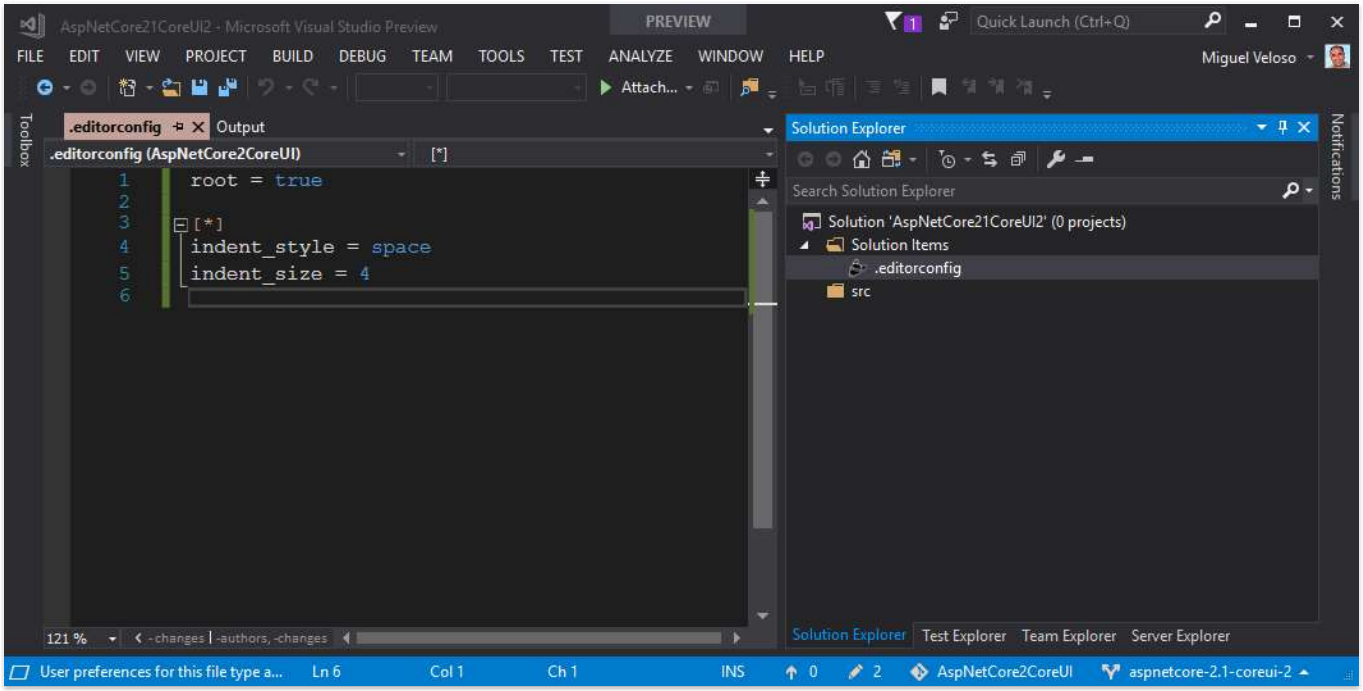
```
root = true

[*]
indent_size = 4
indent_style = space

```

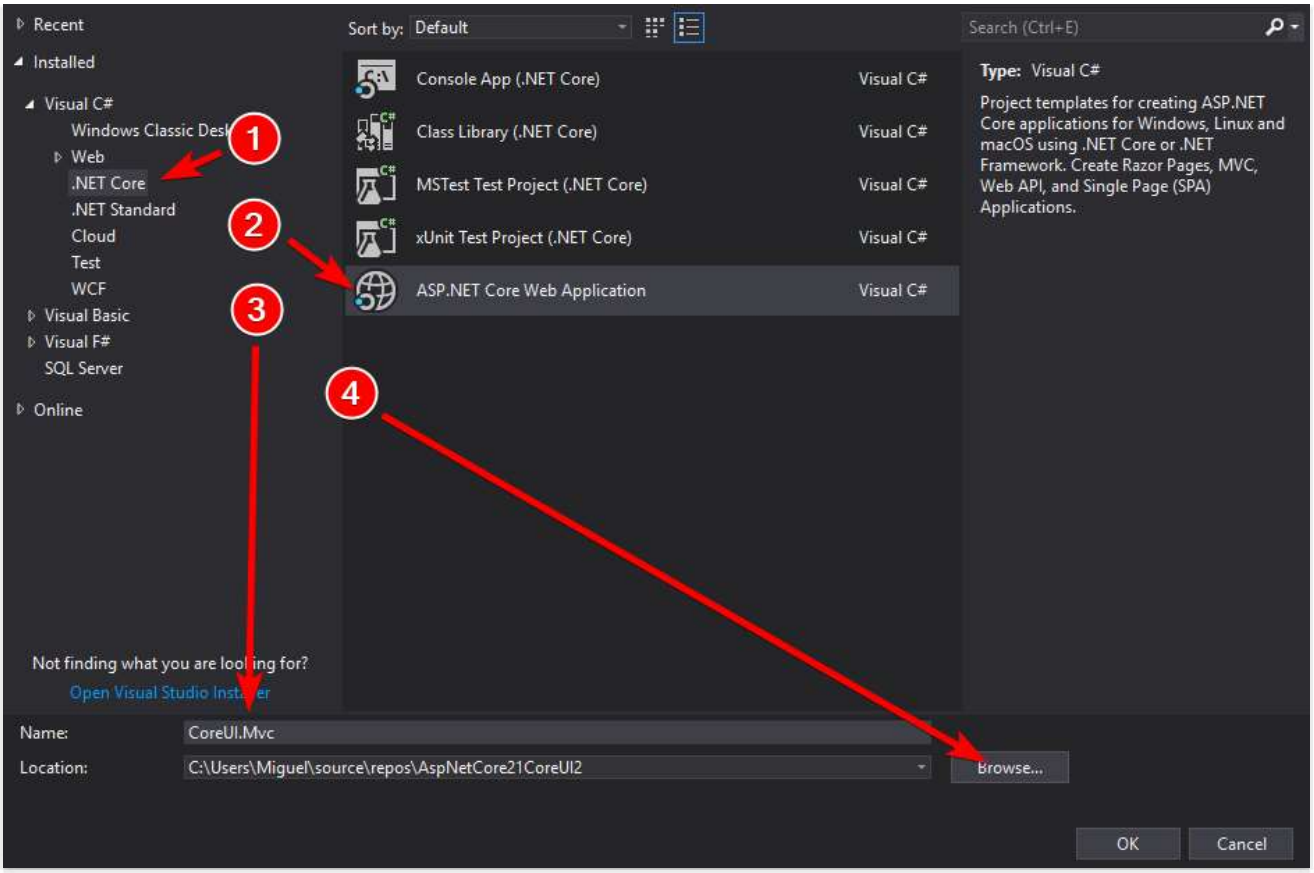
```
8    [*.cshtml]
     indent_size = 2
```

Right now your solution should look like this:

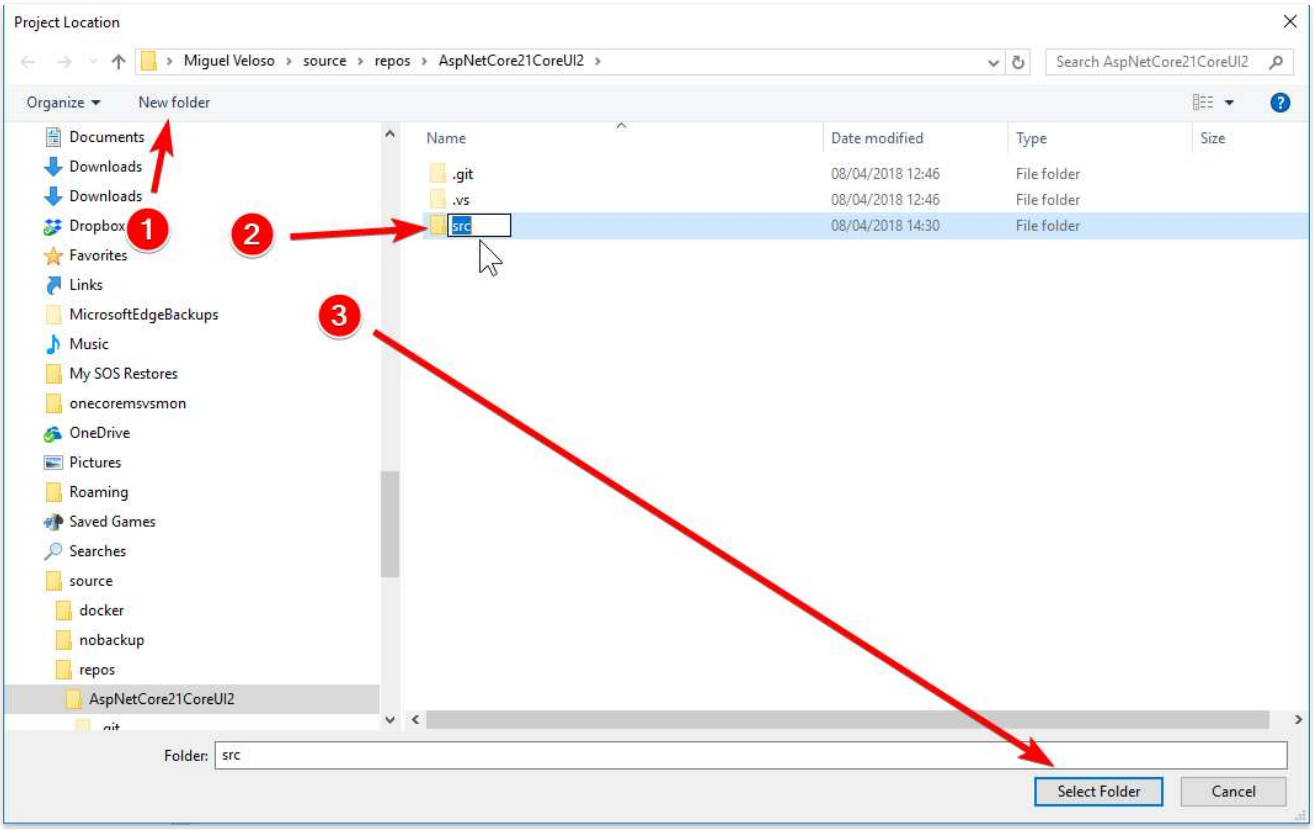


1.2 - Add an ASP.NET MVC Core 2.1 project

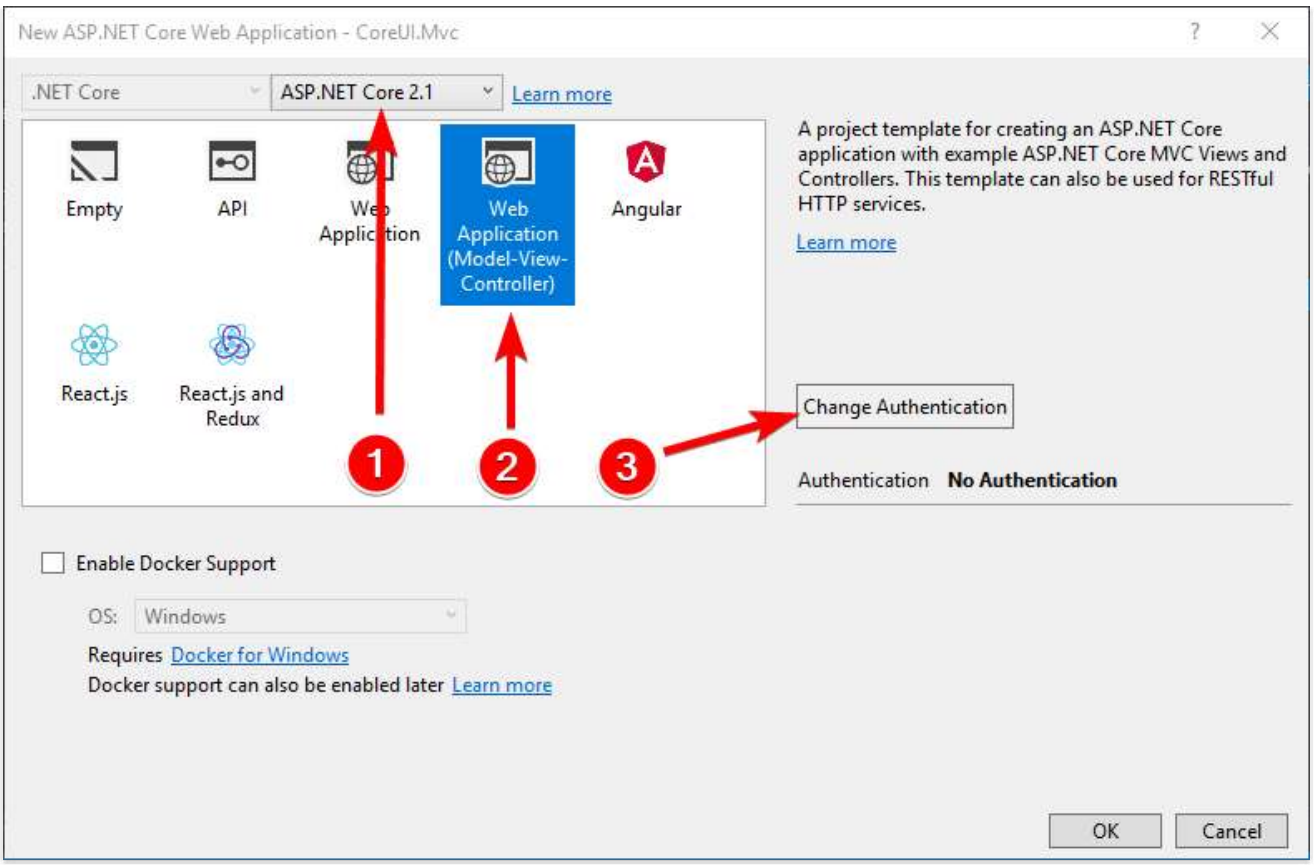
1. Create the **CoreUI.Mvc** project of type **ASP.NET Core Web Application** in the **src** solution folder.



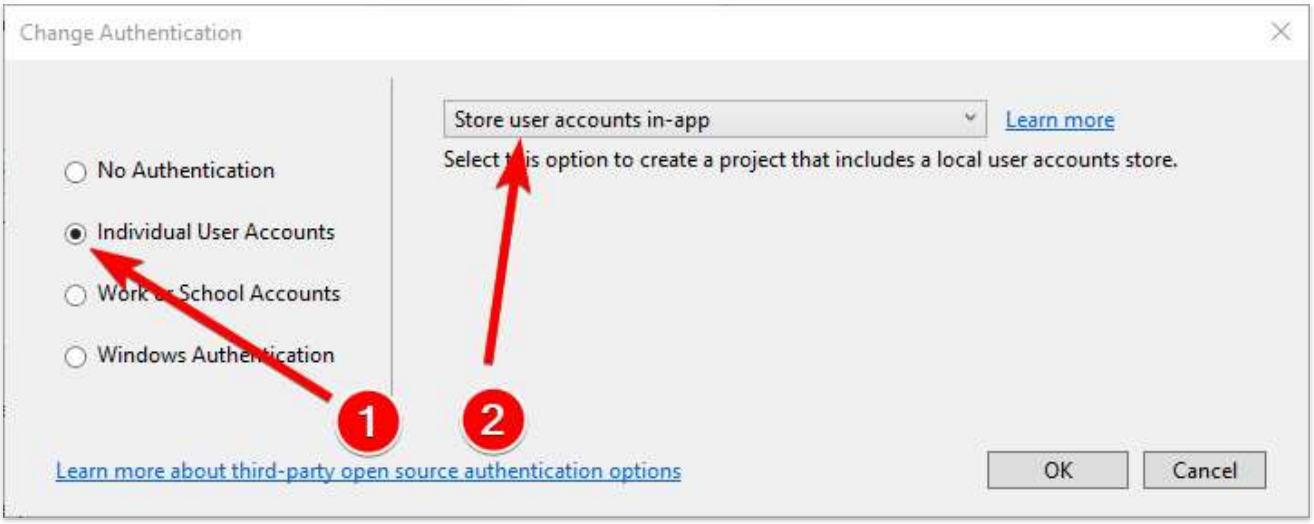
and upon browsing for the folder, create the **src** folder in the file system:



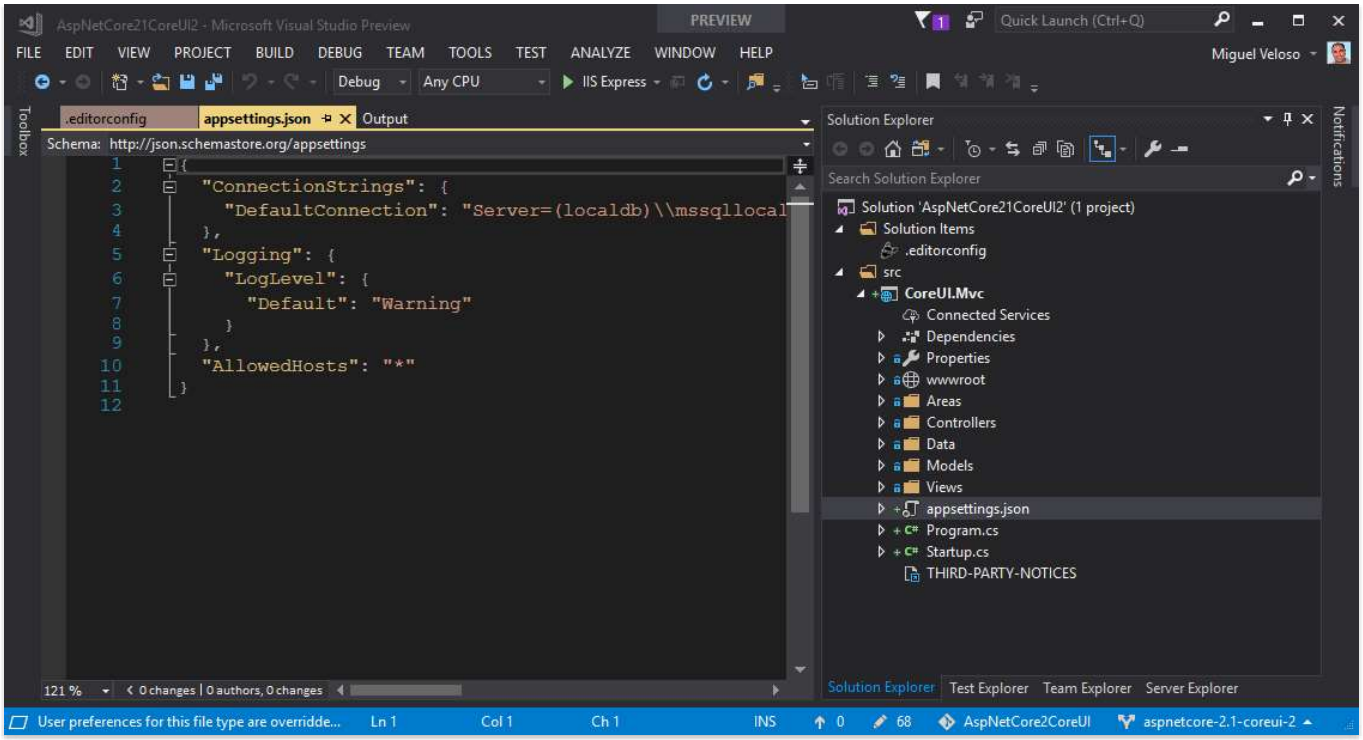
2. Select an **ASP.NET Core 2.1 MVC** type application and



3. Change authentication to **Individual User Accounts** and **Store user accounts in-app**



Right now your solution should look like this:



1.3 - Create the database

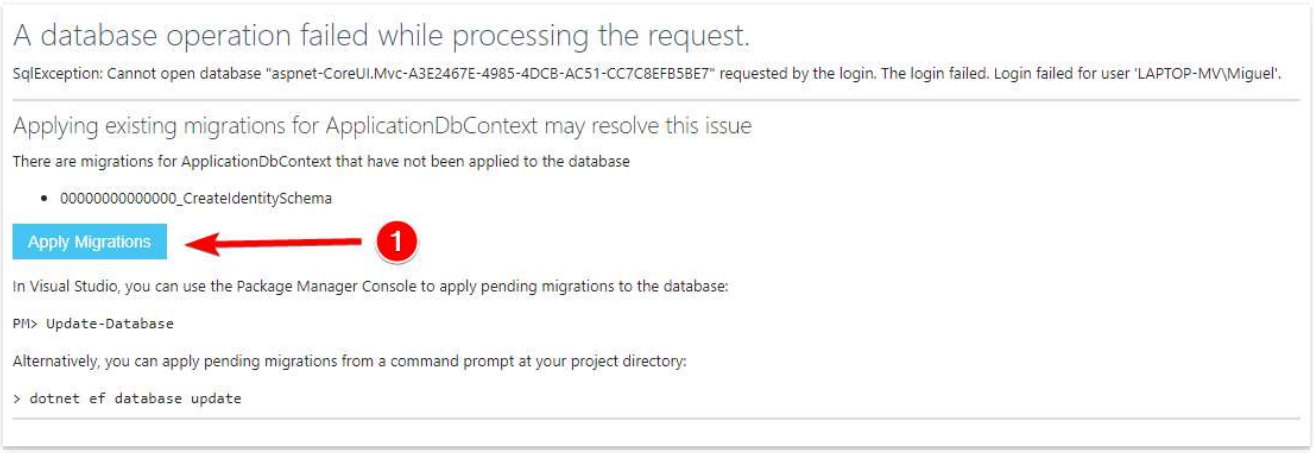
1. Change the connection string in `appsettings.json` file to set a nicer database name.

```
Server=(localdb)\\mssqllocaldb; Database=CoreUI.Mvc; Trusted_Connection=True; Mu
```

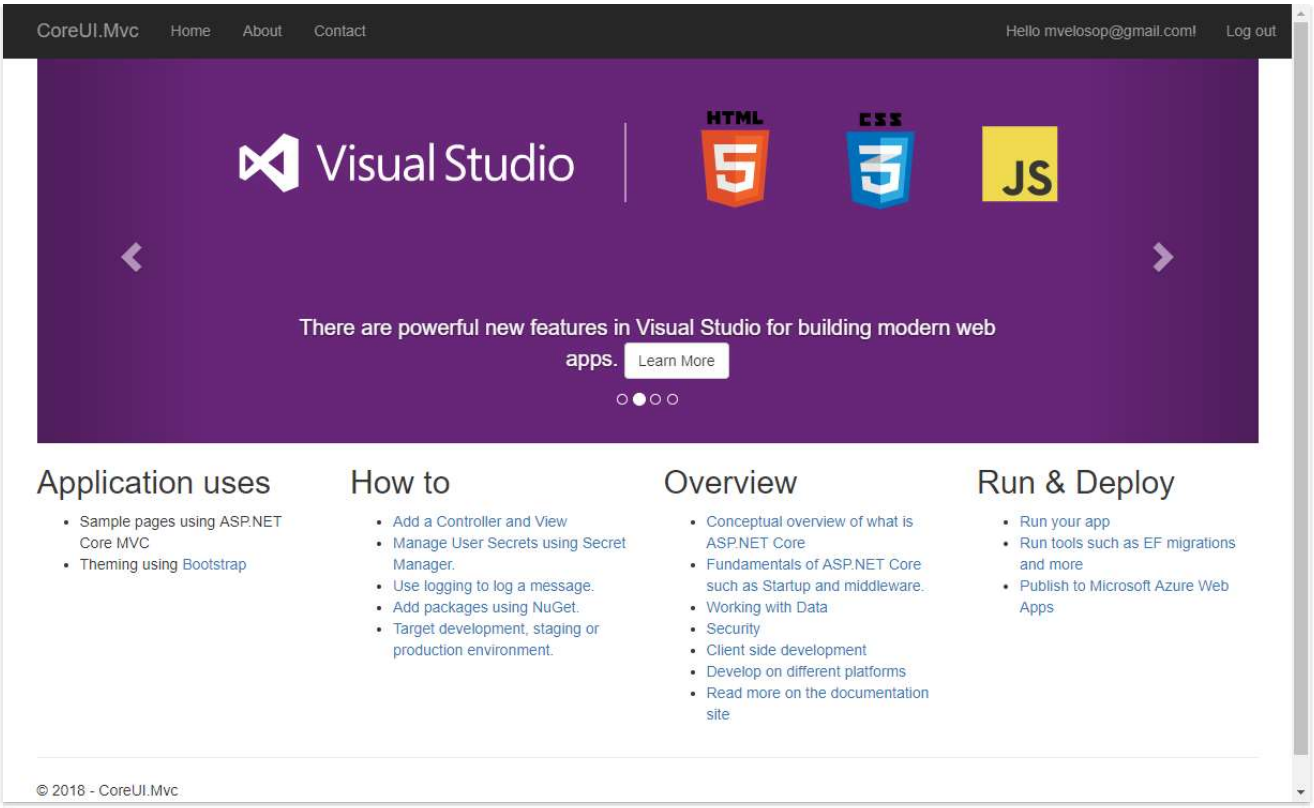
2. Run the application using [Ctrl]+[F5]

3. Sign up to force database creation

4. Click **Apply Migrations**, when you get the database missing error:



5. Refresh the screen (re-posting registration) when the database creation process is complete, to finish user registration



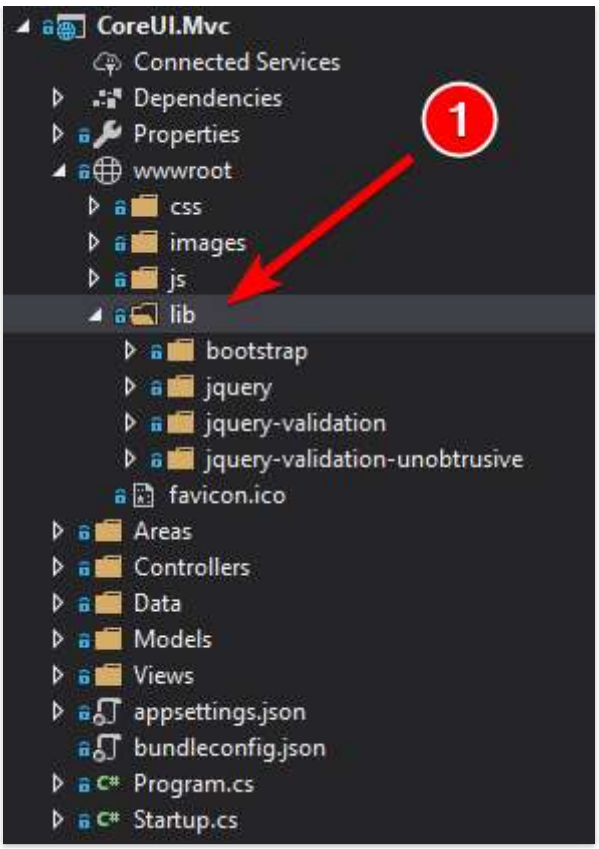
This is a good time to save the project in your repo!

1.4 - Delete original client side libraries

The **wwwroot** folder is the root of the application's client-side, so all client-side static files should be within this folder tree.

We're now going to remove all the client-side libraries included by the Visual Studio template, because we're going to use CoreUI's ones.

1. Open the wwwroot folder and make a note of the libraries used:



The bootstrap folder contains all of the "look" and general "feel" of the application, but this is what's going to be replaced by CoreUI, so we'll just ignore this folder completely.

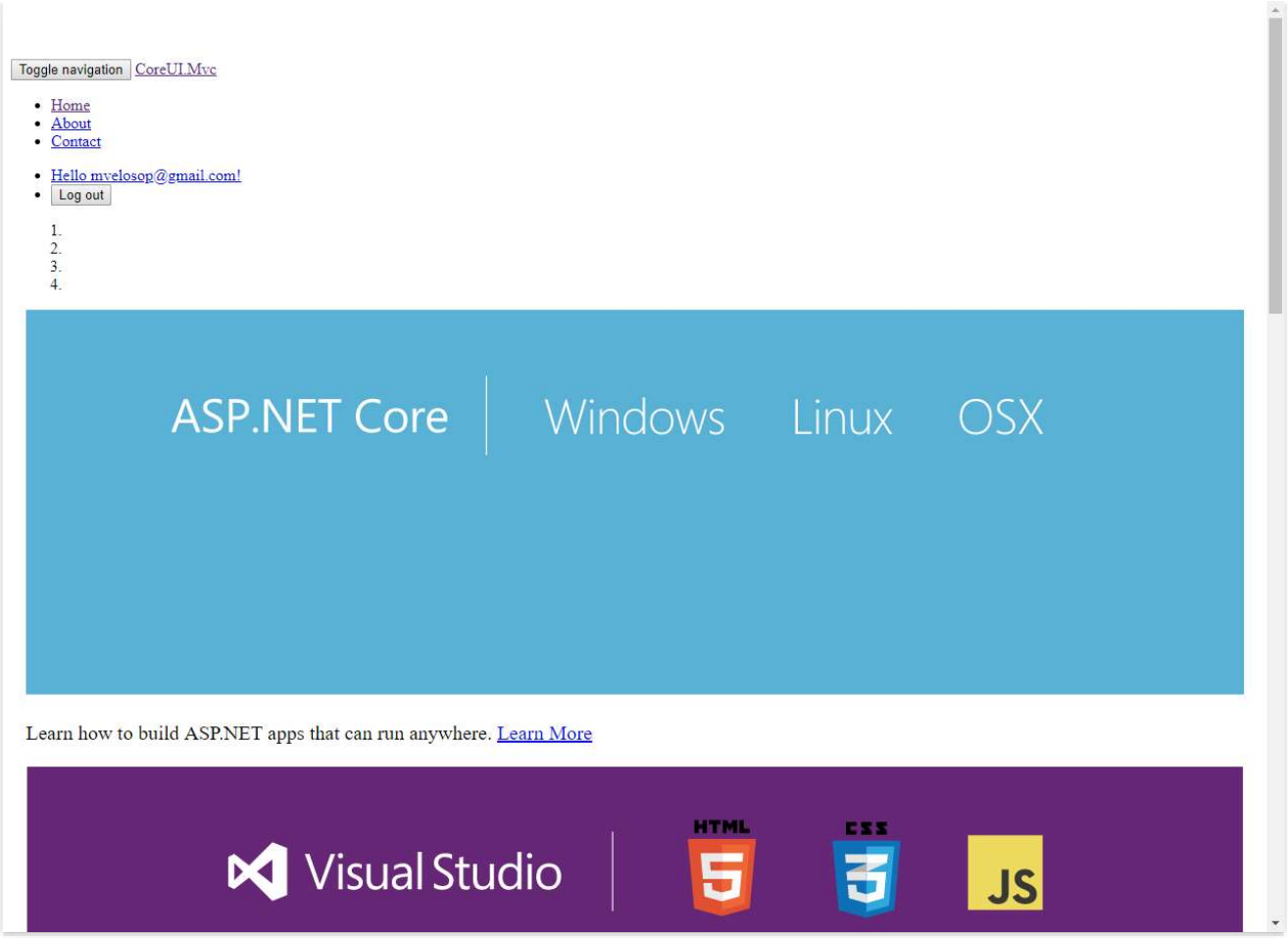
CoreUI is based on Bootstrap, but creates an adapted **style.css** compiling Bootstrap's **.scss** files.

The **jquery*** libraries are used for client side interactivity and validation, so we will later add them to the MVC CoreUI app, but we'll just take a note of the libraries used, we can check the versions in the **.bower.json** file in each folder, in this case:

- jquery (3.3.1)
- jquery-validation (1.17.0)
- jquery-validation-unobtrusive (3.2.9)

2. Delete the **wwwroot\lib** folder.
   (If you can't delete the **lib** folder, you might need to close VS and do it from the file explorer.)

3. Run the application with [Ctrl]+[F5] to display it without any style (or Javascript)



Let's save this version to the repository now

2 - Prepare the CoreUI deployment site

We're now going to prepare a deployment (distribution) site from the latest version of CoreUI, which we'll later copy to our ASP.NET Core MVC application.

2.1 - Prepare your base CoreUI repository

At this point you have two options:

1. Clone the master CoreUI repository in GitHub to learn the process in detail or
2. Clone my ASP.NET Core MVC 2.1 specific fork of CoreUI repository in GitHub to get results faster.

My fork adds a set of Node (Javascript) programs to generate basic Razor views from CoreUI's static html files, with just one command.

I've already submitted a PR to include this in the master CoreUI repo but, in the mean time, I'm guessing you will prefer the express route, so I'll explain the steps assuming your are cloning my fork of CoreUI.

Anyway in this section (#2) you should get about the same results from both repositories, section #3 is where the big differences will show.

So, let's start by cloning my fork, from the command line execute:

```
://github.com/mvelosop/coreui-free-bootstrap-admin-template mvelosop-coreui-free-bootstrap-a
```

In my fork, the default branch is `aspnetcore` (ASP.NET Core MVC specific), the default branch in the master repo is `master` .

We'll now create a new branch to do our customizations, let's call it **dist**, so:

```
cd ./mvelosop-coreui-free-bootstrap-admin-template
git checkout -b dist aspnetcore
```

2.2 - Create the distribution folder (standard static files)

First we need to install client-side dependencies, so, from the **coreui-free-bootstrap-admin-template** folder, execute this command:

```
npm install
```

That creates the **node_modules** folder with all dependencies, as configured in the **dependencies** collection in the **package.json** file.

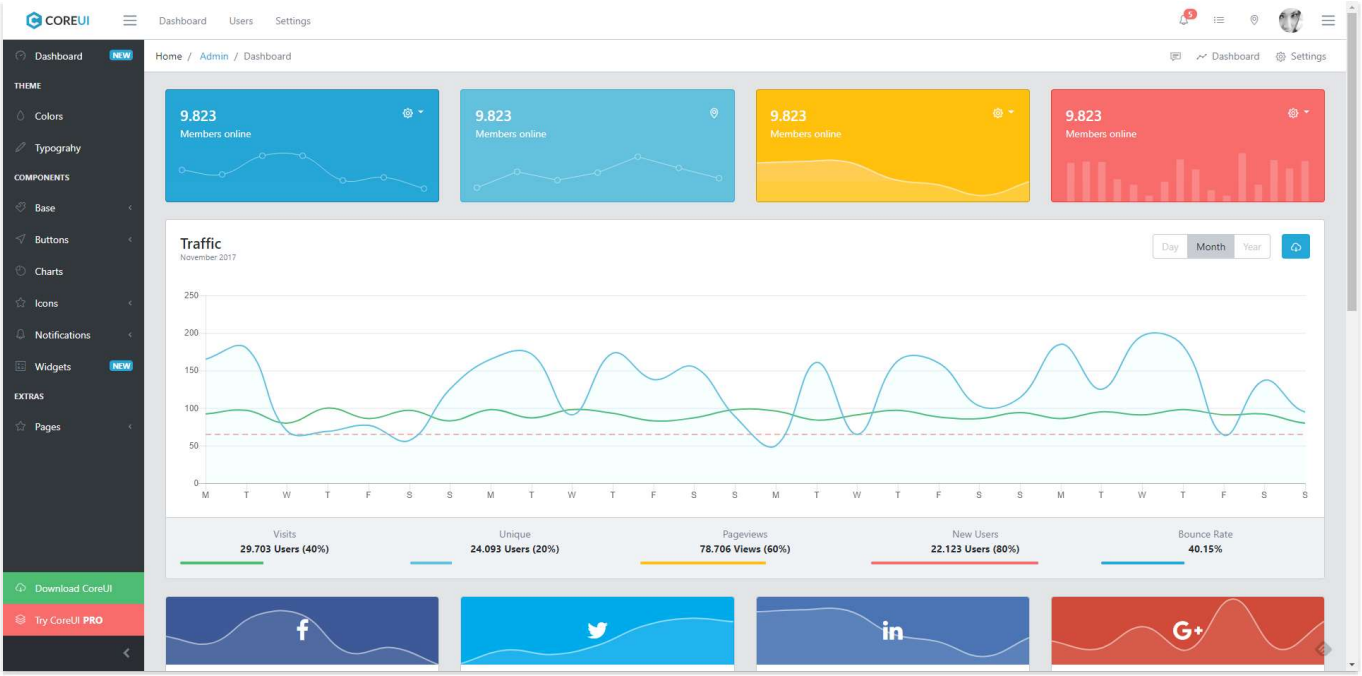
My fork adds some scripts to **package.json**, and required additional packages:

- **build-aspnetcore** : This is the one that makes the magic, and
- **test** : This is the one that makes sure the magic is fine ;-)

Execute this command to verify the site is working properly:

```
npm run serve
```

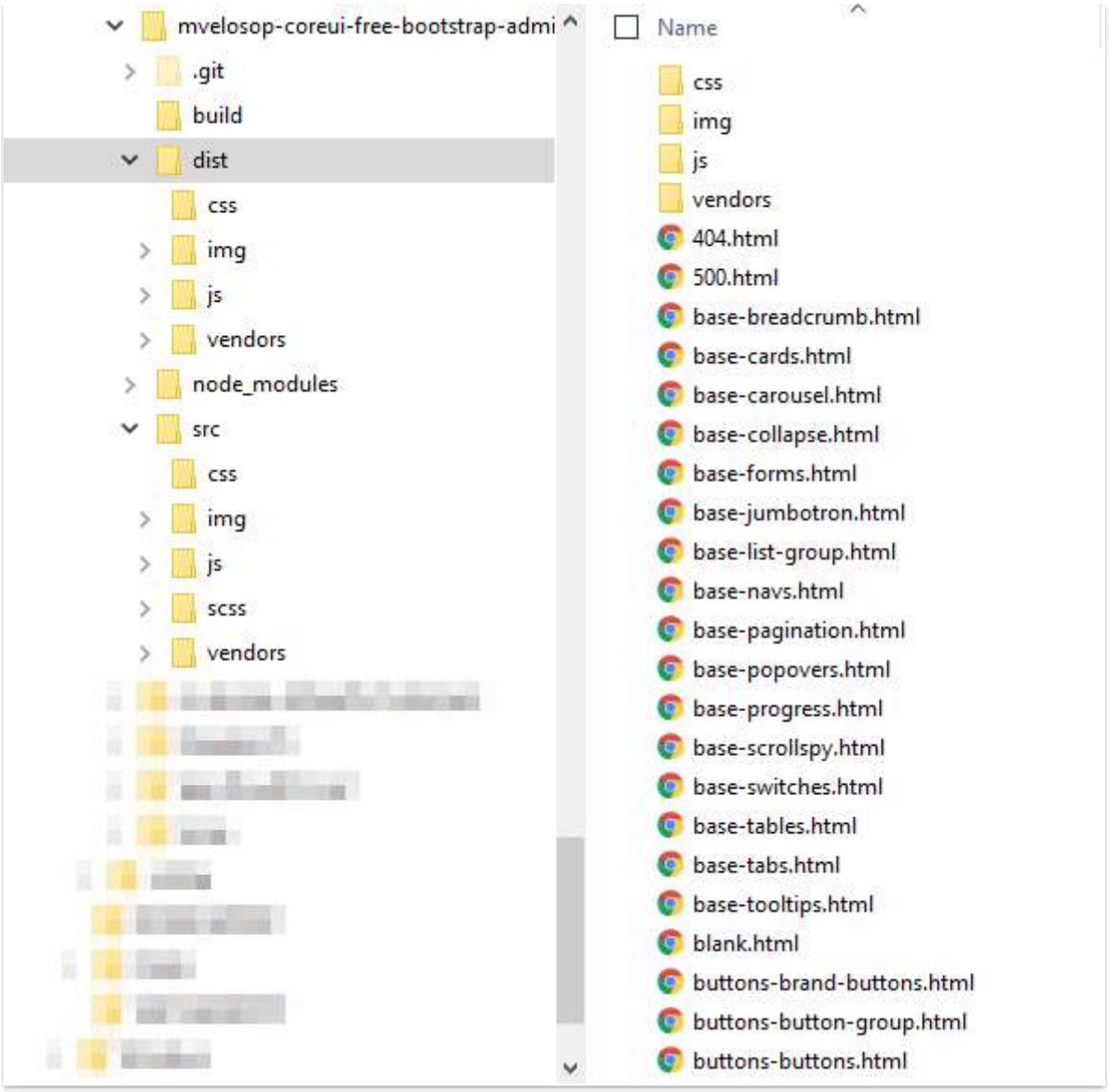
That should open your default browser at http://localhost:3000/ where you should see something like this:



To finally generate the distribution folder we just need to execute the command (after interrupting the node server with [Ctrl]+[C]):

```
1    npm run build
```

That creates the **dist** (.gitignore'd) folder inside the repo folder with something like this:



This is the base static deployment site, ready to be published, and by just double-clicking any html file (except 404, 500, login or register) you can explore the base CoreUI template.

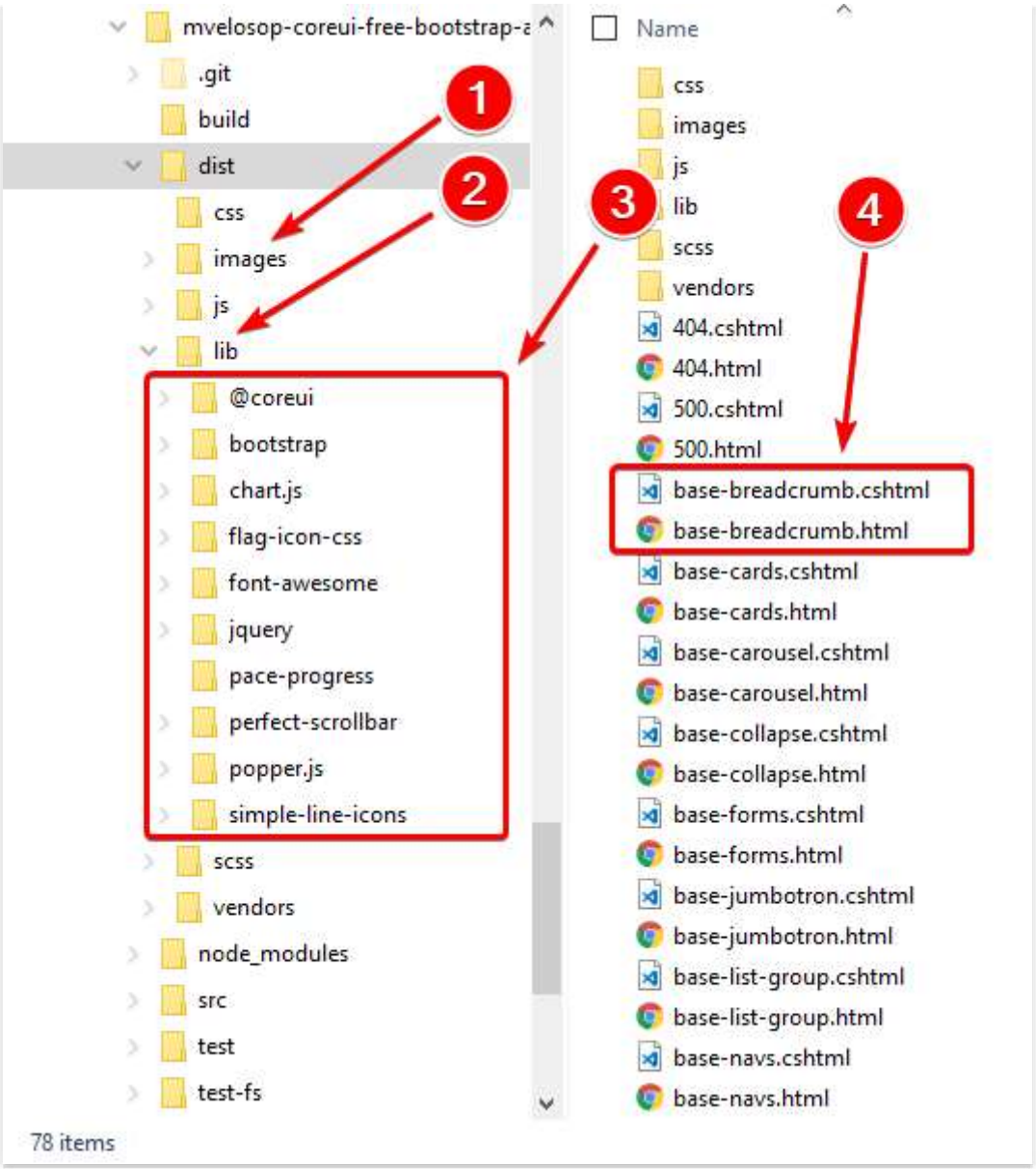
Up to this point, you will get the same result either with the master repo or my fork.

2.3 - Create the distribution folder for ASP.NET Core MVC

Similar to what we just did, to create the distribution folder for ASP.NET Core MVC, **when cloning my fork**, you just need to execute:

```
1    npm run build-aspnetcore
```

This command clears the previous **dist** folder and generates something like this, following the conventions of ASP.NET:

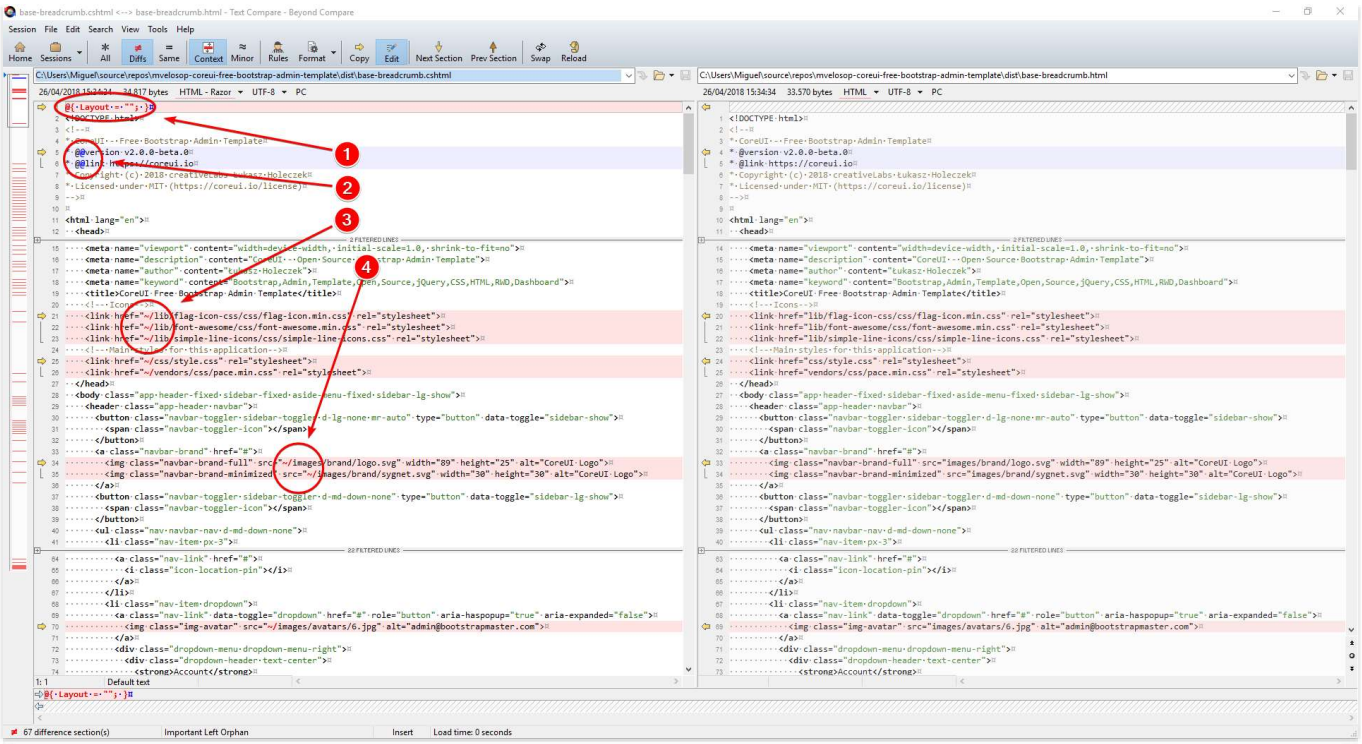


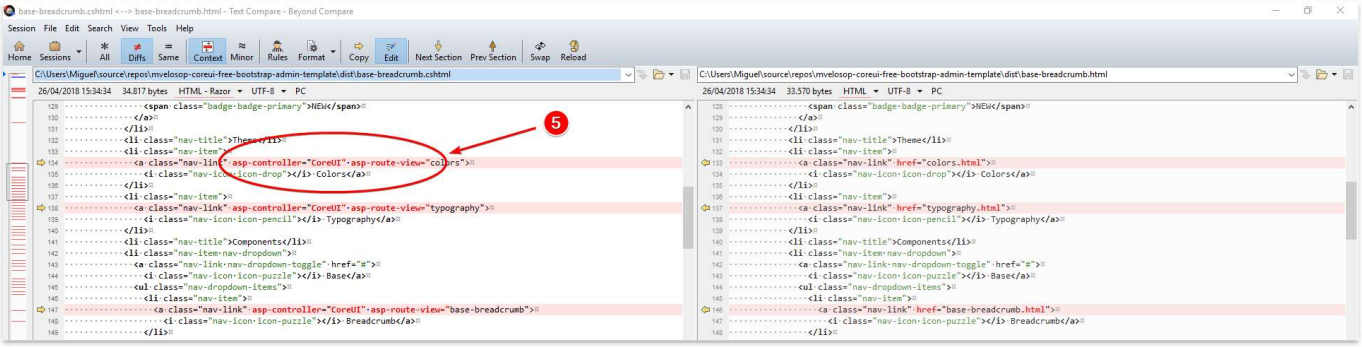
Where we can highlight the following:

1. The images are now in the **images** folder instead of **img**
2. The files from required packages are in the **lib** folder instead of **vendors**
3. The **lib** folder is now organized just like **node_modules** (but including only the files actually referenced, plus any existing and required **.map** file)
4. A Razor view (**.cshtml**) for each static html file of CoreUI.

Just like before, we can double-click any html file (except 404, 500, login or register) to explore the base CoreUI template, with the ASP.NET Core MVC folder structure.

If we open a pair of .html/.cshtml file with Beyond Compare, or a similar diff tool, we can check the differences to understand the changes that the **build-aspnetcore** script does:





1. Clear the default layout because (at this moment) the page has all it needs
2. Escape the `@` character (a Razor reserved symbol)
3. Add the “home” route prefix `~/` for css and js files
4. Add the “home” route prefix `~/` for images and all other static files
5. Change links to html files, to point to ASP.NET controller actions using Razor tag helpers

If you weren’t using the `build-aspnetcore` script from my fork, you’d have to make those changes by hand on each and every html file.

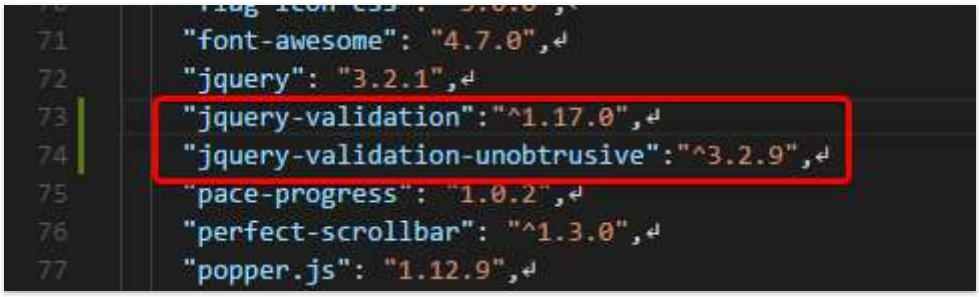
2.4 - Install the dependencies for client-side ASP.NET MVC views

We will now install the dependencies identified in 1.4.

To install the dependencies we just have to add these lines to the **dependencies** collection on **packages.json**, referencing the latest versions of the packages:

```
"jquery":"3.3.1",
"jquery-validation":"1.17.0",
"jquery-validation-unobtrusive":"3.2.9",
```

So the file should result in something like this:



If you prefix a version number with the caret symbol `^`, the minor version and update number might change every time you run the **install** command.

Then, we have to install the new dependencies using:

```
npm install
```

One of the tasks of **build-aspnetcore** is to copy all referenced package files from **node_modules** to **lib**, so we create a file named **vendors.html** (or whatever) to include the files we'll be referencing from the app's Razor views, like this:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Vendor list</title>
</head>

<body style="font-family: Arial, Helvetica, sans-serif; font-size: 18px;">

<h3>Files to deploy to dist/lib</h3>
<ul>
  <li>"node_modules/jquery-validation/dist/jquery.validate.min.js"</li>
  <li>"node_modules/jquery-validation/dist/additional-methods.js"</li>
  <li>"node_modules/jquery-validation/dist/localization/messages_es.js"</li>
  <li>"node_modules/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js
</ul>

</body>

</html>
```

This file will be scanned during the **build/build-aspnetcore** command to select the files that will be copied from **node_modules** to **dist/lib**.

It is important (for the script to work) to include the relative path and filename, from the home folder, including the very same "node_modules" folder, in quotation marks.

The program will also include the related **.map** file (if it exists).

So we can now create/update the distribution folder with:

```
1    npm run build-aspnetcore
```

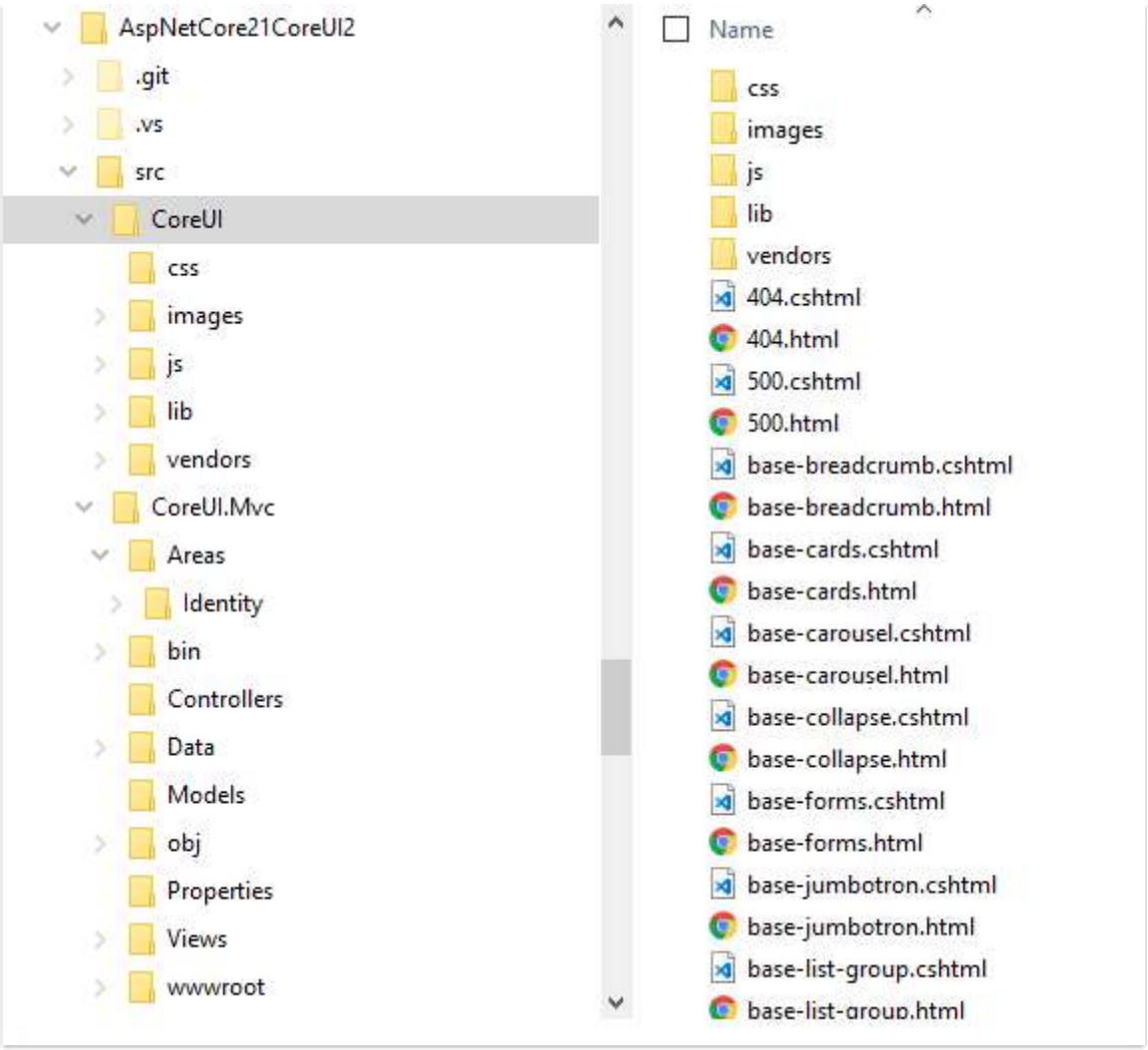
> **Client-side package managers**.
>
> - At some point you might want to use some specialized client-side package manager, like WebPack, but we won't cover that in this post.

2.5 - Copy the distribution folder into the solution

Now we'll just copy the contents of the **dist** folder into the new **src\CoreUI** folder of our solution.

The **src\CoreUI** will be our **reference folder** we'll use to compare to new versions of CoreUI, when they become available, to update the components in our application as needed.

The VS solution should now look like this:



In a moment we will merge that **src\CoreUI** folder to our app's **wwwroot** and **Views** folders.

This is an excellent time to commit to your local repo!

> **ℹ src\CoreUI is not part of the Visual Studio solution.**
>
> 1. Notice that, although src\CoreUI is within the solution's folder structure and under source control, it's not part of the Visual Studio solution, i.e. you will not see it in the solution explorer.

3 - Integrate the CoreUI reference folder into the MVC application

We'll do a basic integration first, just to verify all Razor views work properly:

1. Copy all static files, other than .html, to the **wwwroot** folder
2. Create a generic controller to display all Razor views
3. Copy the **Razor** views to the controller's Views folder

Then we'll rearrange the views using partials to make it more like a standard Razor MVC app.

This is where a tool like Beyond Compare really shines, specially when it's time to update the files to new versions of CoreUI.

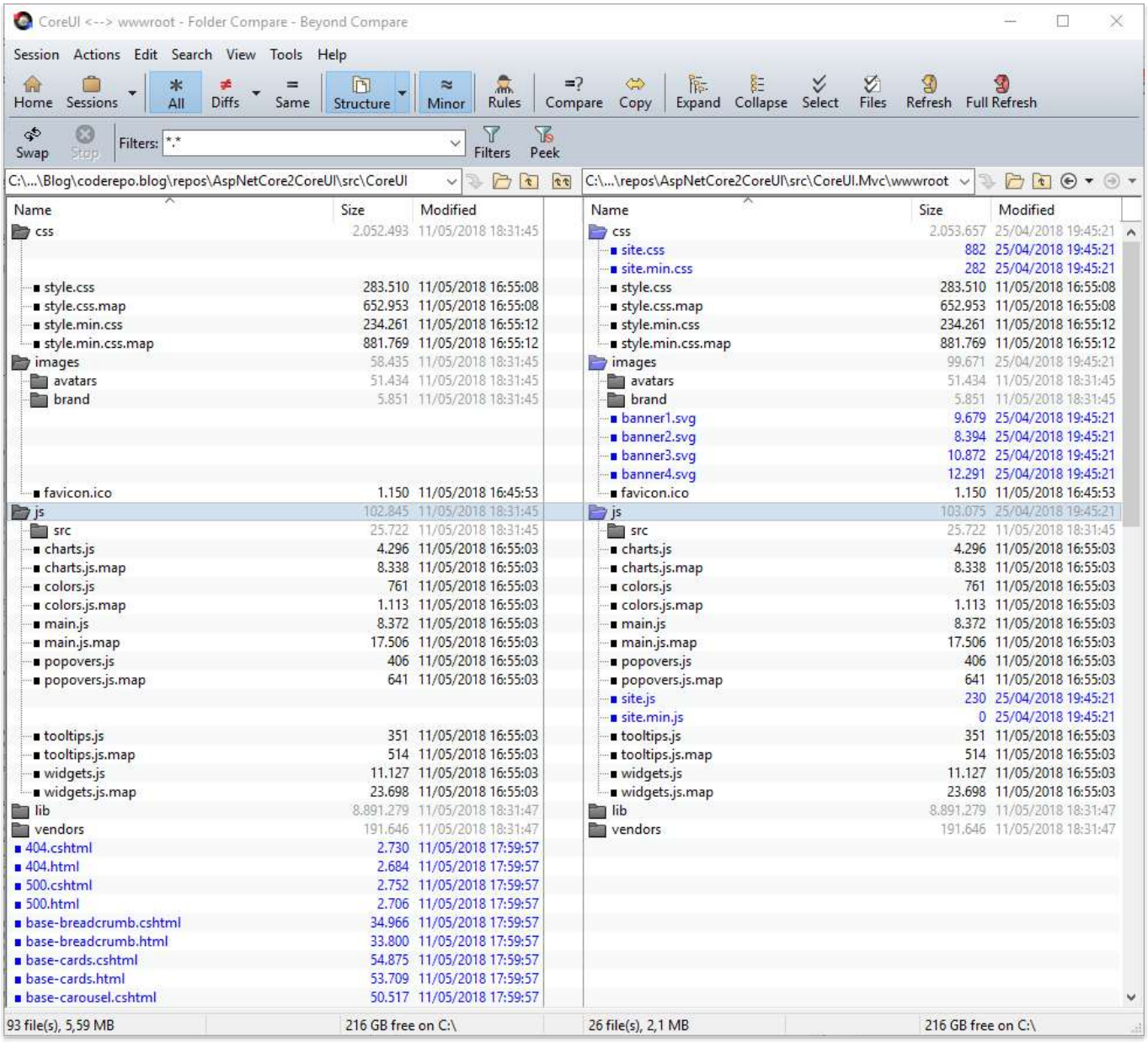
3.1 - Copy static files to wwwroot folder

So we have to copy these folders from **src\CoreUI** to **src\CoreUI.Mvc\wwwroot**:

- css
- images
- js
- vendors

This **vendors** folder actually contains CoreUI specific css files, but I just didn't want to change any of the standard scripts from the template.

If using Beyond Compare, the result should be something like this:



3.2 - Create a generic controller for CoreUI views

Next we'll create a simple controller to display any of the CoreUI Razor views (***.cshtml**), that just receives the name of the view to display.

CoreUIController.cs

[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Controllers\

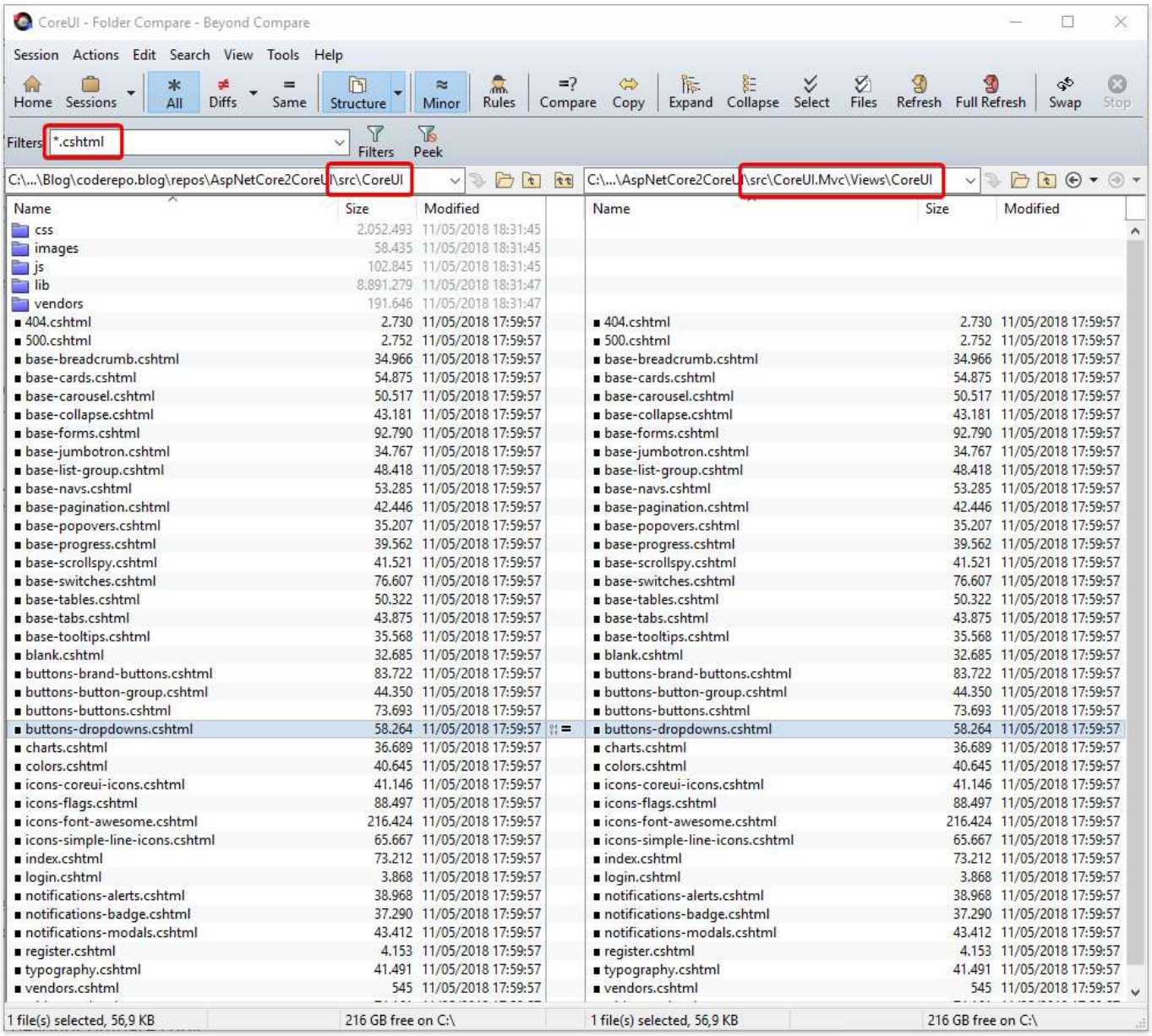
```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using Microsoft.AspNetCore.Mvc;
6
7   namespace CoreUI.Mvc.Controllers
8   {
9       [Route("CoreUI")]
10      public class CoreUIController : Controller
11      {
```

```
12            [Route("{view=Index}")]
13            public IActionResult Index(string view)
14            {
15                ViewData["Title"] = "CoreUI Free Bootstrap Admin Template";
16
17                return View(view);
18            }
19        }
20    }
```

We also need to create the corresponding **Views\CoreUI** folder for the Razor views for this controller.

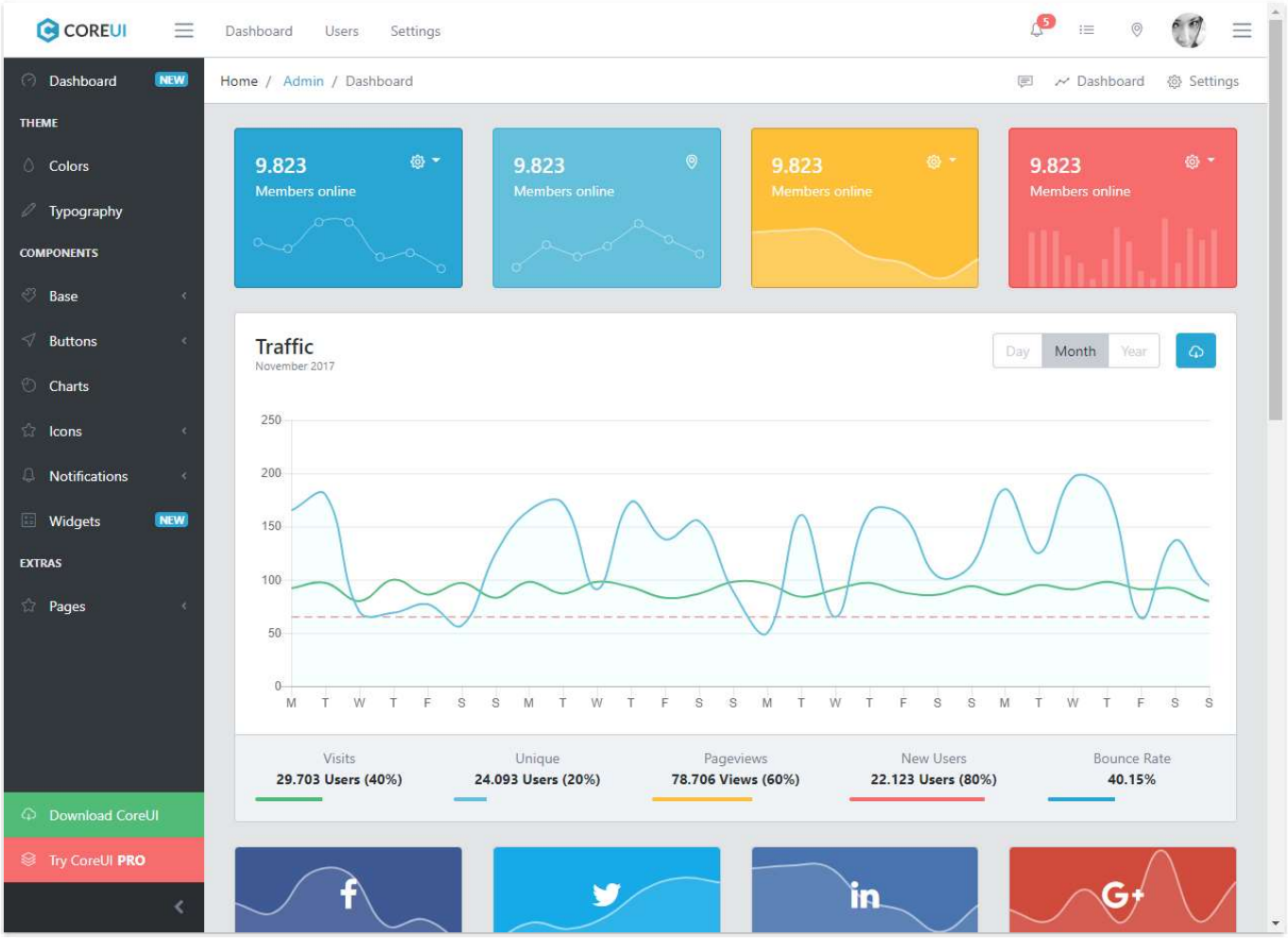
3.3 - Copy the Razor views to the Views\CoreUI folder

Using BeyondCompare copy the generated Razor views from the CoreUI reference folder to the Views\CoreUI folder:



3.4 - Run the app

You should now be able to run the application with [Ctrl]+[F5] and when navigating to https://localhost:#####/CoreUI/Index (##### = port assigned by VS) to get this:



You should be able to navigate to any page of the template and all requests will be handled by the controller, which you can verify in the address bar.

Once again, this is a great time to commit your work.

3.5 - Create a _Layout view

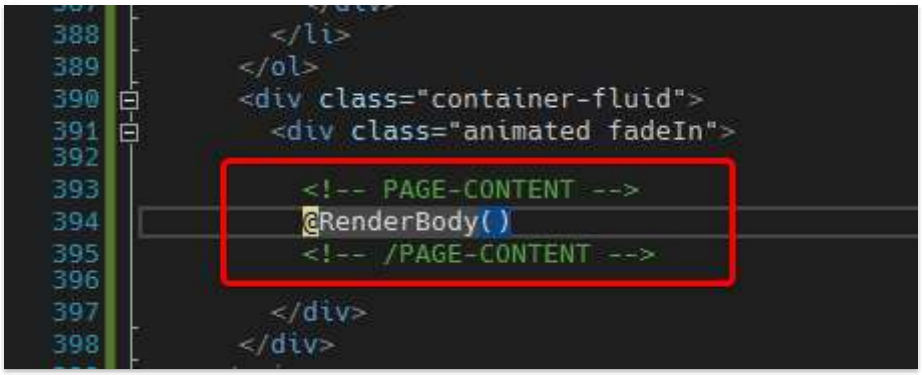
When working in Razor, the rendered page usually has, at least, two parts:

1. The **content**, which is the core of whatever you want to show at any one time, e.g. the Index page, and can be named whatever makes sense, e.g. **Index.cshtml**.
2. The **layout**, that "surrounds" the content and is usually named **_Layout.cshtml**, but is just another Razor view we use differently.

Correlating this to the generated CoreUI Razor views, it's easy to realize that for any view (except 404, 500, login and register):

- The layout is equivalent to **blank.cshtml**
- The content is equivalent to the difference between the view and **blank.cshtml**

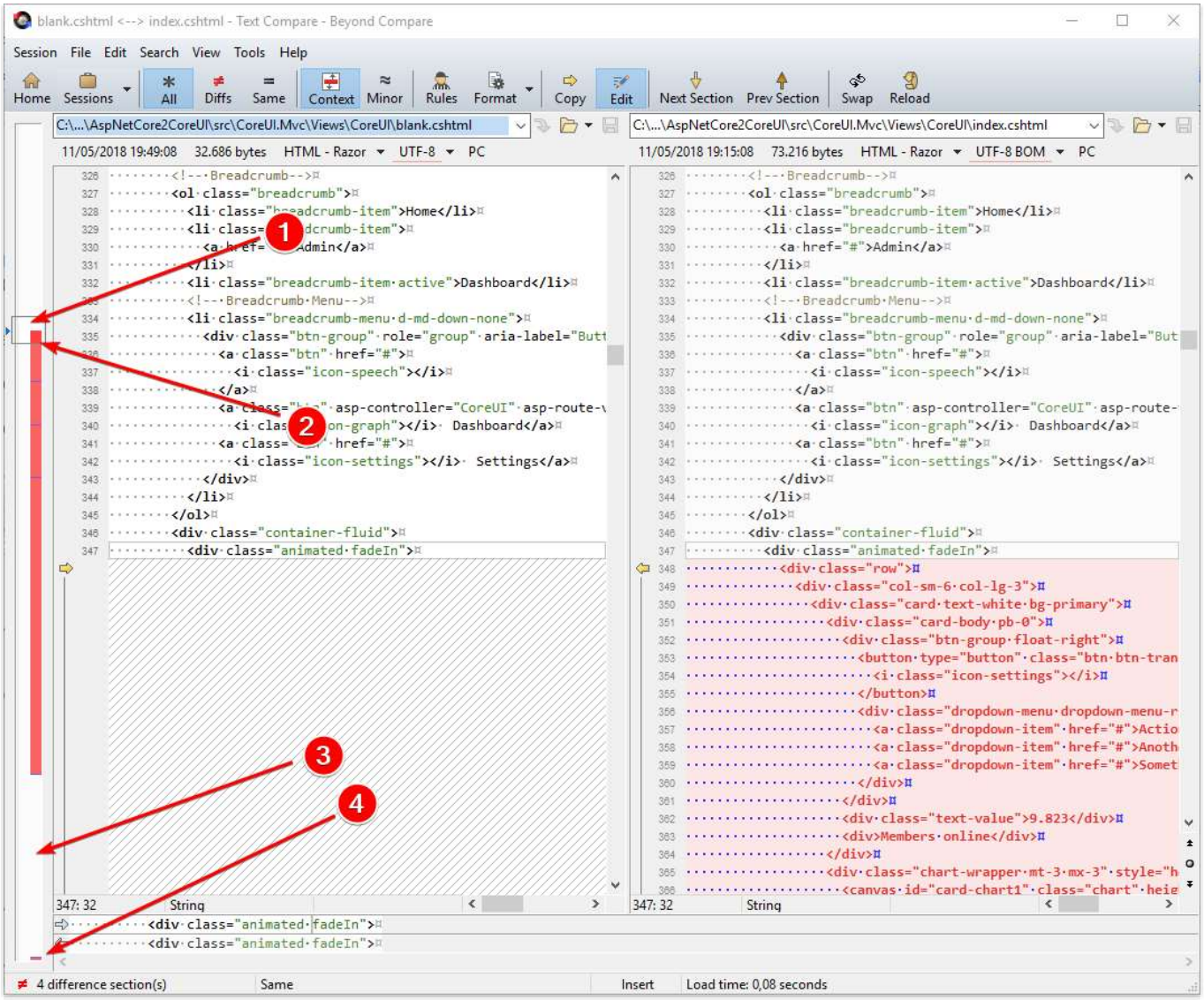
So let's begin by copying **blank.cshtml** to **_Layout.cshtml** and adding the render body helper in **_Layout.cshtml** like so:



In ASP.NET MVC, when we request a view from the controller, the render engine:

1. Finds the requested view
2. Finds the configured layout for the view
3. Renders the layout inserting the view content at the `@RenderBody()` helper

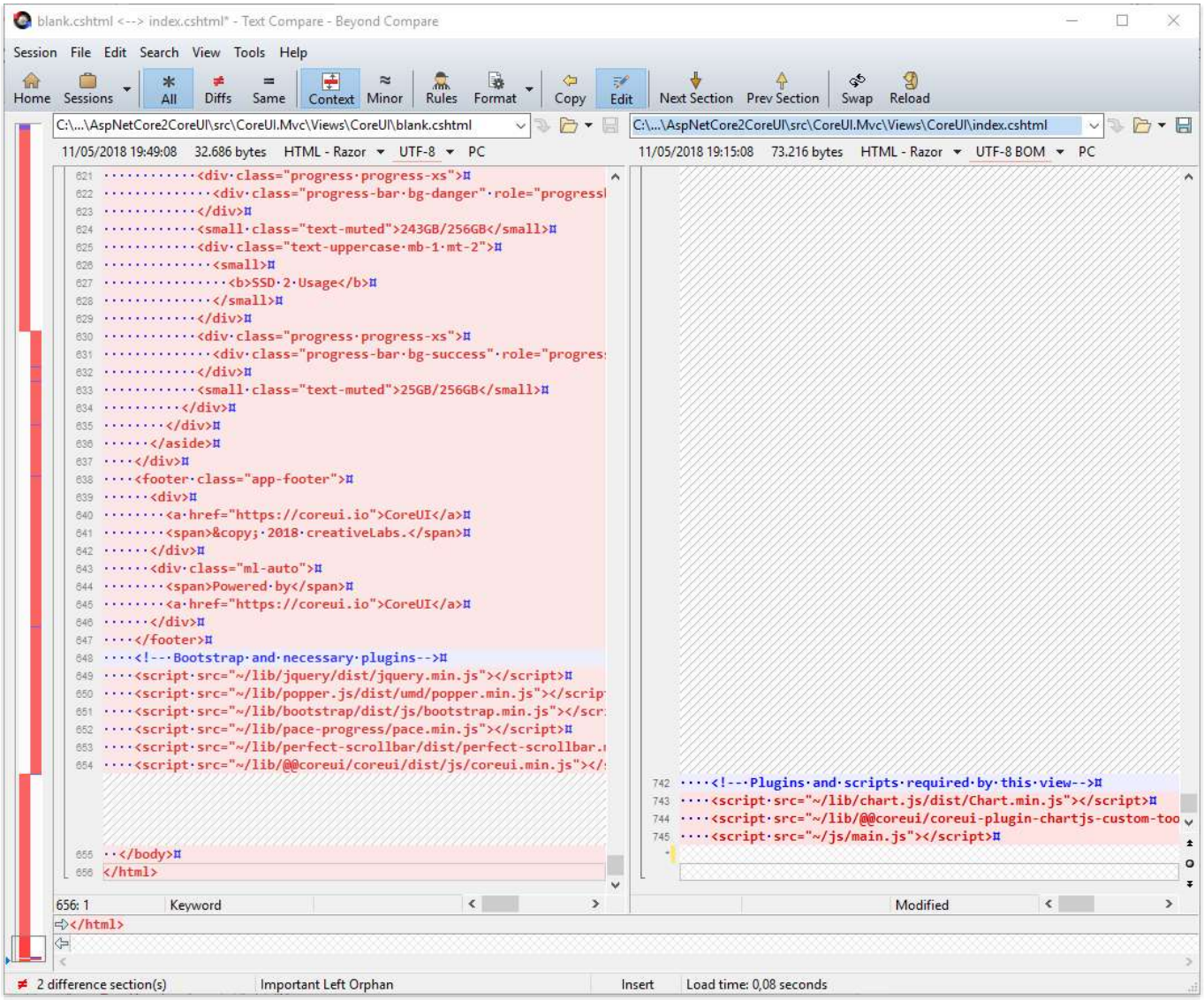
So let's compare now **blank.cshtml** (left) to **index.cshtml** (right), you should get something like this:



On the left thumbnail view we can identify four zones:

1. Before the index content zone both files are identical
2. In the index content zone the **blank.cshtml** has no lines
3. After the index content zone both files are identical again
4. There are a few additional lines at the end of the **index.cshtml** file

So when deleting the common lines from zones **1**, **2** and **the last two lines** on the **index.cshtml** file (on the right side) you should get to this:



Notice the four lines at the end that exist only in **index.cshtml**.

The way to handle that is creating a **Scripts** section in the **index.cshtml** file and adding a call to the **RenderSection** helper in the **_Layout.cshtml**.

After fixing that, the last lines of **Index.cshtml** should be:

```
    </div>
    <!--/.col-->
</div>
<!--/.row-->

@section Scripts {
    <!-- Plugins and scripts required by this view-->
    <script src="~/lib/chart.js/dist/Chart.min.js"></script>
    <script src="~/lib/@coreui/coreui-plugin-chartjs-custom-tooltips/dist/js/custo
    <script src="~/js/main.js"></script>
}
```

And now the last lines of **_Layout.cshtml** should be:

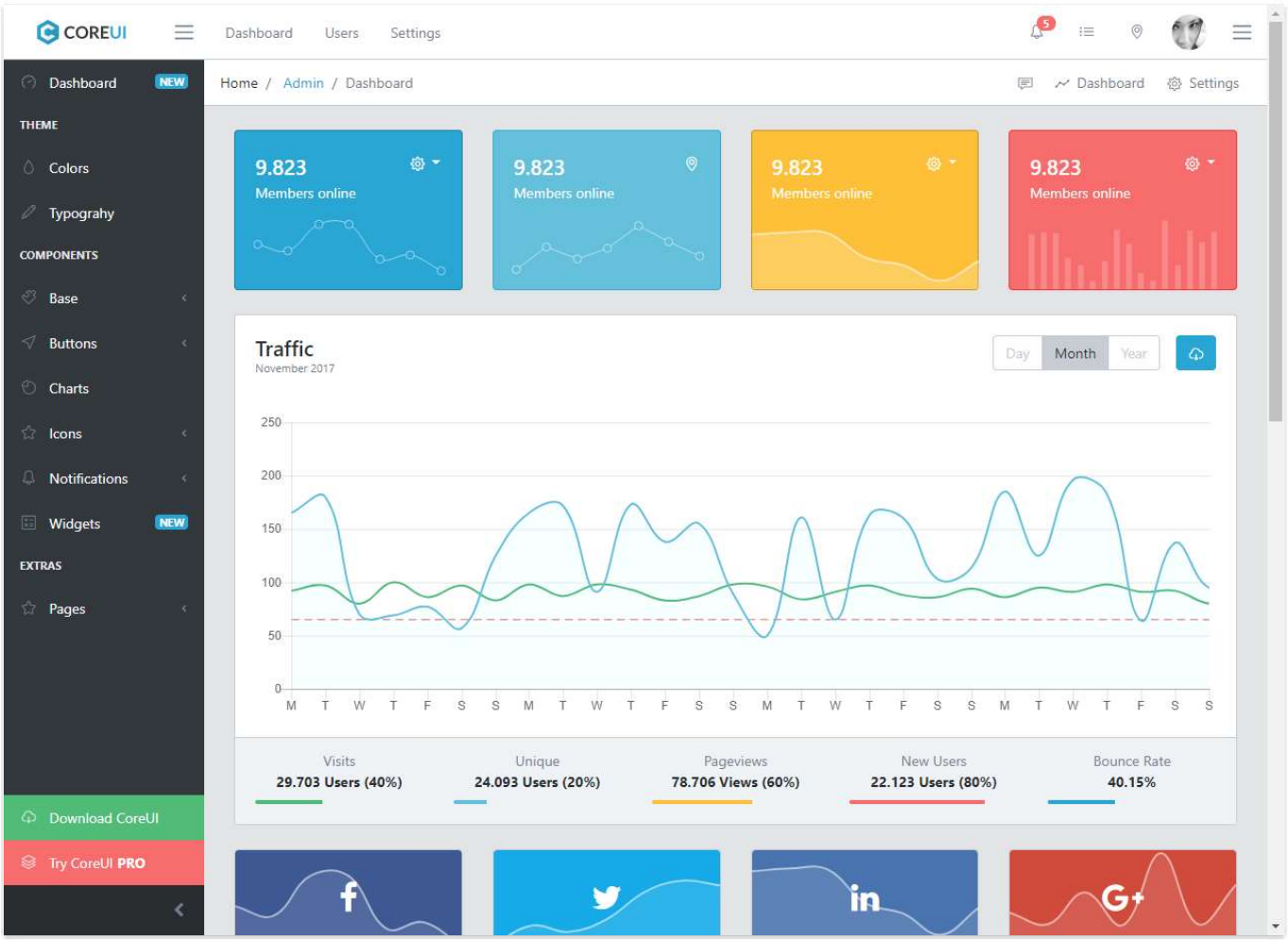
```
    </div>
  </footer>

  <!-- Bootstrap and necessary plugins-->
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/popper.js/dist/umd/popper.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
  <script src="~/lib/pace-progress/pace.min.js"></script>
  <script src="~/lib/perfect-scrollbar/dist/perfect-scrollbar.min.js"></script>
  <script src="~/lib/@coreui/coreui/dist/js/coreui.min.js"></script>

  <!-- Plugins and scripts required by the views-->
  @RenderSection("Scripts", required: false)

</body>
</html>
```

So, if we now run the application again and refresh the https://localhost:#####/CoreUI/Index page, we should get this:



Which looks just like before, only this time we have separated content from layout.

Try commenting out the `@RenderBody()` line we added to **_Layout.cshtml** to check what happens, in case it's not clear at this point.

You can also search for the content marker comments we added to **_Layout.cshtml**.

This is another good time to commit your work

3.6 - Convert the rest of the CoreUI Pages

You can now repeat the process with the rest of the CoreUI pages, except for 404, 500, login and register, that don't have a layout.

There are a couple of additional details, but I will not cover them here, you can take a look at the final views in the repo.

3.7 - Componentize the _Layout view

It's not practical to have a massive 700+ lines _Layout view, so it's best to split it into smaller components.

I will not go through the whole process here, but will only show the final result, including the merge with the default ASP.NET Core MVC **_Layout.cshtml** file, that you can find in the **Views\Shared** folder and you can check all details in the post's repo:

_Layout.cshtml
[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\Shared\

```
<!DOCTYPE html>
<!--
* CoreUI - Free Bootstrap Admin Template
* @@version v2.0.0
* @@link https://coreui.io
* Copyright (c) 2018 creativeLabs Łukasz Holeczek
* Licensed under MIT (https://coreui.io/license)
*
* Adaptated to ASP.NET Core MVC 2.1 by Miguel Veloso
* http://coderepo.blog
-->

<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no
  <meta name="description" content="CoreUI - Open Source Bootstrap Admin Template">
  <meta name="author" content="Łukasz Holeczek">
  <meta name="keyword" content="Bootstrap,Admin,Template,Open,Source,jQuery,CSS,HTML,RWD
  <title>@ViewData["Title"] - CoreUI.Mvc</title>

  <!-- Icons-->
  <link href="~/lib/@@coreui/icons/css/coreui-icons.min.css" rel="stylesheet">
  <link href="~/lib/flag-icon-css/css/flag-icon.min.css" rel="stylesheet">
  <link href="~/lib/font-awesome/css/font-awesome.min.css" rel="stylesheet">
  <link href="~/lib/simple-line-icons/css/simple-line-icons.css" rel="stylesheet">

  <environment include="Development">
    <!-- Template styles -->
    <link href="~/css/style.css" rel="stylesheet">
    <link href="~/vendors/pace-progress/css/pace.css" rel="stylesheet">
```

```
33        <!-- styles tweaks -->
34        <link rel="stylesheet" href="~/css/site.css">
35      </environment>
36
37      <environment exclude="Development">
38        <!-- Template styles -->
39        <link href="~/css/style.min.css" rel="stylesheet" asp-append-version="true">
40        <link href="~/vendors/pace-progress/css/pace.min.css" rel="stylesheet" asp-append-ve
41        <!-- styles tweaks -->
42        <link href="~/css/site.min.css" rel="stylesheet" asp-append-version="true">
43      </environment>
44
45    </head>
46    <body class="app header-fixed sidebar-fixed aside-menu-fixed sidebar-lg-show bg-gray-200
47
48      <!-- APP-HEADER-->
49      <partial name="_app-header" />
50      <!-- /APP-HEADER-->
51
52      <partial name="_CookieConsentPartial" />
53
54      <div class="app-body">
55        <div class="sidebar">
56
57          <!-- SIDEBAR-NAV -->
58          <partial name="_sidebar-nav" />
59          <!-- /SIDEBAR-NAV -->
60
61          <button class="sidebar-minimizer brand-minimizer" type="button"></button>
62        </div>
63        <main class="main">
64
65          <!-- VIEW-BREADCRUMBS -->
66          @RenderSection("Breadcrumbs", required: false)
67          <!-- /VIEW-BREADCRUMBS -->
68
69          <div class="container-fluid">
70
71            <!-- VIEW-STYLES -->
72            @RenderSection("Styles", required: false)
73            <!-- /VIEW-STYLES -->
74
75            <div class="animated fadeIn">
76
77              <!-- VIEW-CONTENT -->
78              @RenderBody()
79              <!-- /VIEW-CONTENT -->
80
81            </div>
82
83            <!-- VIEW-MODALS -->
84            @RenderSection("Modals", required: false)
85            <!-- /VIEW-MODALS -->
86
87          </div>
88        </main>
89
90        <!-- ASIDE-MENU -->
91        <partial name="_aside-menu" />
92        <!-- /ASIDE-MENU -->
93
```

```
    </div>
    <footer class="app-footer">
      <div>
        <a href="https://coreui.io">CoreUI</a>
        <span>&copy; 2018 creativeLabs.</span>
      </div>
      <div class="ml-auto">
        <span>Powered by</span>
        <a href="https://coreui.io">CoreUI</a>
      </div>
    </footer>

    <environment include="Development">
      <script src="~/lib/jquery/dist/jquery.min.js"></script>
    </environment>
    <environment exclude="Development">
      <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
              asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
              asp-fallback-test="window.jQuery"
              crossorigin="anonymous"
              integrity="sha384-K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfHIi9k8z8l9ggpc8X+Ytst4yBo/
  </script>
    </environment>

    <script src="~/lib/popper.js/dist/umd/popper.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
    <script src="~/lib/pace-progress/pace.min.js"></script>
    <script src="~/lib/perfect-scrollbar/dist/perfect-scrollbar.min.js"></script>
    <script src="~/lib/@@coreui/coreui/dist/js/coreui.min.js"></script>

    <environment include="Development">
      <script src="~/js/site.js" asp-append-version="true"></script>
    </environment>
    <environment exclude="Development">
      <script src="~/js/site.min.js" asp-append-version="true"></script>
    </environment>

    <!-- VIEW-SCRIPTS -->
    @RenderSection("Scripts", required: false)
    <!-- /VIEW-SCRIPTS -->

  </body>
  </html>
```

It’s worth noting that the Breadcrumbs are handled as a section in the view, rather than as a partial view, because they are probably tied the view. I case they are not, you should now be able fit it to your needs.

Anyway we’ll view in a moment about an interesting way to handle this using nested layouts.

Then again this is a great time to commit your work

4 - Integrate CoreUI views and MVC application

There are just a few steps missing to finishing the integration with the ASP.NET Core MVC application.

1. Move **_Layout.cshtml** and partials to the **Views\Shared** folder
2. Display the standard MVC views with the new **_Layout.cshtml**
3. Integrate Account views with CoreUI template

Item #1 is sort of trivial, only worth noting that the **_app-header-nav.cshtml** view will now contain the original ASP.NET Core MVC menu:

_app-header-nav.cshtml
[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\Shared\

```
<ul class="nav navbar-nav d-md-down-none">
  <li class="nav-item px-3">
    <a asp-area="" asp-controller="Home" asp-action="Index">Home</a>
  </li>
  <li class="nav-item px-3">
    <a asp-area="" asp-controller="Home" asp-action="About">About</a>
  </li>
  <li class="nav-item px-3">
    <a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a>
  </li>
</ul>
```

You can, of course just set this to whatever best fits your needs.

Also worth noting here, is that item #2 just happens because the views layout is configured to be **_Layout.cshtml** in:

_ViewStart.cshtml
[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\

```
@{
    Layout = "_Layout";
}
```

But we don't want (just for fun) to display the breadcrumbs view in the standard ASP.NET MVC views.

We will solve this using "nested layouts" to check a nice ASP.NET MVC feature.

4.1 - Applying nested layouts for breadcrumbs

When splitting up the layout it became evident that needed something special to handle the breadcrumbs, as they are something that has to adapt to the context, probably for every view.

So it was kind of obvious that breadcrumbs should be a section on the view, but adding it to all CoreUI views was kind of tedious.

So an interesting solution for this was adding a local layout on the **Views\CoreUI** folder to include the breadcrumbs and separate the breadcrumbs items and menu, like this:

_CoreUILayout.cshtml

[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\CoreUI\

```
@{ Layout = "_Layout"; }

@* view-styles *@
@section Styles {
  @RenderSection("Styles", required: false)
}

@*view-breadcrumbs*@
@section Breadcrumbs {
  <ol class="breadcrumb">

    <!-- BREADCRUMB-ITEMS -->
    <partial name="_breadcrumb-items" />
    <!-- /BREADCRUMB-ITEMS -->

    <!-- BREADCRUMB-MENU -->
    <partial name="_breadcrumb-menu" />
    <!-- /BREADCRUMB-MENU -->

  </ol>
}

@RenderBody()

@* view-modals *@
@section Modals {
  @RenderSection("Modals", required: false)
}

@* view-scripts *@
@section Scripts {
  @RenderSection("Scripts", required: false)
}
```

To configure the layout for all CoreUI views, we just need to add this file in the **Views\CoreUI** folder:

_ViewStart.cshtml

[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\CoreUI\

```
@{ Layout = "_CoreUILayout"; }
```

So this is how these parts work this together in the **Views\CoreUI** folder:

1. The view content is rendered at **@RenderBody()** on the **_CoreUILayout** layout.

2. The **Scripts** section is rendered at line 32 on **_CoreUILayout.cshtml**
3. But, as line 32 is within another **Scripts** section, it is then rendered at line 133 on **_Layout.cshtml**
4. Because **_Layout.cshtml** is the layout for the **_CoreUILayout.cshtml**, as configured on its first line.

So, this way, any view can define a **Breadcrumbs** section, just like lines 9-21 in **_CoreUILayout.cshtml**, and it could be:

1. A local partial view (e.g. **_breadcrumb-items**)
2. A shared one (e.g. **_breadcrumb-menu**)
3. Just the required html markup right there or
4. No breadcrumbs at all, as for the initial Razor views created by VS.

One final detail, to properly handle the breadcrumbs this way, it's necessary to add a little tweak to the css in **src\CoreUI.Mvc\wwwroot\css\site.css**:

```
1   .container-fluid > .animated {
2       margin-top: 24px !important;
3   }
```

So, not surprisingly, this is the resulting **blank.cshtml** view, noting all sections shown there are just markers and optional, i.e. can be omitted completely.

blank.cshtml
[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Views\CoreUI\

```
1   @* view-styles *@
2   @section Styles {
3   }
4
5   @*view-breadcrumbs*@
6   @section Breadcrumbs {
7   }
8
9   @* view-content *@
10
11  @* view-modals *@
12  @section Modals {
13  }
14
15  @* view-scripts *@
16  @section Scripts {
17  }
```

If we now go back to https://localhost:#####/CoreUI/Index, we'll see the same view, but this time as a composition of the main content on the layout and the partial views.

And now with the MVC menu on the top bar and the breadcrumbs only in the CoreUI views.

4.2 - Integrate Account views from Identity

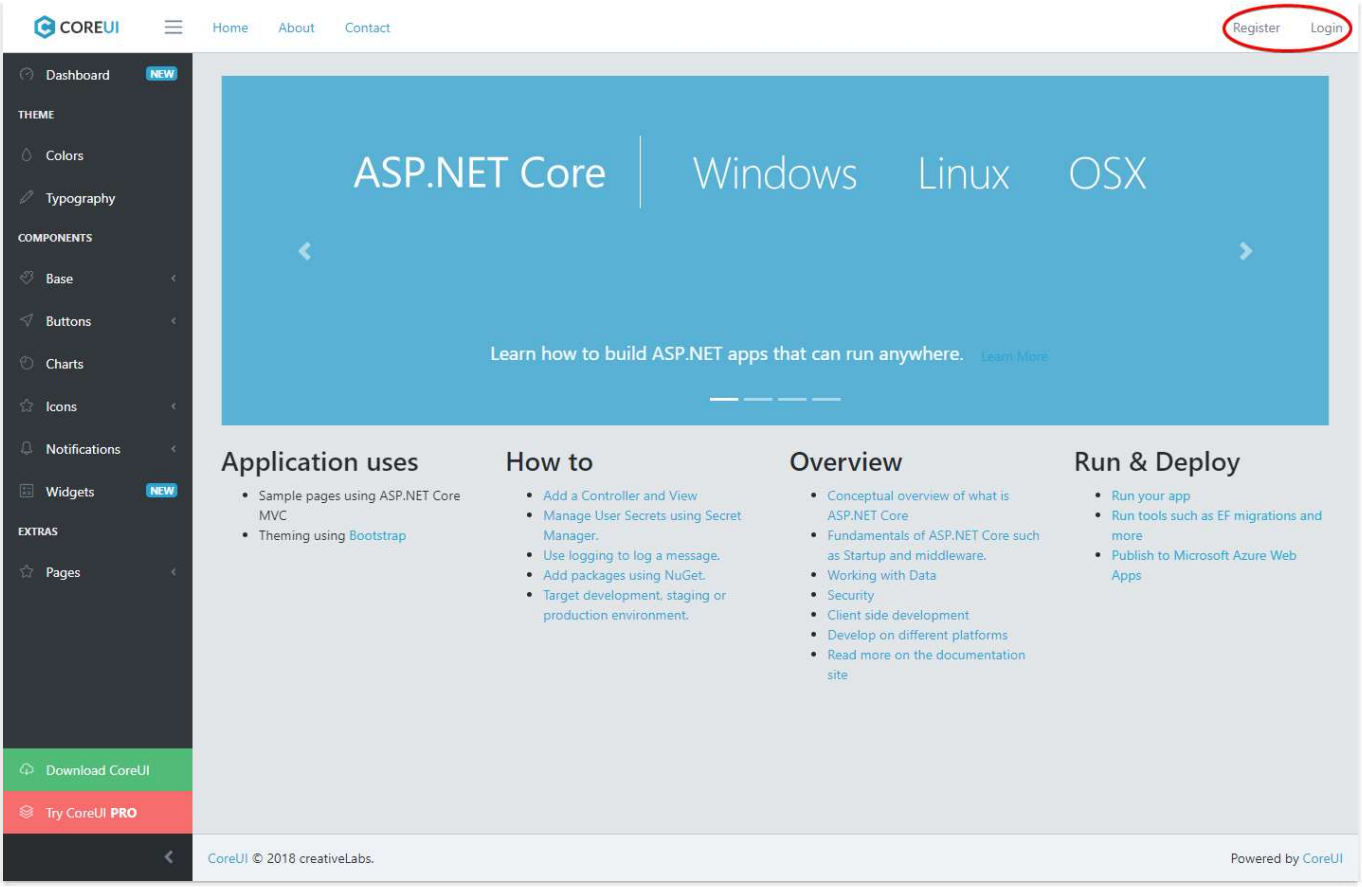
> **ℹ Customizing Identity 2.1 UI views**.
>
> - Starting with ASP.NET Core 2.1 there is new feature that allows including Razor views in class libraries, so the views are no longer generated, so you have to follow this procedure to customize them in case you need it.

Just one more step to finish, so the standard Identity views integrate nicely with CoreUI.

I will not get into the details here, but the "high" level steps, using **_LoginPartial.cshtml** as a guide, are:

1. Show all user related elements in the **_app-header.cshtml** partial only when the user is logged in and
2. Add link to Profile and Logout options in **_user-nav.cshtlm** partial and, finally
3. Delete **_LoginPartial.cshtml**

So the final result, when the user is not logged in should like this:



4.3 - Customize Identity views

From VS 2017 v15.7 and later, ASP.NET Core MVC project templates use the default Identity views from the **Microsoft.AspNetCore.Identity.UI** package so I had to do a little customization following the instructions in https://blogs.msdn.microsoft.com/webdev/2018/04/12/asp-net-core-2-1-0-preview2-now-available/#user-content-customize-default-identity-ui.

To display the navigation options for the user profile view I had to customize the **Account\Manage\ManageNav** view to apply the correct CoreUI classes like so:

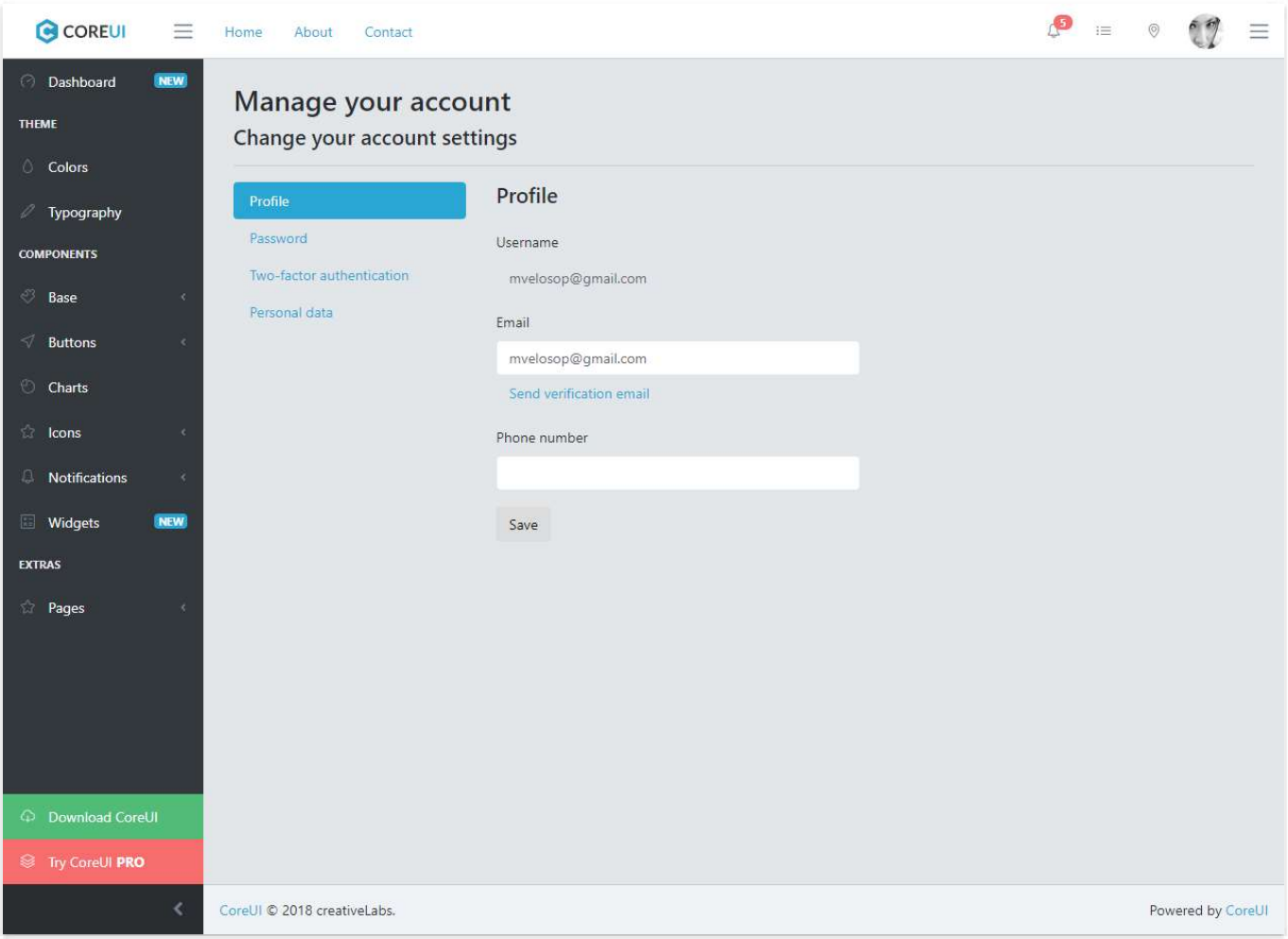
_ManageNav.cshtml

[AspNetCore2CoreUI-2.0] src\CoreUI.Mvc\Areas\Identity\Pages\Account\Manage\

```
1   @inject SignInManager<IdentityUser> SignInManager
2   @{
3       var hasExternalLogins = (await SignInManager.GetExternalAuthenticationSchemesAsync()
4   }
5   <ul class="nav nav-pills flex-column">
6       <li class="nav-item"><a class="nav-link @ManageNavPages.IndexNavClass(ViewContext)"as
7       <li class="nav-item"><a class="nav-link @ManageNavPages.ChangePasswordNavClass(ViewCo
8       @if (hasExternalLogins)
9       {
10          <li class="nav-item"><a class="nav-link @ManageNavPages.ExternalLoginsNavClass(Vi
11      }
12      <li class="nav-item"><a class="nav-link @ManageNavPages.TwoFactorAuthenticationNavCla
13      <li class="nav-item"><a class="nav-link @ManageNavPages.PersonalDataNavClass(ViewCont
14  </ul>
```

Plus some other details you can better check in the repo.

To get a profile view like this:



So, at this point, I think a good starting point for an elegant and attractive user interface for your next project.

Summary

In this article we looked in quite detail at the process for adapting the latest version of CoreUI HTML template to ease the development of attractive ASP.NET Core MVC 2.1 applications.

I hope you've found this post useful and invite you let me know your opinion in the **comments section**.

Thank you,

Miguel.

Follow @mvelosop

Related links

.NET Core 2.1.0-rc1 with SDK 2.1.300-rc1 - x64 SDK Installer
https://download.microsoft.com/download/B/1/9/B19A2F87-F00F-420C-B4B9-A0BA4403F754/dotnet-sdk-2.1.300-rc1-008673-win-x64.exe)

Beyond Compare, from Scooter Software
http://www.scootersoftware.com/

Bootstrap 4.1
https://getbootstrap.com/docs/4.1/getting-started/introduction/

CoreUI 2.0.0 in GitHub
https://github.com/coreui/coreui-free-bootstrap-admin-template/tree/v2.0.0

CoreUI build-aspnetcore PR in GitHub
https://github.com/coreui/coreui-free-bootstrap-admin-template/pull/379

Customize default ASP.NET Core 2.1 Identity UI
https://blogs.msdn.microsoft.com/webdev/2018/04/12/asp-net-core-2-1-0-preview2-now-available/#user-content-customize-default-identity-ui

EditorConfig Language Service for Visual Studio
https://marketplace.visualstudio.com/items?itemName=MadsKristensen.EditorConfig

Node
https://nodejs.org/

npm
https://www.npmjs.com/

Razor tag helpers
https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1

Share

Follow @mvelosop

- **TAGS**

User Experience Client Side Development Bootstrap 4 CoreUI

Previous Article

Construyendo aplicaciones elegantes con ASP.NET Core MVC 2.1 y CoreUI 2 (Bootstrap 4)

May 21, 2018 POSTS

14 Comments **www.coderepo.blog** **1 Giorgos Betsos**

Recommend **Share** **Sort by Best**

Join the discussion…

maxmax • 13 days ago

Very nice piece of work! Thanks for the tutorial.
Wanna check, after i publish the project, im able to host it in my local IIS, the home page able to show. However, when i click any link from navbar, or the hearder bar, all other View()/Page cannot be show.. any idea?

Reply • Share ›

Miguel Veloso Mod maxmax • 11 days ago

Hi maxmax,

I found this: https://docs.microsoft.com/...

I haven't tried it but it looks like it's the way to go.

Hope this helps.

Reply • Share ›

Miguel Veloso Mod maxmax • 13 days ago

Hi maxmax, thanks for the comments.

Would you post the issue in GitHub, please? https://github.com/mvelosop...

Reply • Share ›

jameshoughton • 25 days ago

Hi - great step-by-step tutorial for a newbie to asp.net core (not to dev) until I get to this :-

3.2 - Create a generic controller for CoreUI views
Next we'll create a simple controller to display any of the CoreUI Razor views (*.cshtml), that just receives the name of the view to display.

Can you explain how this is performed?

Thanks.

• Reply • Share ›

Miguel Veloso Mod → jameshoughton • 25 days ago

Hi James, thanks for the comment!

It's actually pretty simple, once you are clear on how MVC "magic" works.

I guess you are used to actions returning just View(), right?

What happens under the covers is that the framework (well, the view renderer), by convention, renders the view with the same name as the action, hence, when you create a view with the proper name, everything just works.

However, there's an overload for the View() method that accepts the name of the view to render, instead of just following the name convention (try hitting [F12] with the cursor over the View() method).

So View() can actually render any view you specify, if you only pass a name, the view renderer searches for that view, first in the controller's view folder, then in the shared folders etc., but you can also specify a complete path, including the extension and that will be the rendered view, so you get the idea.

So what I'm doing here, instead of creating 39 action, is just passing the view name in the route (e.g. /CoureUI/charts) and passing that name to the view renderer. Try passing /CoreUI/charts2 in the browser.

Hope it's clearer now, but just let me know otherwise.

Best.

• Reply • Share ›

James → Miguel Veloso • 25 days ago

ps. It may help if you were to post your whole project to github, that way I (we) could inspect your code to see what's missing/clarify any syntax.

• Reply • Share ›

Miguel Veloso Mod → James • 25 days ago

There it is, in the top of the post, the "Source Code" section ;-)

• Reply • Share ›

James → Miguel Veloso • 25 days ago

Doh! It's getting late! Thank you!

Reply • Share ›

James ➚ Miguel Veloso • 25 days ago

HI Miguel - thank you so much for the quick response, it's massively appreciated. You're making a real difference to my understanding of asp.net core (i've been a dev for 25 years but mostly desktop).

Do I need to actually create a new "Controller" class and if so what code goes inside it?

Sorry to burden you on a Sunday!

Thanks again, your long and hard work on this article is very much appreciated.

James..

Reply • Share ›

Miguel Veloso Mod ➚ James • 25 days ago

Hi James,

Yes, you do have to create a new controller class.

There's a lot of details there, but I suggest you follow this tutorial first: https://docs.microsoft.com/... and then check the MVC fundamentals page: https://docs.microsoft.com/....

If you are coming from the desktop then you'll probably need a bit of http/html fundamentals, bit by bit, because it'll seem overwhelming.

The most important part now is that you understand how data in a form reaches an action, so I'd begin with https://developer.mozilla.o... and then https://developer.mozilla.o....

Bookmark the Mozilla Developer Network (MDN) https://developer.mozilla.org, that's probably THE Internet reference for web developers.

So... you have some homework to do ;-)

Best.

Reply • Share ›

James ➚ Miguel Veloso • 25 days ago

HI - thank you again.

I think I (or Microsoft) have complicated this further but having two different types of Razor pages, asp.net Razor pages and MVC Razor pages. I am using the former and I realise this differs in many ways from the MVC

flavour.

I am very familar with HTML etc so that's not an issue.

Thank you again.
James.

Reply • Share ›

Miguel Veloso Mod → James • 25 days ago

Just consider Razor pages as another tool, like a hammer and a screwdriver, each one has its intended application.
You can probably "drive" a screw with a hammer, but... 😉

1 • Reply • Share ›

Hakan Lindestaf • a month ago

Very nice write-up! Just a small typo at the top, says MAY-2017, should probably be 2018. Thanks! (also would be nice with the comments section in English)

Reply • Share ›

Miguel Veloso Mod → Hakan Lindestaf • a month ago

Hi Hakan, thanks for the comment and the heads-up!

Both issues have been addressed now, thanks!

3 • Reply • Share ›

ALSO ON **WWW.CODEREPO.BLOG**

AUTHOR

Miguel Veloso

Independent Software Consultant
Certified ScrumMaster

Acerca de mi / About me

TAGS

Entity Framework Core (6) Bootstrap 4 (4) CSharp (4) Client Side Development (4) CoreUI (4)

Domion (4) User Experience (4) Architecture (3) Migrations (3) EF Core CLI (2)