

Integrating AdminLTE with ASP.NET Core 3.1 – Detailed

by Mukesh Murugan | Updated on Jul 12, 2020 | ASP.NET Core

In this article, we will learn about Integrating AdminLTE with ASP.NET Core 3.1 MVC or really any other Bootstrap based UI Frameworks **completely from scratch**. We will also go through about integrating Identity Authentication to our MVC Application. Also, you will gain quite a lot of practical knowledge on Views, Layouts, Partial Views, Conditional Rendering, Navigation Indicator and much more.

You can find the source code of this demonstration at my [Github](#).

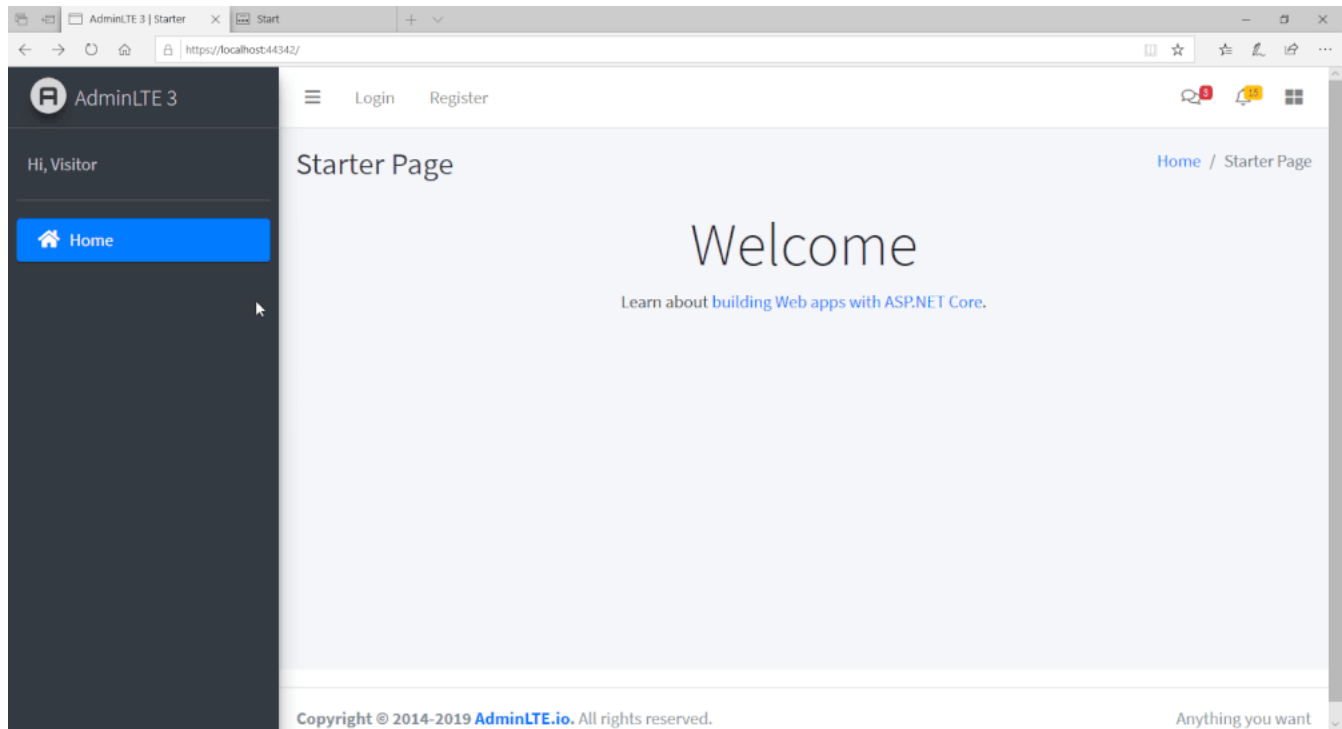
Disclaimer : This is a pretty huge article with around 3500+ words. Do bookmark this page and continue 😊

Table of Contents

1. What we will learn and build?
2. The need to Integrate Third-Party Bootstrap UI
3. What's AdminLTE?
4. Downloading AdminLTE
5. Exploring the Folder Structure
6. Setting up ASP.NET Core MVC Project with Authentication
7. Understanding Layouts and Partial Views
8. Integrating AdminLTE with ASP.NET Core
9. Copying the Required Resources
10. Adding Layout pages & Partial Views
11. Adding Navigation
12. Navigation Indicator
13. Integrating the UI with existing Authentication
14. Enabling Authentication
15. Authenticated Status Based Menu
16. Update – Issue with filterizr Plugin for AdminLTE 3.0.5
17. Summary

What we will learn and build?

Here is a small demonstration of what we would have built by the end of this tutorial.



Here are the Features included.

- Integration with a Third Party Bootstrap Template.
- Clean usage and separation of Layouts, Views and Partial Views.
- Integration of Identity Authentication
- Scaffolded Identity
- Conditional Rendering
- Login / Register / Logout and more

The need to Integrate Third-Party Bootstrap UI

If you are back-end developer like me with just **OKayish** skills with recreating an entire HTML / CSS / JS Template from scratch, you would probably want to look into other options that make your application look 100x more Professional. Now, there are quite a lot of options, including paid templates as well.

For this article, we will use an Open Sourced Dashboard Template that is quite popular.

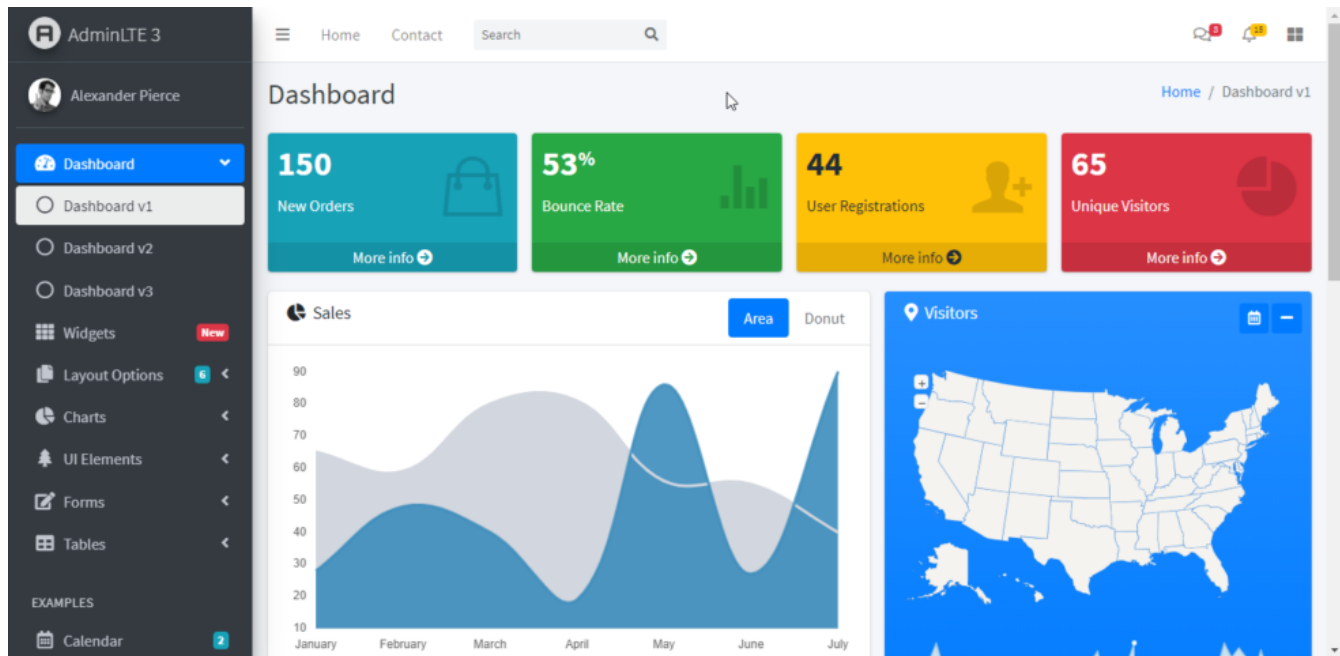
Here are the advantages of integrating an already built template.

- Professional UI
- Already Tested.
- Responsive.
- Would have a bunch of reusable components like datatables, forms, and more so that you don't have to re-invent the wheel.

What's AdminLTE?

AdminLTE is an open sourced Admin Dashboard Template that is built over Bootstrap. It is packed with quite a lot of responsive and commonly used components that are very easily integrated with your webapplications.

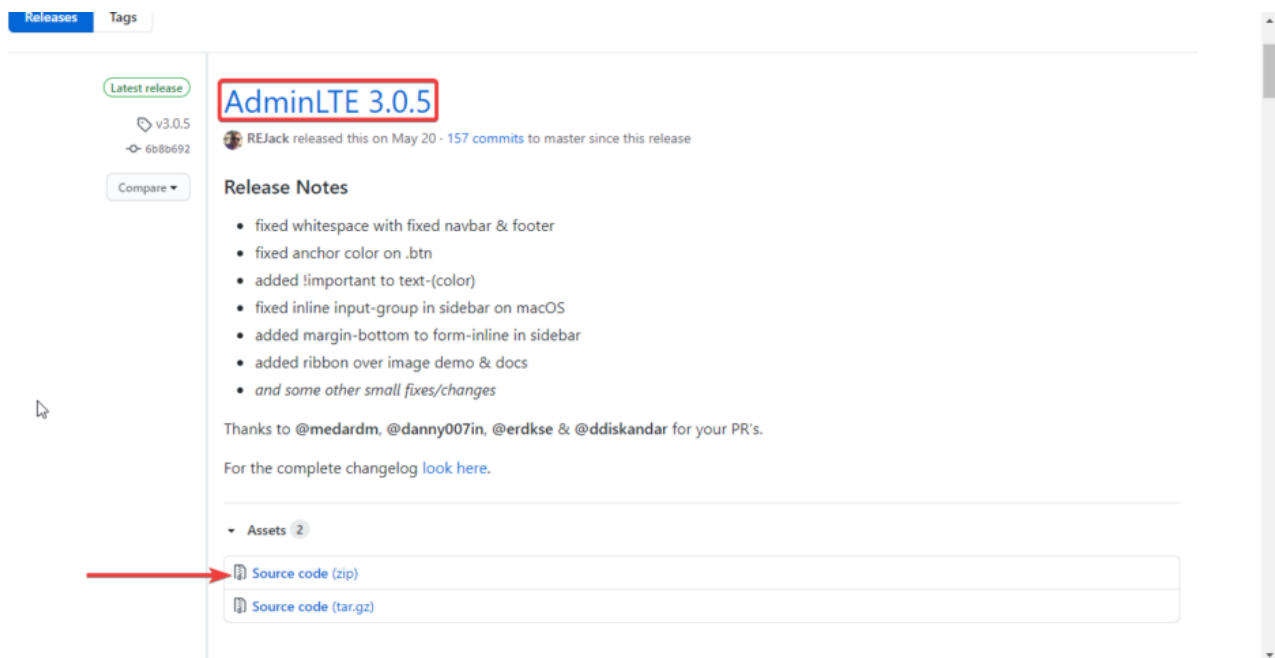
To get a better picture, click [here](#) to see a demo of AdminLTE in action.



You may notice how premium it already looks. For now, these pages are not bound to any server side applications. They are just plain old HTML files. In this article we will integrate this UI with our ASP.NET Core MVC Application with some clean practices.

Downloading AdminLTE

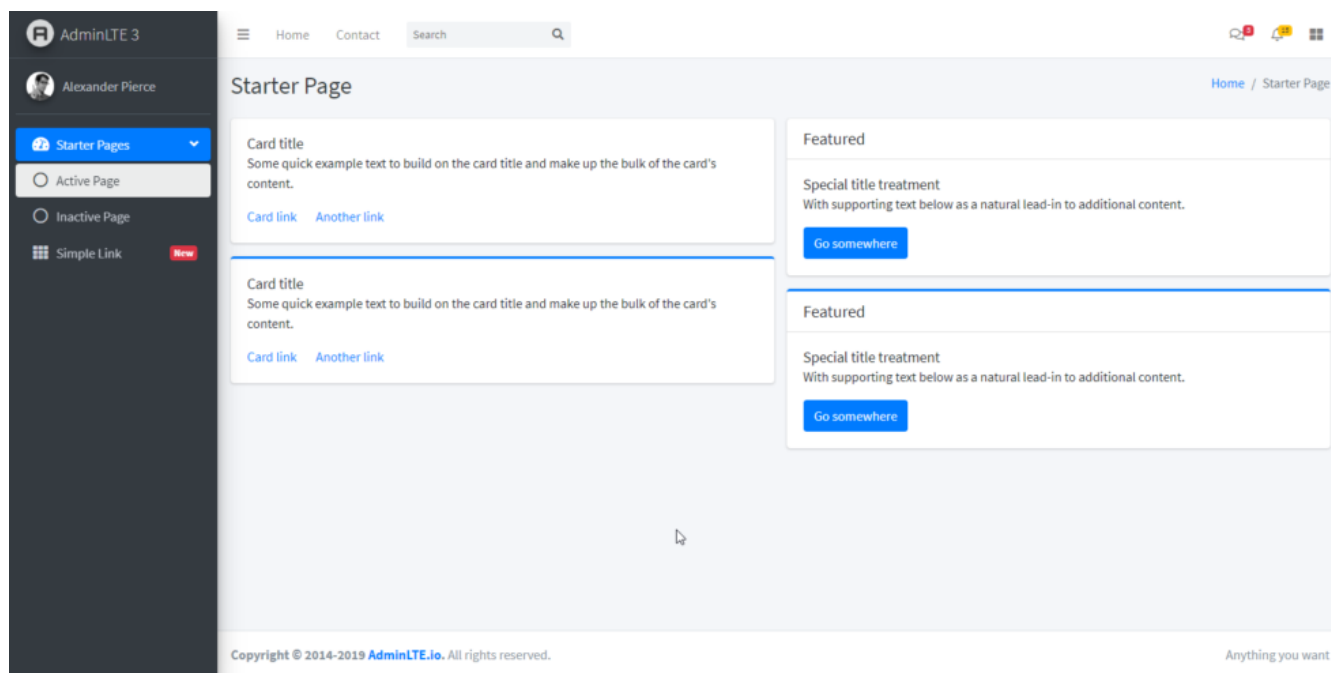
AdminLTE is completely **FREE** to use. [Follow this link to start downloading](#). At the time of writing this article, 3.0.5 is the latest version available. Click on the link to **Source Code** to download the zipped file on to your machine.



Exploring the Folder Structure

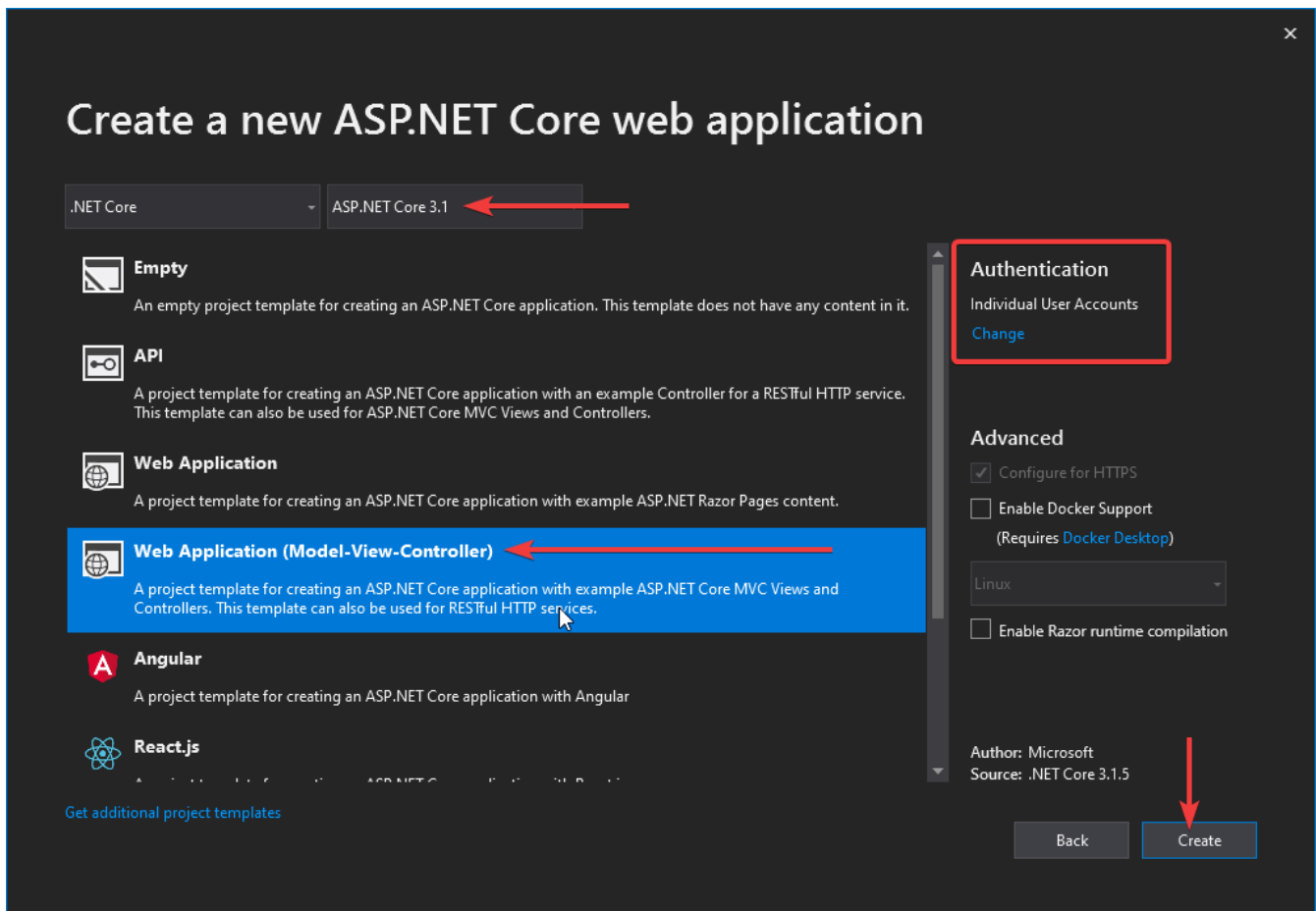
Once downloaded, extract the zipped file. Here you will find a bunch of folders and files. We will not have to touch each and every file, rather just a few. I will give you a brief overview on what each folder contains.

- dist/ – This is the distribution folder that contains all the css and js files, mostly all the static files of the application. We will need to copy this folder over to wwwroot folder of our MVC Project later on.
- pages/ – Here you get a list of all pre-made HTML files to refer to. This is quite an important section as it uses all the available components and can be helpful to check out how components are being utilized.
- plugins/ – 3rd party JS plugins like select2, jquery, datatables, etc are contained here. We will need this folder too.
- starter.html – Here we get a minimal setup of the HTML file. We will use this page to generate the _Layout.cshml for our ASP.NET Core MVC Application. I have attached a screenshot below.



Setting up ASP.NET Core MVC Project with Authentication

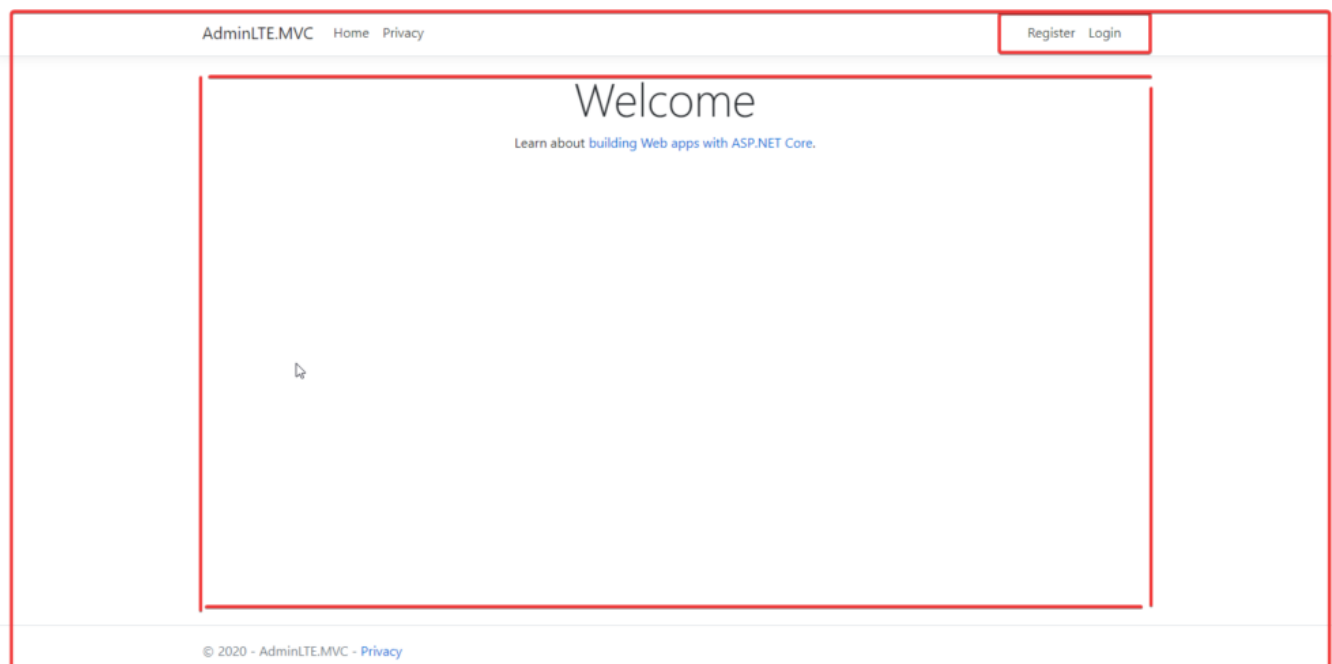
Let's create a new ASP.NET Core Application with the Model-View-Controller (MVC) Template. **Make sure you select the authentication mode to Individual User Accounts.** This enables us to use the built in Authentication (using Microsoft Identity). I will be using [Visual Studio 2019 Community](#).



Now that we have our AdminLTE files and ASP.NET Core Application ready, let's start integrating them. Before getting started, let's see how the default layout of ASP.NET Core work.

Understanding Layouts and Partial Views

Build and run the Application. This is the default layout that comes out of the box with ASP.NET Core 3.1 Web Applications.



This is how the page is split into.

- Main Layout Page – This is the master layout defined at /Views/Shared/_Layout.cshtml
- Navigate Panel – Within the master layout, a partial view reference is defined that calls the _LoginPartial.cshtml page.
- Content Body – Here is where the actual content goes.

PS, ASP.NET Core uses .cshtml (Razor markup files) extension for it's pages.

Now why this kind of separation?

In ASP.NET Core MVC, you can generate Views (CHTML) via controllers. Imagine a real life application having multiple controllers and views. You really don't want to define the entire HTML for each view do you? Because we already know that we will be having a site-wide common template. So what this Layout concept does is that, you define the layout cshtml just once, and add the content of each other pages dynamically within the layout page.

So, you define the entire HTML Part that essentially would be common throughout the application, and you separate the dynamic content in another cshtml page. This way, we can make the application much more maintainable and reduce the size of the entire application. Get the point?

Let's examine the file where the default layout is defined. Since we created a MVC templated application, we can find this file at ../Views/Shared/_Layout.cshtml in the solution structure. You can see that this page starts with a HTML tag. This suggests that this is an entire HTML page with all the body and head tags.

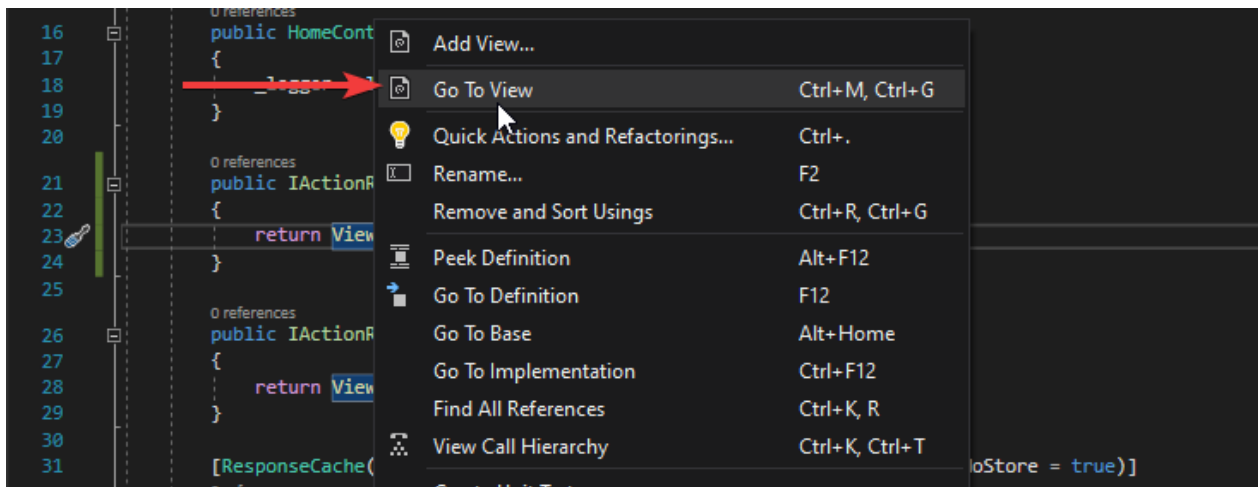
Now somewhere at line 33 or so, you would see the following.

```
1. <div class="container">
2.     <main role="main" class="pb-3">
3.         @RenderBody()
4.     </main>
5. </div>
```

@RenderBody is used to render the content of the child page. When you run the ASP.NET Core application, it navigates to the root of the application, that is, localhost:XXXX/Home and invokes the Index Method in the HomeController (as it is set as the default route). Let's see what this method contains.

```
1. public IActionResult Index()
2. {
3.     return View();
4. }
```

It just returns a view (.cshtml). Rightclick on the View and click Go To View. This will take us to associated CSHTML File.



You will be navigated to Views/Home/Index.cshtml. Here is the content of the page. This is exactly what we saw on the page when we ran the application. Makes sense?

```

1.  @{
2.      ViewData["Title"] = "Home Page";
3.  }
4.
5.  <div class="text-center">
6.      <h1 class="display-4">Welcome</h1>
7.      <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps
8.  </div>

```

This content is loaded by the @RenderBody tag, so that on runtime you get the complete web page (including the static HTML content and the dynamic code generated by C#).

Now, let's talk a bit on partial views. In your _Layout.cshtml page, you may find this somewhere in 20th line or so.

```

1.  <partial name="_LoginPartial" />

```

Unlike Views, Partial Views in ASP.NET Core renders the HTML output within another View's rendered output. There can be cases where your applications has components that can be reused anywhere within the application. In our case we have the top Navigation bar that is quite common through the application. You can find the _LoginPartial.cshtml in the shared folder as well.

Integrating AdminLTE with ASP.NET Core

With the basic concepts of Layouts, View and partial Views clear, it will be easier to integrate any 3rd Party HTML Layout to our ASP.NET Core Applications. Let's proceed.

Copying the Required Resources

As mentioned earlier, AdminLTE is built over Bootstrap. Hence it contains quite a lot of jquery and js integrations as well. We would not be just copying the HTML content, but also the related resources, like css, images, libraries, js files , etc.

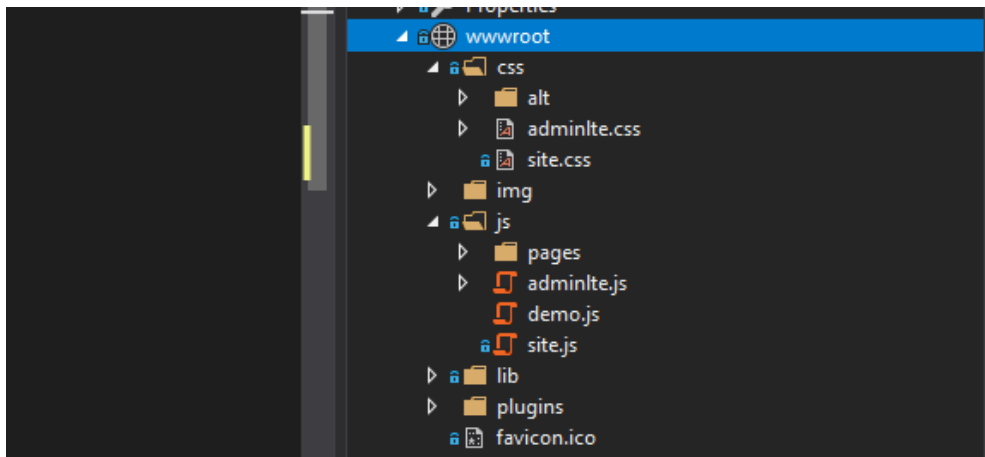
In our AdminLTE Folder, Navigate to `\dist\css` and copy the content over to the `wwwroot\css` folder in our Visual Studio. Do the same for the `js` folder as well.

PS, to copy the contents over, just copy the selected folder/file and go to Visual Studio. Here you may find a `wwwroot` folder. This is the folder meant to hold the static files of ASP.NET Core applications. Just paste the copied file here with a simple `CTRL+V` command.

Next copy the entire `img` folder to the `wwwroot` folder. Navigate back to the root of the AdminLTE folder and copy the `plugins` folder to the `wwwroot` folder as well.

Note that we will not be using all the plugins in the `plugins` folder for now. You may need to remove the unused files once you are done with the application to save small space. For now, let's keep them.

This is how your `wwwroot` folder would look like after we are done copying the content.



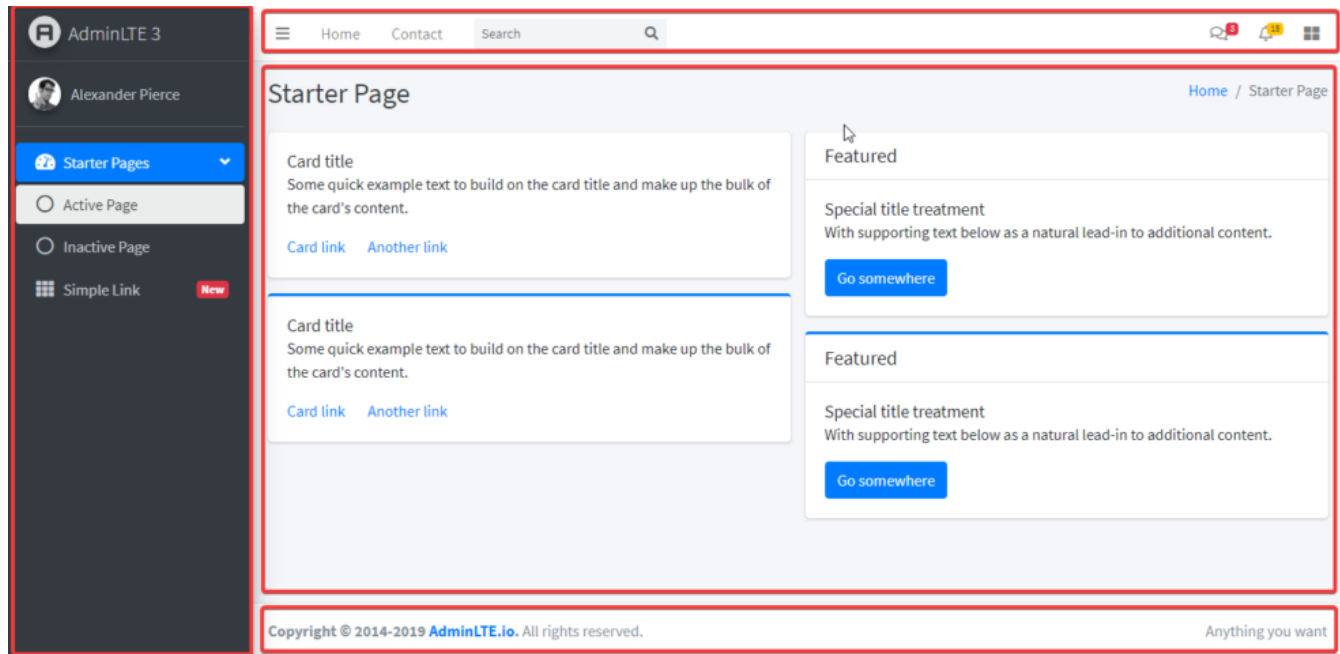
Now that we have all the required resources moved to our application. Let's wire up the Layout Page.

Adding Layout pages & Partial Views

An Application can have multiple Layout pages. For this tutorial, let's not disturb the existing `_Layout.cshtml` page. Rather, let's build one specifically for AdminLTE.

In the Shared Folder, create a new folder named `AdminLTE`. Here is where you would want to put all the `.cshtml` related to AdminLTE.

Before building the layout pages and partial views, let's decide on how we will separate the HTML content. Note that we will be using the `starter.html` page to build the Layout. Open up the `starter.html` page on your browser.



Here is how we could split up the page into.

- Side Navigation
- Top Navigation
- Body
- Footer

Additionally we will want to add 2 more partial views that will hold the references to the css and js files respectively. This makes it easier to sort and manage your resource references.

Under the AdminLTE Folder create a new View and name it `_Layout.cshtml`. **Make sure to uncheck the partial view and layout page options.**

Next, let's start adding the partial view files. In the same AdminLTE folder, add a new view and name it `_MainNavigation.cshtml`. **This time, check the "create as a partial view" option.**

View name:

Template:

Model class:

Data context class:

Options:

☒ Create as a partial view

☐ Reference script libraries

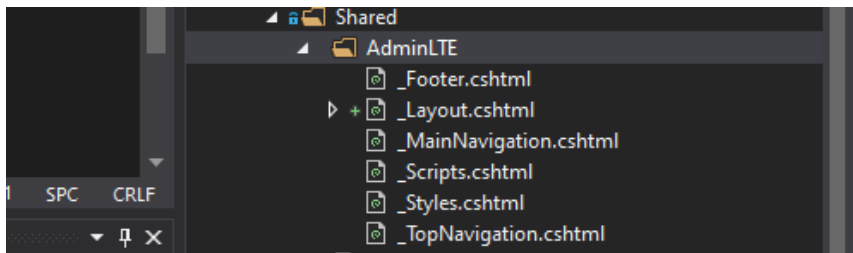
☐ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Similarly, add other partial views with the following file names.

- _TopNavigation
- _Footer
- _Scripts
- _Styles

This way, you can neatly structure you Layouts.



Let's start adding content to each file. So, now we have the idea of how we will split the HTML. Open up starter.html in a code editor. I use VS Code. The starter.html has over 330 lines of HTML. Let's move the code one by one.

Scripts. Go to the end of the file. Above the body tag, you can find few of the script reference there. Cut it and Paste it over to _Scripts.cshtml. Replace in the starter.html with the following.

```
1. <partial name="AdminLTE/_Scripts" />
```

Footer. Just above where the scripts were defined, you can find the footer container. Cut this and move to _Footer.cshtml. Add the following instead in the starter.html. We will move the codes from here to the _Layout.cshtml once we are done with the partial views.

```
1. <partial name="AdminLTE/_Footer" />
```

Body. You can find a div class "content". Delete the entire child div which has a class named "row" This is where we would want to put our @RenderBody tag.

Main Navigation – Search for a class "main-sidebar". Cut it and move it to _MainNavigation.cshtml. Here you have all the sidebar items at once place for you to extend.

```
1. <partial name="AdminLTE/_MainNavigation" />
```

Top Navigation – Search for the class "main-header" and move the content over to _TopNavigation.cshtml.

```
1. <partial name="AdminLTE/_TopNavigation" />
```

Styles. Below the title tag, you can see a bunch of stylesheet references. Cut and paste to _Scripts.cshtml and Replace with the following instead.

```
1. <partial name="AdminLTE/_Styles" />
```

Now, we have moved all the possible componets to partial views. What remains in our code editor would look something like this. Move this to the _Layout.cshtml

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="utf-8" />
5.     <meta name="viewport" content="width=device-width, initial-scale=1" />
6.     <meta http-equiv="x-ua-compatible" content="ie=edge" />
7.
8.     <title>AdminLTE 3 | Starter</title>
9.     <partial name="AdminLTE/_Styles" />
10. </head>
11. <body class="hold-transition sidebar-mini">
12.     <div class="wrapper">
13.         <partial name="AdminLTE/_TopNavigation" />
14.         <partial name="AdminLTE/_MainNavigation" />
15.         <div class="content-wrapper">
16.             <div class="content-header">
17.                 <div class="container-fluid">
18.                     <div class="row mb-2">
19.                         <div class="col-sm-6">
20.                             <h1 class="m-0 text-dark">Starter Page</h1>
21.                         </div>
22.                         <div class="col-sm-6">
23.                             <ol class="breadcrumb float-sm-right">
24.                                 <li class="breadcrumb-item"><a href="#">Home</a></li>
25.                                 <li class="breadcrumb-item active">Starter Page</li>
26.                             </ol>
27.                         </div>
28.                     </div>
```

```

29.         </div>
30.     </div>
31.     <div class="content">
32.         <div class="container-fluid">
33.             @RenderBody()
34.         </div>
35.     </div>
36. </div>
37. <aside class="control-sidebar control-sidebar-dark">
38.     <div class="p-3">
39.         <h5>Title</h5>
40.         <p>Sidebar content</p>
41.     </div>
42. </aside>
43. <partial name="AdminLTE/_Footer" />
44. </div>
45. <partial name="AdminLTE/_Scripts" />
46. </body>
47. </html>

```

Congrats, we have separated the HTMLs to layouts and partial views. Now build and run the application. You would see absolutely no change. Why? Because we have not mentioned anywhere to use our new layout, have we?

For this, Navigate to Views/Home/Index.cshtml and mention the layout manually.

```

1.  @{
2.      ViewData["Title"] = "Home Page";
3.      Layout = "~/Views/Shared/AdminLTE/_Layout.cshtml";
4.  }

```

After this, run the application again. You would see quite a lot of changes. (which you are probably not happy with :p). The entire page would be broken. However, you can see that we are able to display the content we passed from the View. Now let's fix the page.



Any guesses on why this page is broken? It's quite simple if you already have experiences with HTML pages. This is commonly because the page cannot find the stylesheets that contains the styles defined. We copied the styles and scripts to the Partial Views, but we did not change the references. Let's change the paths to point to the specific files in the wwwroot folder.

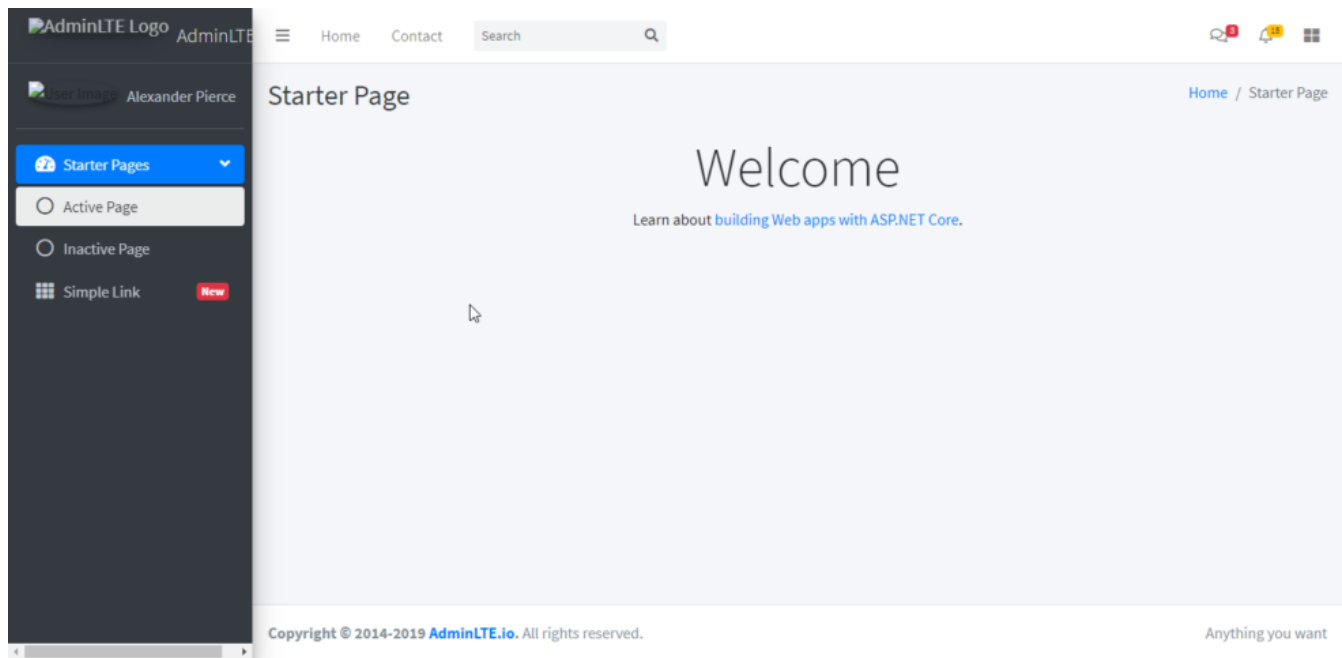
Open up `_Styles.cshtml` and modify as below

1. `<link rel="stylesheet" href="~/plugins/fontawesome-free/css/all.min.css">`
2. `<link rel="stylesheet" href="~/css/adminlte.min.css">`
3. `<link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700"`

Open up `_Scripts.cshtml` and modify as below

1. `<script src="~/plugins/jquery/jquery.min.js"></script>`
2. `<script src="~/plugins/bootstrap/js/bootstrap.bundle.min.js"></script>`
3. `<script src="~/js/adminlte.min.js"></script>`

That's it. Run the application again.



There you go! You can see that the page starts looking great. The only issue is a broken image reference. You can fix it similarly by going to `_MainNavigation.cshtml` and fixing the reference issue like we did earlier.

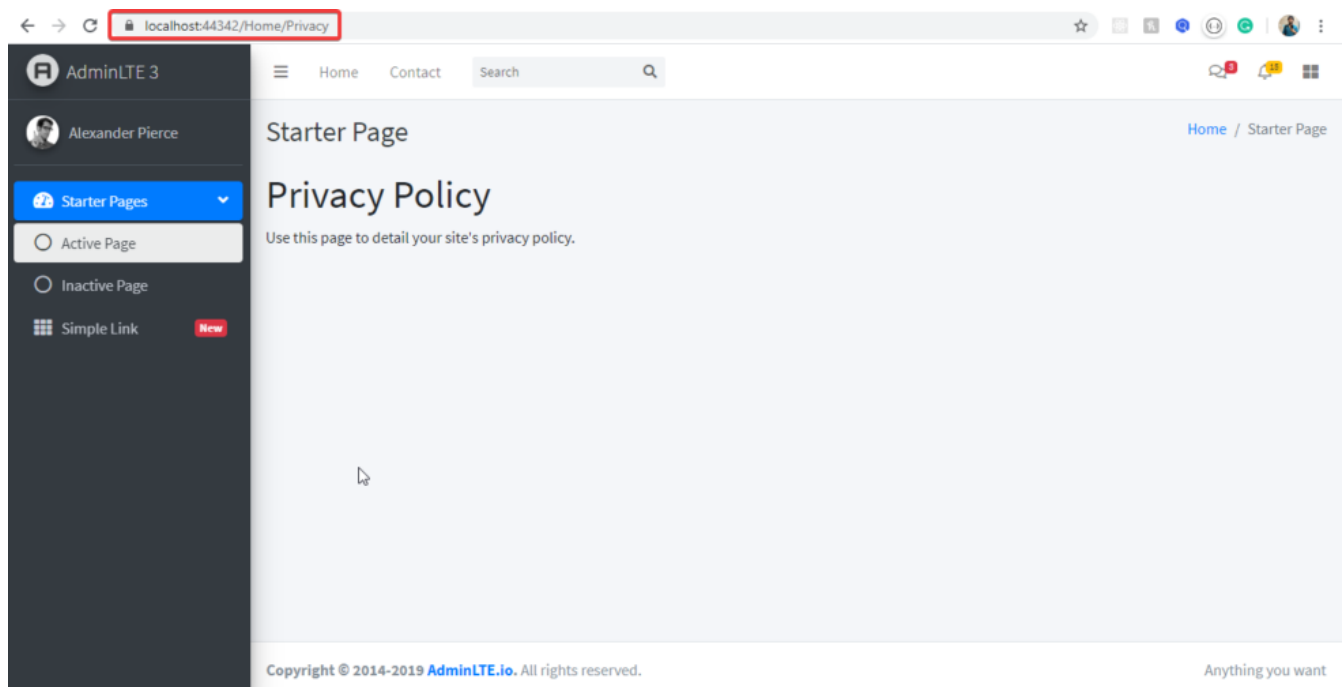
Now, let's add this layout to the Privacy Page too! Go to `Views/Home/Privacy.cshtml` add in the following line.

```

1.  @{
2.      ViewData["Title"] = "Privacy Policy";
3.      Layout = "~/Views/Shared/AdminLTE/_Layout.cshtml";
4.  }

```

Run the application and navigate to `localhost:xxx/home/privacy`. This is what we get. We are all setup now.



Adding Navigation

Let's add some navigation menu to our _MainNavigation.cshtml. What we need to do is simple. Remove the static paths and add add links to our controller methods. For now, we will add 2 Navigation Items. Home and Privacy.

- Home will be linked to Home/Index
- Privacy page will be linked to Home/Privacy

```

1. <aside class="main-sidebar sidebar-dark-primary elevation-4">
2.     <a href="~/Home" class="brand-link">
3.         
5.         <span class="brand-text font-weight-light">AdminLTE 3</span>
6.     </a>
7.     <div class="sidebar">
8.         <div class="user-panel mt-3 pb-3 mb-3 d-flex">
9.             <div class="image">
10.                
12.            <div class="info">
13.                <a href="#" class="d-block">Alexander Pierce</a>
14.            </div>
15.        </div>
16.        <nav class="mt-2">
17.            <ul class="nav nav-pills nav-sidebar flex-column" data-widget="treeview" r
18.                <li class="nav-item ">
19.                    <a asp-controller="Home" asp-action="Index" class="nav-link">
20.                        <i class="nav-icon fas fa-home"></i>
21.                        <p>
22.                            Home
23.                        </p>
24.                    </a>
25.                </li>
26.                <li class="nav-item ">
27.                    <a asp-controller="Home" asp-action="Privacy" class="nav-link">
28.                        <i class="nav-icon fas fa-lock"></i>
29.                        <p>
30.                            Privacy
31.                        </p>
32.                    </a>
33.                </li>
34.            </ul>
35.        </nav>
36.    </div>
37. </aside>

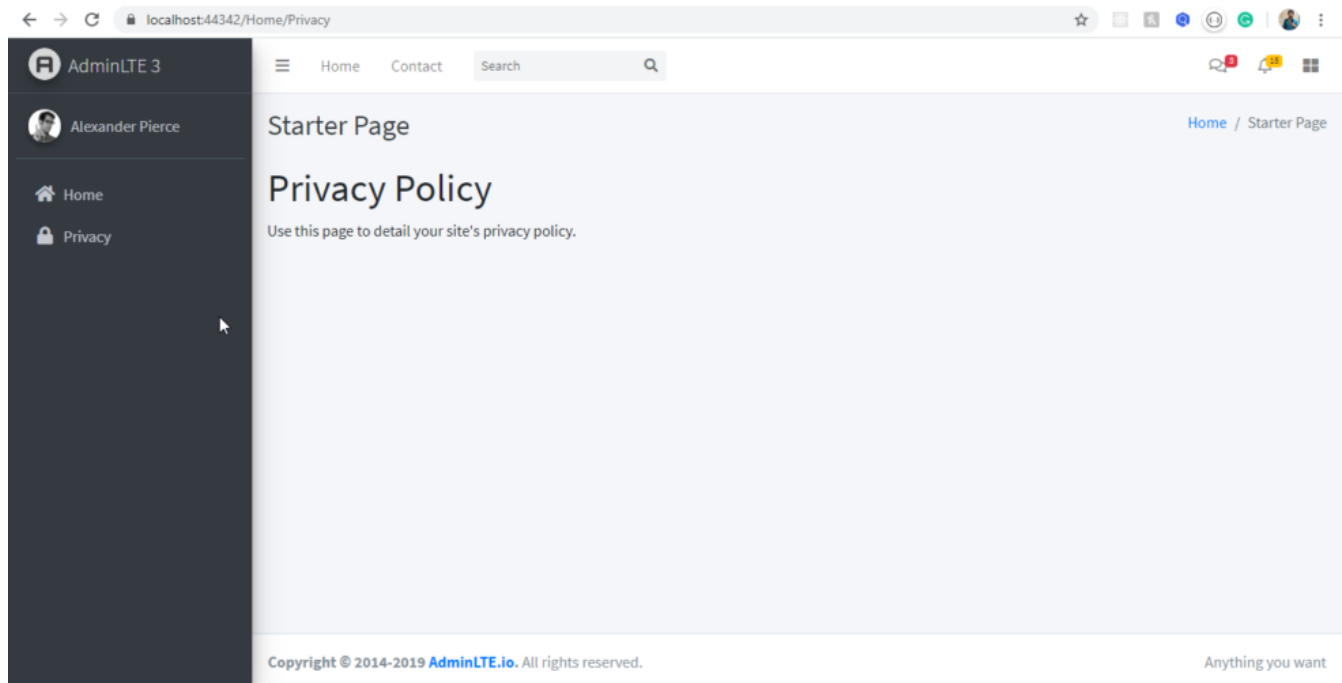
```

Line 3 and 10 – image reference fix.

Line 19 – You can see for the navigation Item named “Home”, we are linking to the controller “Home” and Index actions.

Line 27 – For the Privacy page, we like to the Home Controller and Privvacy method.

Run the application and check.



So that's working too. But there is one detail that we are missing. Navigation Indicator. There is no indication in our sidebar about the currently viewed page. For now, since we have just 2 navigation item, it's quite fine to check the URL and understand which page we are at right now. But this is not ideal. Let's fix it first.

Navigation Indicator

For this to work, we need data from our ASP.NET Core regarding the current controller and action method. And based on this we need to change the class of the corresponding navigation item to active. Active means the current page.

Add a new folder in the root of the project. Name it Helpers. Under it, add a new NavigationIndicatorHelper class.

```

1.  public static class NavigationIndicatorHelper
2.  {
3.      public static string MakeActiveClass(this IUrlHelper urlHelper, string controller,
4.      {
5.          try
6.          {
7.              string result = "active";
8.              string controllerName = urlHelper.ActionContext.RouteData.Values["controll
9.              string methodName = urlHelper.ActionContext.RouteData.Values["action"].ToS
10.             if (string.IsNullOrEmpty(controllerName)) return null;
11.             if (controllerName.Equals(controller, StringComparison.OrdinalIgnoreCase))
12.             {
13.                 if (methodName.Equals(action, StringComparison.OrdinalIgnoreCase))
14.                 {
15.                     return result;
16.                 }
17.             }
18.             return null;
19.         }
20.         catch (Exception)
21.         {
22.             return null;
23.         }

```



```

24.     }
25. }

```

Here is what this helper class does. It has an extension method with the URLHelper. By this way, you can invoke it in the cshtml page too. It takes in the controller and the action method name and checks it with the current route data. If they match, we return a string "active", else null.

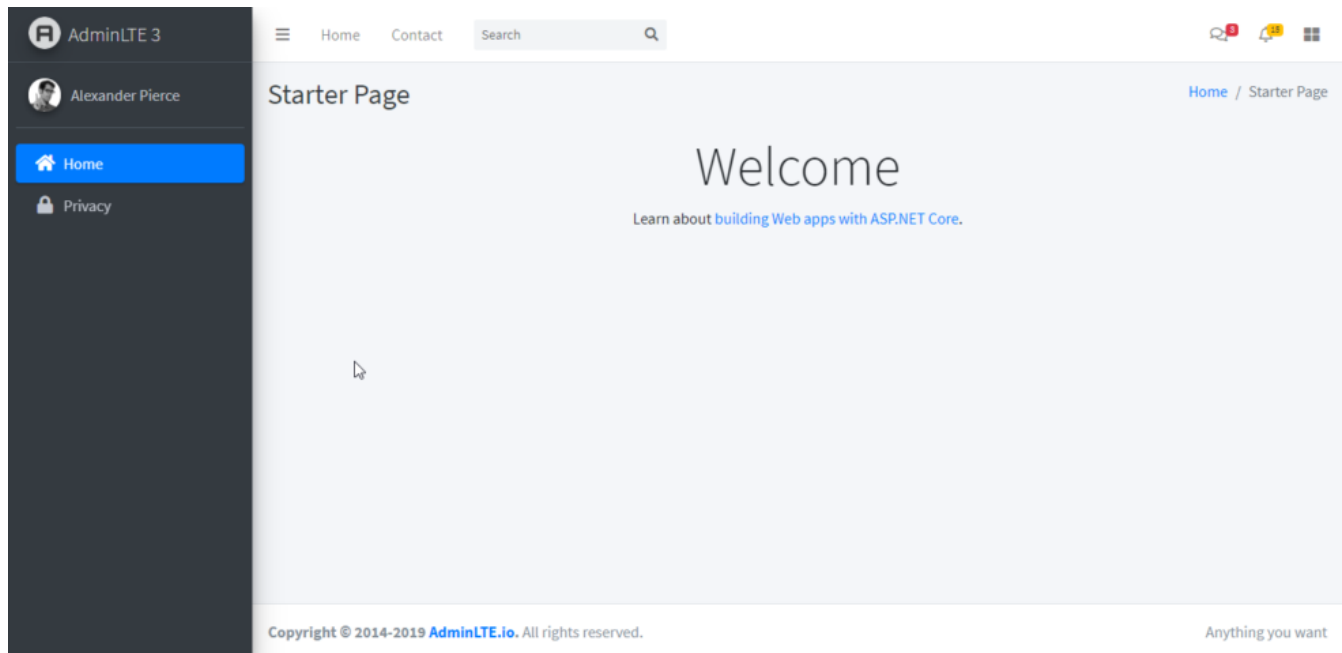
Go to `_MainNavigation.cshtml`. Modify / Add these lines

```

1. @using static AdminLTE.MVC.Helpers.NavigationIndicatorHelper;
2. .
3. .
4. <a asp-controller="Home" asp-action="Index" class="nav-link @Url.MakeActiveClass("home
5. .
6. .
7. <a asp-controller="Home" asp-action="Privacy" class="nav-link @Url.MakeActiveClass("ho

```

Run the application. Now, we have got a working navigation indicator. Sweet, yeah?



With that out of the way, as promised, let's check out how to integrate authentication using the existing Identity Authentication.

So Far So Good?

Stay up to date! Get all the latest & greatest articles / in depth Guides on .NET Core / ASP.NET Core delivered straight to your inbox. Subscribe now!

Integrating the UI with existing Authentication

Let me draw you a scenario. The requirement is that we need to secure a particular view. In our case, we have Index and Privacy pages. Let's say we keep the Index Page available for everyone. But limit the access to the Privacy page and allow access only if the user is authenticated. We will also want to hide the item from the navigation menu if not authenticated.

If the visitors tries to access the resource (without auth) by navigating directly to ../Home/Privacy, we redirect him to a login page. So this is what we will be doing here. This is quite a practical scenario right?

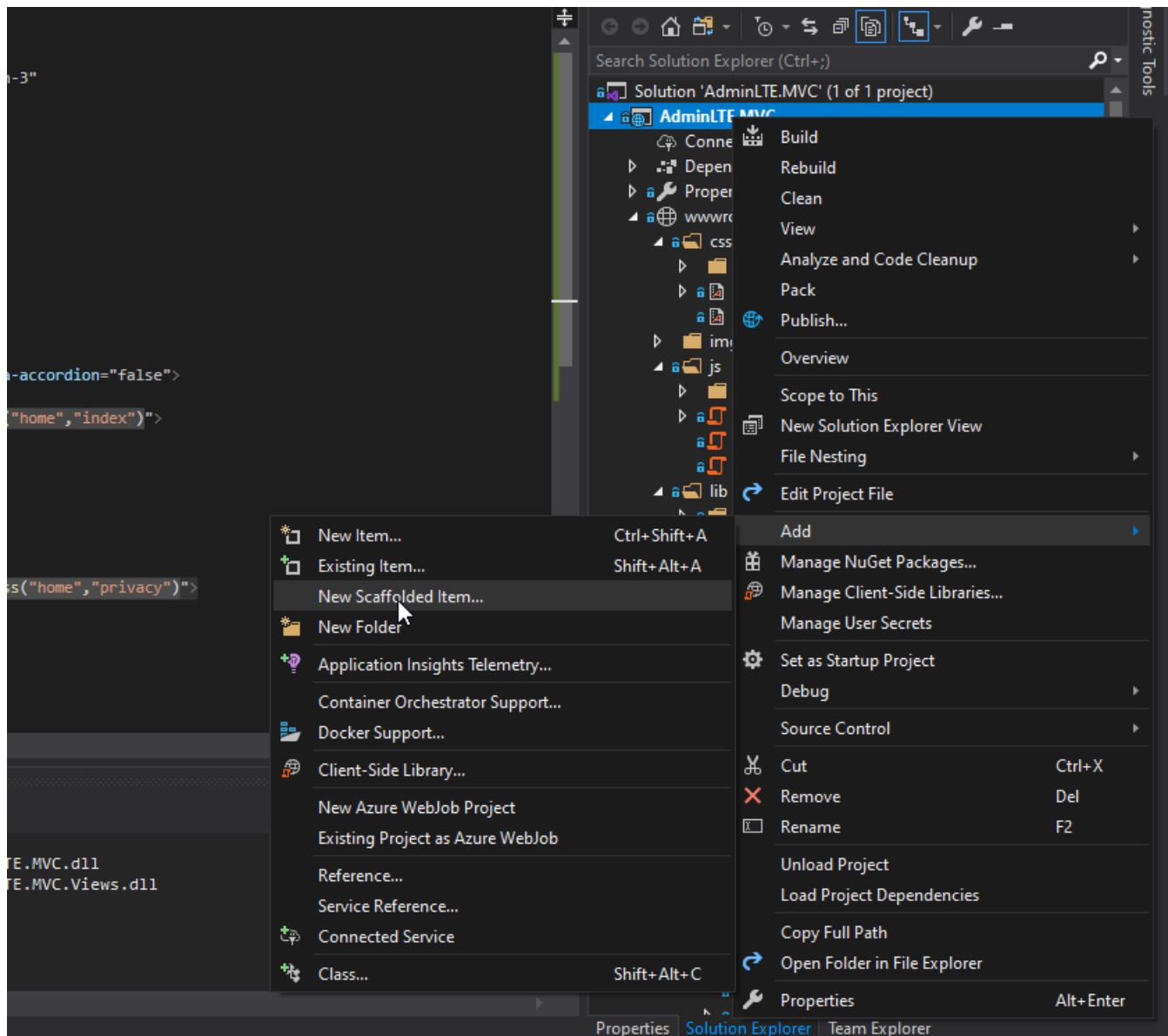
AdminLTE comes with a default Login and Register page too! It's located under ../pages/examples folder as login.html and register.html.

By now you would be quite clear on how to integrate the UI. But there is a catch here. If you examine our ASP.NET Core MVC application, you would no where find a Login or Register page. But we added authentication to the application while creating it, Remember? So how does that work.

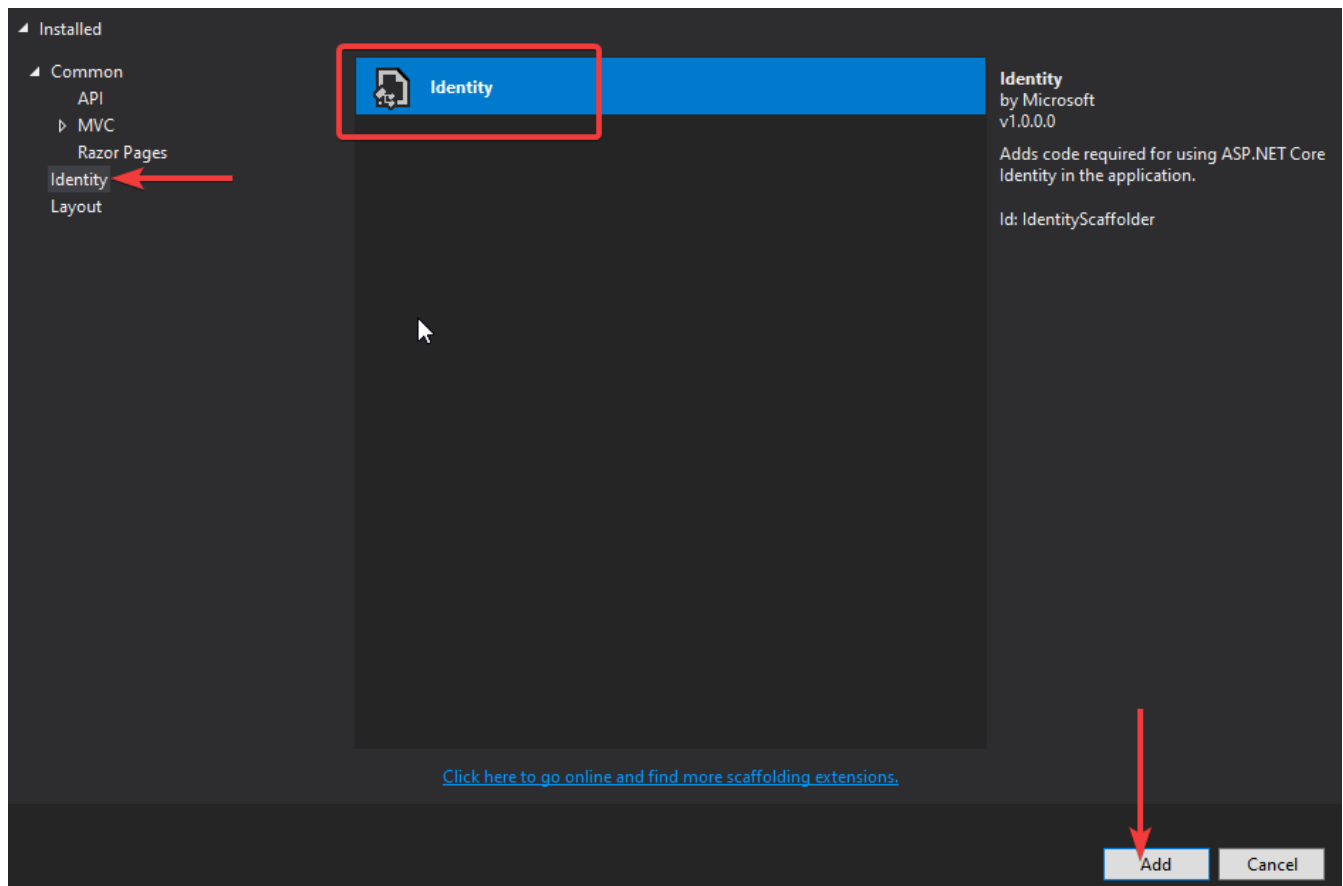
Sometime, during the announcement of ASP.NET Core 2.1. Microsoft revealed that the Identity UI (all the pages related to Microsoft Identity) would be moved to a Razor Library. Thus, it works out of the box even though it's not visible to us.

But, what if we had to do some modification to it? That's where Identity Scaffold comes to play. Follow these steps to bring back the Login and Register Razor Pages for us to modify.

Right click on the Project -> Add New -> New Scaffolded Item.

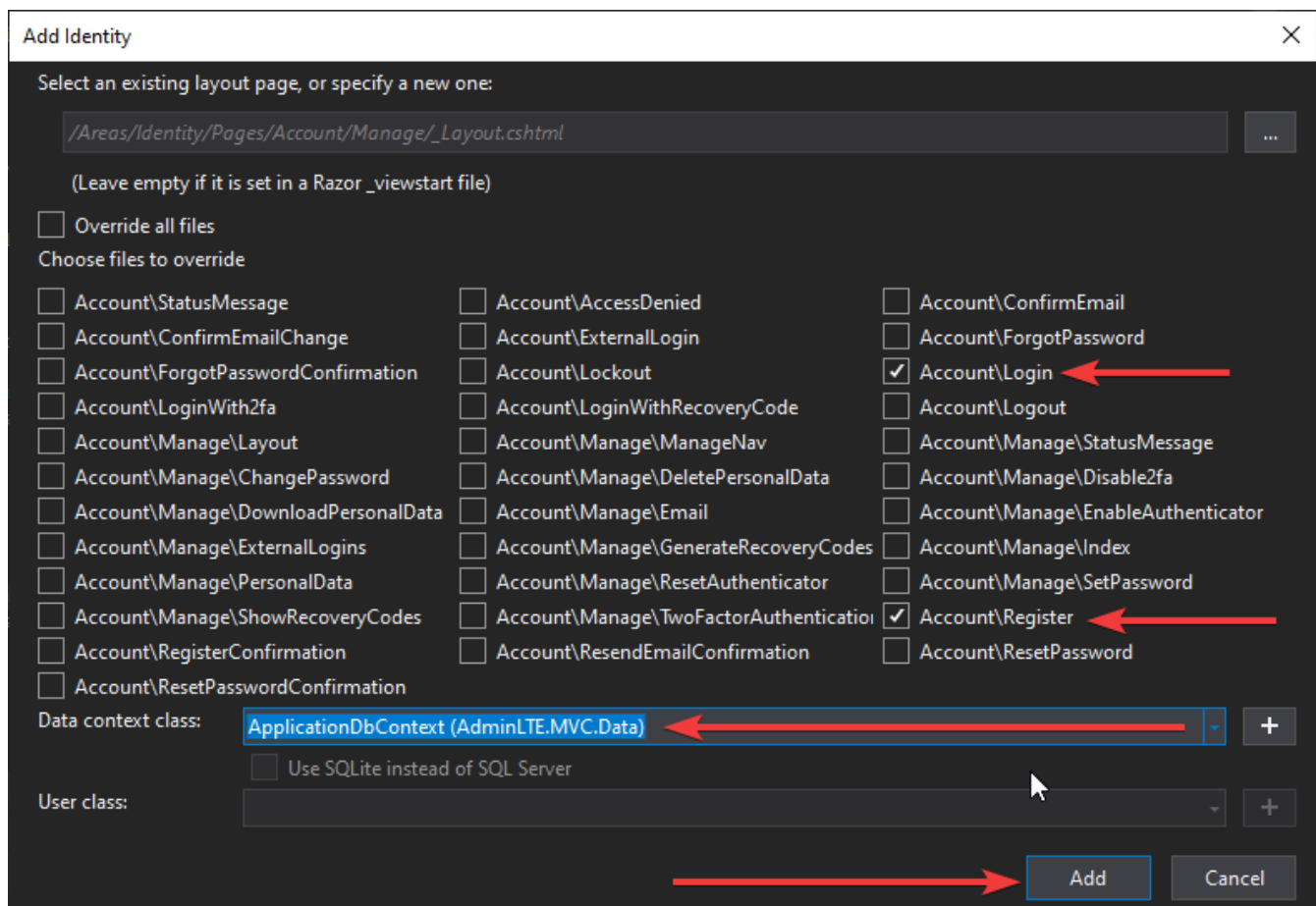


In this next dialog, select Identity and click Add.

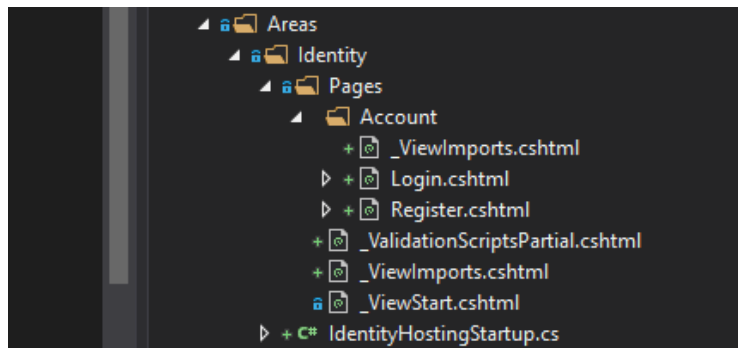


Now, we get to select the required Identity pages. To keep things simple, let's add only the Login and Registration page. Make sure you select the data class as well. Click Add. Now, Visual Studio does its magic and generate the selected files.

In the background, Visual Studio also makes a DataContext class for you and registers it in the startup services with a default local db.



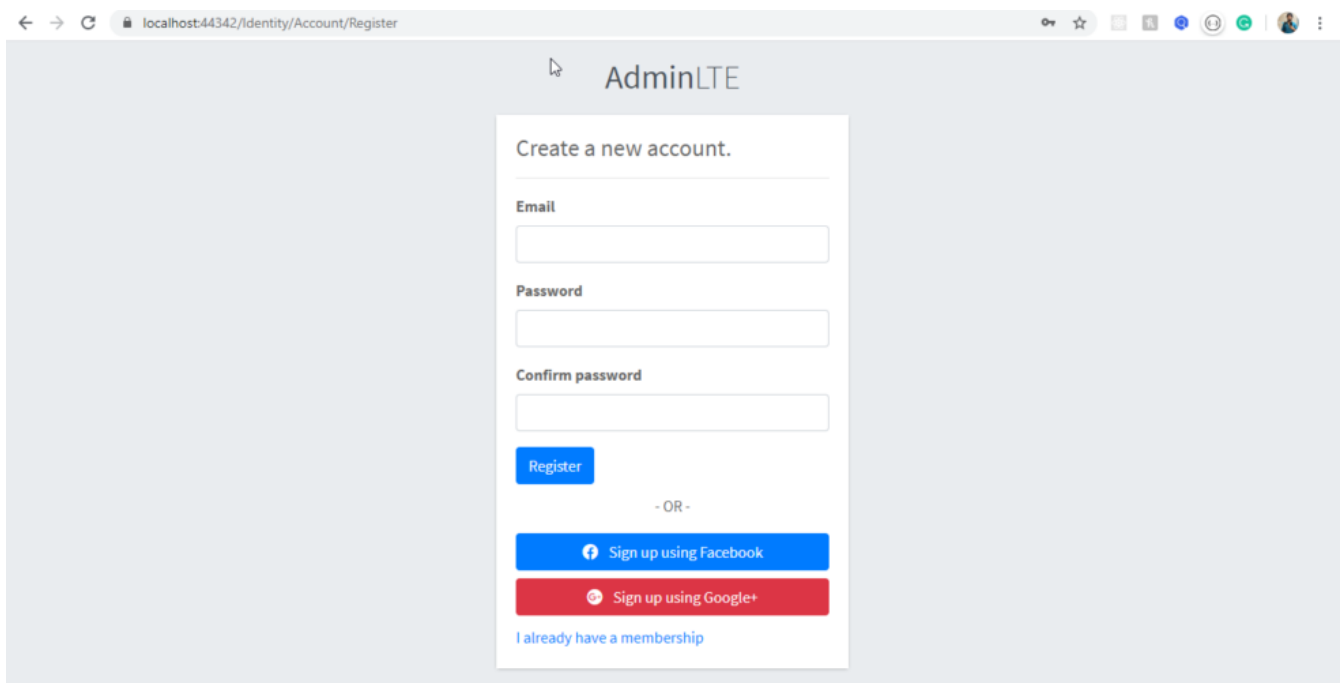
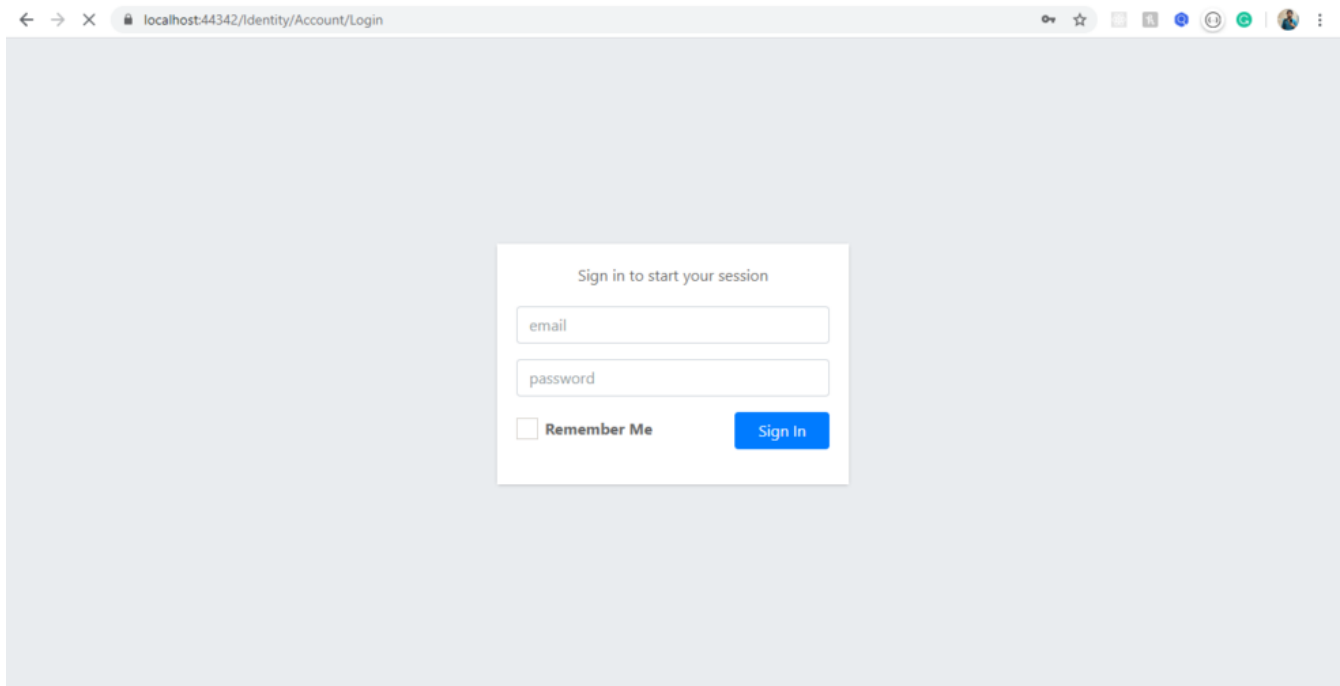
Once it's done, you see the following folder with our selected Identity Pages.



Now we will integrate these Views with the login.html and the register.html. It would help you if you do this integration on your own, to help understand the scenarios better. To keep the article compact, i will just the just add the links to the changed files.

- Login.cshtml – <https://github.com/iammukeshm/AdminLTE.MVC/blob/master/AdminLTE.MVC/Areas/Identity/Pages/Account/Login.cshtml>
- Register.cshtml – <https://github.com/iammukeshm/AdminLTE.MVC/blob/master/AdminLTE.MVC/Areas/Identity/Pages/Account/Register.cshtml>

These are the resulting page. Pretty neat, yeah?



Enabling Authentication

Now that we have our Identity Pages ready, let us setup our ASP.NET Core Application to enable Authentication.

We will secure all the Controller methods by default. As per our requirement, we need to allow any random visitor to use the Home/Index method.

Navigate to Startup.cs/ConfigureServices method add these lines.

```
1. services.AddMvc(o =>
2. {
3.     var policy = new AuthorizationPolicyBuilder()
4.         .RequireAuthenticatedUser()
5. }
```

```

        .Build();
6.         o.Filters.Add(new AuthorizeFilter(policy));
7.     });

```

This will add a new policy to the ASP.NET Core MVC Application that every method needs an authenticated user, unless we define it as [AllowAnonymous]

Now, go to the Home Controller and add [AllowAnonymous] above the Index method. This means that the method can be accessed by anyone.

```

1.     [AllowAnonymous]
2.     public IActionResult Index()
3.     {
4.         return View();
5.     }

```

That's it. Just run the application and check. You would be able to navigate to the Home page without any issues. But when you go to the Privacy page, you will be redirected to the login page. How simple was that.

Authenticated Status Based Menu

We do not want the anonymous user to see the Privacy link in the navigation. To hide the item, go to _MainNavigation.cshtml

Here is the simple logic. We will need to add a condition (if) to check if the current user is authenticated, if yes, display the privacy link. Else, no. Here is how you code it.

```

1.     @if (User.Identity.IsAuthenticated)
2.     {
3.         <li class="nav-item ">
4.             <a asp-controller="Home" asp-action="Privacy" class="nav-link @Url.MakeActiveC
5.                 <i class="nav-icon fas fa-lock"></i>
6.                 <p>
7.                     Privacy
8.                 </p>
9.             </a>
10.        </li>
11.    }

```

Now, there is one more thing that we can add here using the conditional check of authentication. In the side do you see a random image with a random name? Does it make sense to add a similar condition there? Yeah, right? So what we will do is, show the name of the authenticated user . If not authenticated let's show a "Hi, Visitor".

Modify the _MainNavigation.cshtml like below.

```

1.     @if (User.Identity.IsAuthenticated)
2.     {
3.         <div class="user-panel mt-3 pb-3 mb-3 d-flex">

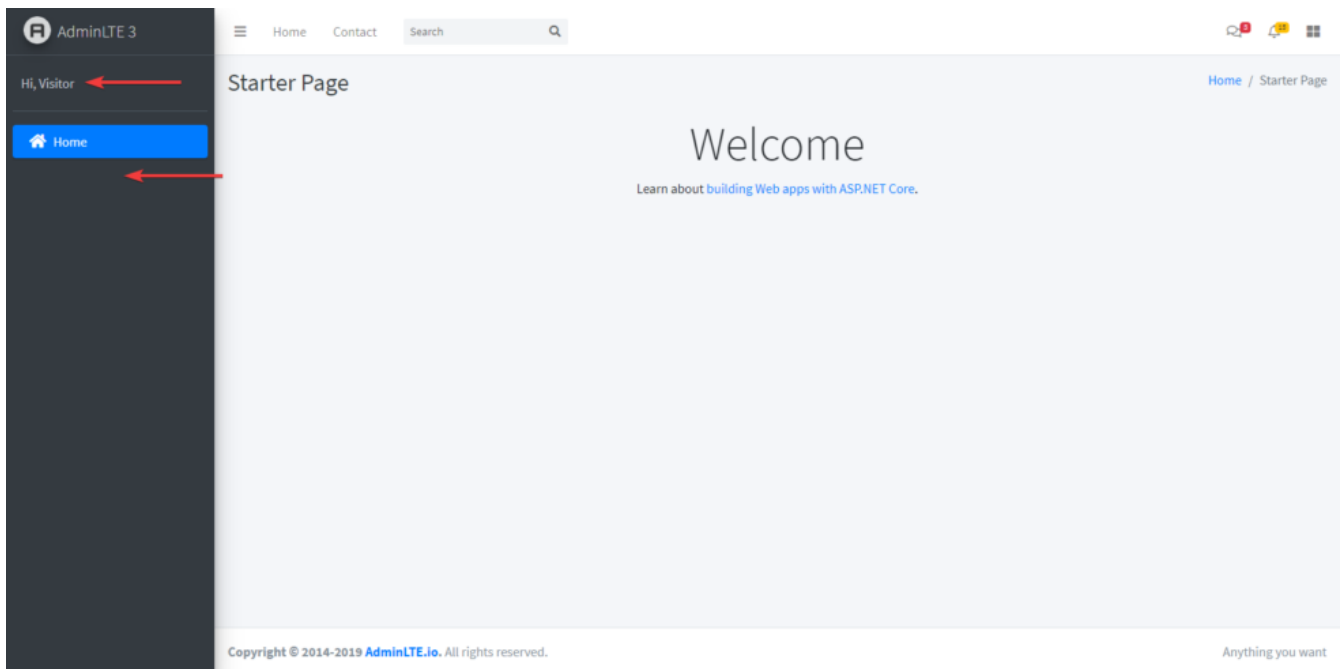
```

```

4.         <div class="image">
5.             
6.         </div>
7.         <div class="info">
8.             <a href="#" class="d-block">Hi, @User.Identity.Name</a>
9.         </div>
10.    </div>
11. }
12. else
13. {
14.     <div class="user-panel mt-3 pb-3 mb-3 d-flex">
15.         <div class="info">
16.             <a href="#" class="d-block">Hi, Visitor</a>
17.         </div>
18.     </div>
19. }

```

Build and run the application.



Quite self explanatory, yeah? Finally let's add the login/register links to the Top Navigation Menu.

Go to `_TopNavigation.cshtml` and modify the code.

```

1.  @if (User.Identity.IsAuthenticated)
2.  {
3.      <li class="nav-item d-none d-sm-inline-block">
4.          <form class="form-inline" asp-area="Identity" asp-page="/Account/Logout" asp-r
5.              <button type="submit" class="nav-link btn btn-link text-dark">Logout</butt
6.          </form>
7.      </li>
8.  }
9.  else
10. {

```


◀ [Progress Bar] ▶

AdminLTE

Create a new account.


Email


Password

Confirm password

Register

- OR -

 Sign up using facebook

 Sign up using Google+

I already have a membership

```
> dotnet ef database update
```

25/36

There are migrations for ApplicationDbContext that have not been applied to the database

- Migrations Applied Try refreshing the page

[Register](#) [Login](#)

This app does not currently have a real email sender registered, see [these docs](#) for how to configure a real email sender. Normally this would be emailed:
[Click here to confirm your account](#)

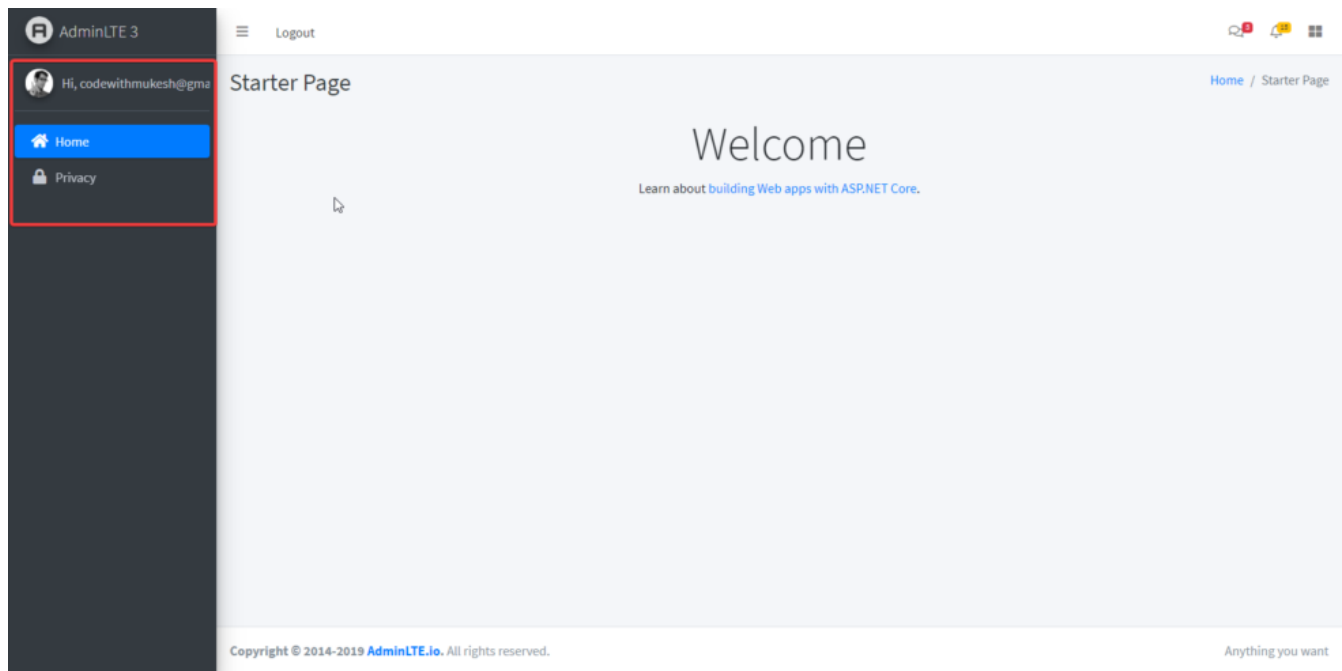
© 2020 - AdminLTE.MVC - [Privacy](#)

[Sign in to start your session](#)

codewithmukesh@gmail.com

•••••

☐ Remember Me[Sign In](#)



Great, everything works as we intended. Let's me wind up this really long article 😊

Update – Issue with filterizr Plugin for AdminLTE 3.0.5

Recently one of the readers posted an issue in the comments section that he faced while working with AdminLTE 3.0.5. The exception goes something like this.

Build: The module `"/FilterizrOptions/defaultOptions"` has no exported member `'RawOptionsCallbacks'`. Did you mean `'import RawOptionsCallbacks from "/FilterizrOptions/defaultOptions"'` instead.

With AdminLTE 3.0.5, the filterizr plugin for some reason still has its own source code files, which can cause TypeScript errors in Visual Studio if you try to run the project (I didn't face such an issue). If you face the similar issue, kindly remove the plugin from your solution. Delete everything in filterizr folder except for the following files:

- filterizr.min.js
- jquery.filterizr.min.js
- vanilla.filterizr.min.js

If you found this article helpful,



Summary

In this detailed beginner-friendly guide, We have learnt Integrating AdminLTE with ASP.NET Core 3.1 MVC. We covered several topics including Layouts, Views, PartialView, Authentication, Identity, Navigation Indicator, and much more. You can check out the complete source code for this demonstration over at my [Github](#). Do not forget to share this article and leave your comments below. Happy Coding 😊

Shall we extend this Application by adding more features to it in another article? Like jquery datatable, CRUD, Role based Authorization, and more? Leave your opinions about it,

1 Shares



Facebook
0



reddit 0



Twitter 0



LinkedIn
0



Pinterest
1

20 Comments



Joshav chhetri on June 21, 2020 at 3:34 pm

thanks for the amazing post. I have been following your blog since 1 week. And also i am looking for post about custom login feature without using individual authentication of identity. 😊

Reply



Mukesh Murugan on June 24, 2020 at 9:42 pm

Hi Joshav, Thanks a lot for the feedback.

Reply



ahmad on June 21, 2020 at 4:46 pm

Yes please, can you add dashboard page with visuals and pdf exports?? thanks in advance It will be very useful to explore all AdminLTE functionality and plugins.

Reply



Ehab on June 22, 2020 at 7:04 pm

Thnx alot, need for sub admin explanation.

Reply



Aijay on June 23, 2020 at 4:59 am

Thanks so much for sharing this educative article.

I look forward to reading role based authentication in .net core 3.1 from you.

Keep it coming because you're doing a good job.

Reply



Mukesh Murugan on June 23, 2020 at 1:01 pm

Thanks, Will be putting out the Role based authentication by the weekend.

Reply



Rogerio on June 23, 2020 at 12:20 pm

Very well crafted and useful tutorial. Thank you sir.

Reply



Mukesh Murugan on June 23, 2020 at 1:00 pm

Thanks Rogerio! Regards

Reply



George on June 24, 2020 at 11:03 am

Thank you sir. Your article was very helpful.

Reply



Mukesh Murugan on June 24, 2020 at 12:33 pm

Thanks a lot for your feedback 😊

Reply



Robinson on June 24, 2020 at 9:00 pm

Exelente, lo aplique a net core con paginas razor, todo funciona bien

Reply



Yishma'el on June 26, 2020 at 6:39 am

Excellent work and keep it up.can you do a project of CRUD operations with sweetAlerts integrated in it .
Especially confirmation to allow user to delete records or not.

Reply



Mukesh Murugan on June 26, 2020 at 12:44 pm

Hi,
Yes,I am already working on it. Will publish the article in a day or 3.

Thanks and Regards

Reply



Scott Trapp on June 26, 2020 at 12:21 pm

Great article. I just recently integrated the AdminLTE with .NET core, but I used Razor Pages instead of MVC. Awesome job.

Reply



Mukesh Murugan on June 26, 2020 at 12:45 pm

Hi,
Quite the same steps yeah?
Thanks and Regards

Reply



sky on June 29, 2020 at 9:23 pm

why dont you start youtube channel and tech step wise. that will be better. i guess.

Reply



Mukesh Murugan on June 30, 2020 at 2:33 pm

Hi, Yes. Ultimately my goal is to have all the tutorials over at Youtube as well. For now I am just preparing the content on my blogs which I would later improvise and use for the Youtube Videos.

Thanks and Regards.

Reply



Cleisson on July 12, 2020 at 5:03 pm

Could you tell me why you gave this error: Build: The module `"/FilterizrOptions/defaultOptions"` has no exported member `'RawOptionsCallbacks'`. Did you mean `'import RawOptionsCallbacks from "/FilterizrOptions/defaultOptions" instead?`

after I tried to do the display inclusion

Reply

Mukesh Murugan on July 12, 2020 at 7:32 pm



With AdminLTE 3.0.5, the filterizr plugin for some reason still has it's own source code files, which might cause TypeScript errors in Visual Studio if you try to run the project (in my case, I was getting Build:Module error on .ts files). The solution is either to remove the plugin if you think you won't need it or inside Solution Explorer, delete everything in filterizr folder except for the following files:

filterizr.min.js
jquery.filterizr.min.js
vanilla.filterizr.min.js

Reply



Name *Cleisson on July 12, 2020 at 8:15 pm

thank you my brother !

Reply

Subscribe To My Newsletter

Join my mailing list to be notified about my latest Articles, Tutorials, and Beginner's Guides on .NET Stack using C# and anything else that you may relate to as a developer.

CATEGORIES

[.NET Core](#)[API](#)[ASP.NET Core](#)[ASP.NET Core Security](#)[Blazor](#)[Coding](#)[Design Patterns](#)[Entity Framework Core](#)

TAGS

[.NET Core](#) [API](#) [asp.net core](#) [Blazor Project Structure](#) [C#](#) [CQRS](#) [IDE](#) [Mediator](#) [Serilog](#) [Structured Logging](#)[Visual Studio](#)

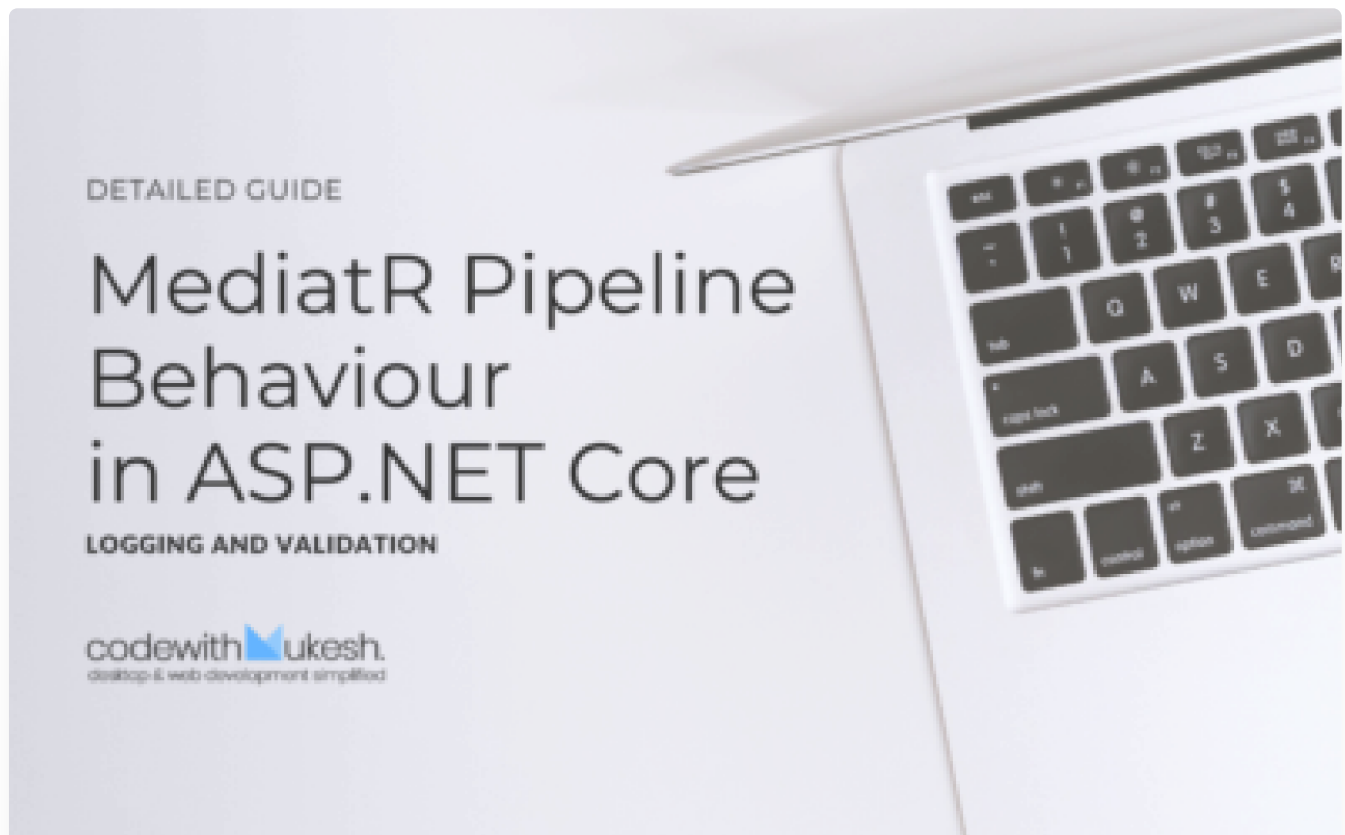
FOLLOW CODEWITHMUKESH



SUPPORT ME!

*Buy me a coffee*

Related Articles



MediatR Pipeline Behaviour in ASP.NET Core – Logging and Validation

by [Mukesh Murugan](#) | Updated on Jul 11, 2020

In one of our previous article, we learnt about CQRS and the usage of the MediatR Library. Now, let's see how to take...

[READ MORE](#)

Globalization and Localization in ASP.NET Core – Detailed

by [Mukesh Murugan](#) | Updated on Jul 5, 2020

In this article, we will go through a less-talked about topic in the ASP.NET Core Community. We will discuss in...

[READ MORE](#)

Custom User Management in ASP.NET Core MVC with Identity

by [Mukesh Murugan](#) | Updated on Jul 4, 2020

In this article, let's go in-depth and understand the functionalities you can achieve with the help of Microsoft...

[READ MORE](#)

Like my Blog Posts?

Join my mailing list to be notified about my latest Articles, Tutorials, and Beginner's Guides on .NET Stack using C# and anything else that you may relate to as a developer.

Name

Email

Subscribe



© codewithmukesh.com 2020 – All rights reserved.