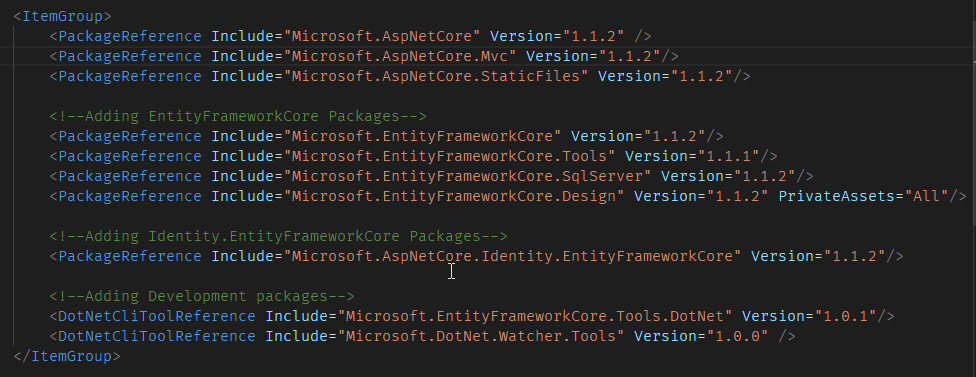
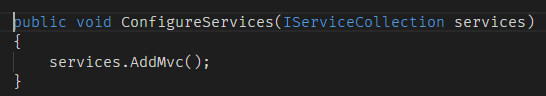
**ASP.NET Core Identity Management**

Steps

1. Create an empty folder and Open visual studio code. Set the folder as the current root for Visual studio. (File >Open Folder)
2. Open the Visual studio code Terminal and create a new “Empty web application” using the command “**dotnet new web**”.
3. It creates an Empty web application project structure.
4. S
5. Add the following packages to the .**csproj** file and restore the packages by running the “**dotnet restore**” command.



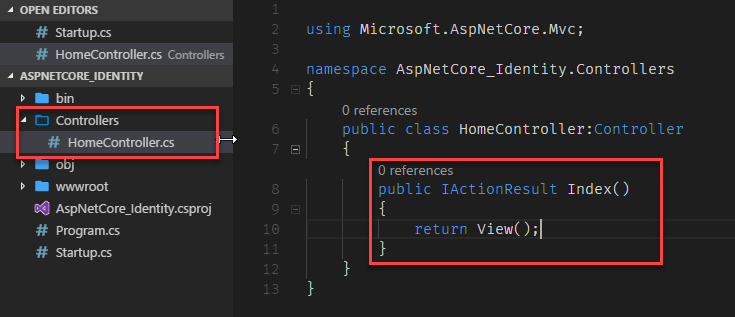
1. Open the Startup.cs and add the Mvc to the Configure services method for IoC configuration.



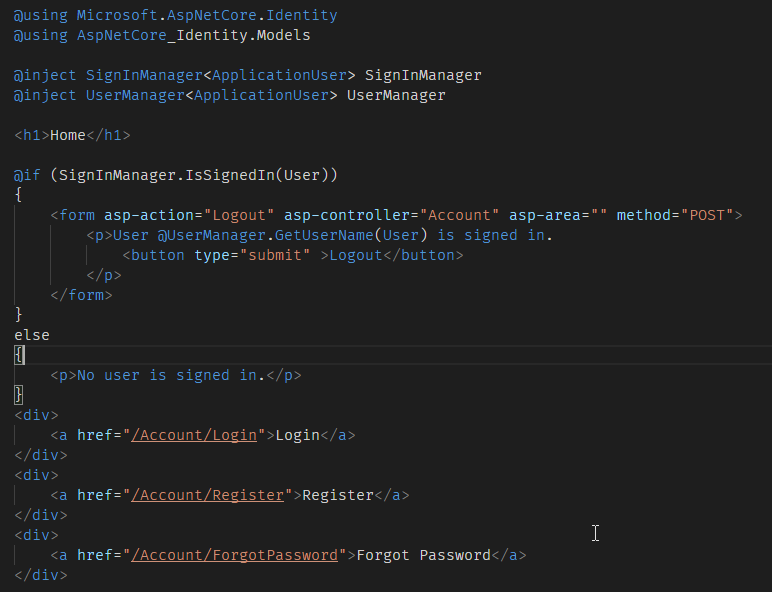
1. Update the Configure method to handle static file requests and MVC requests.



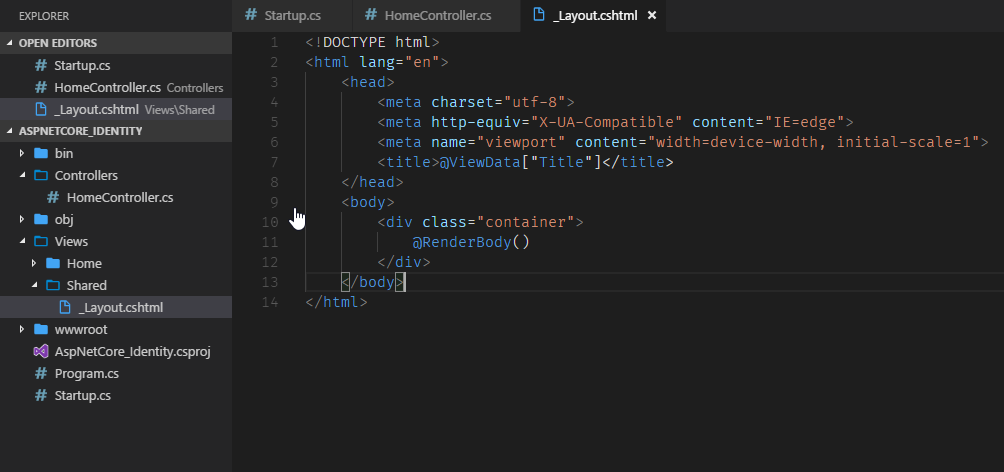
1. Create a new folder in the project root and rename it as “**Controllers**”.
2. Add a new Controller to the ***Controllers*** folder and name it as **HomeController**.
3. Add and **Index** action to the **HomeController**.



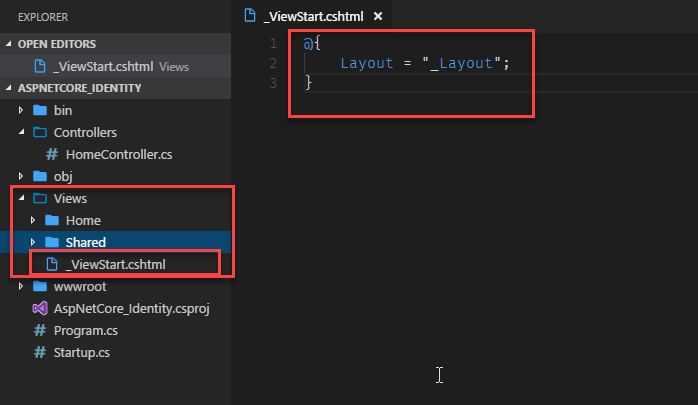
1. Create a new folder in the root and name it as “**Views**”. Also, create two subfolders inside it one with the name “**Home**” and another one as “**Shared**”.
2. Create a view file named “**Index.cshtml**” inside the “**Home**” folder.



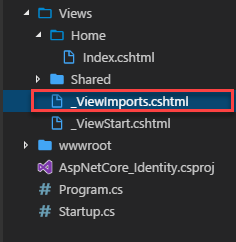
1. Create a view file named **“\_Layout.cshtml**” inside the “**Shared**” folder. Add the following code inside it.

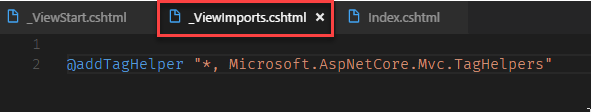


1. Add a new file named **“\_ViewStart.cshtml**” inside the **Views** folder. Add the following code inside the file.



1. Add one more file to the **Views** folder named **“\_ViewImports.cshtml**”. Add the following code to the file.

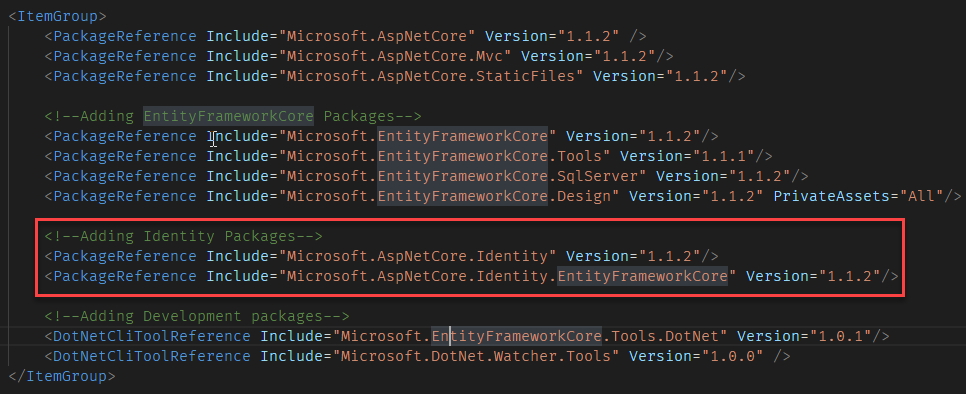




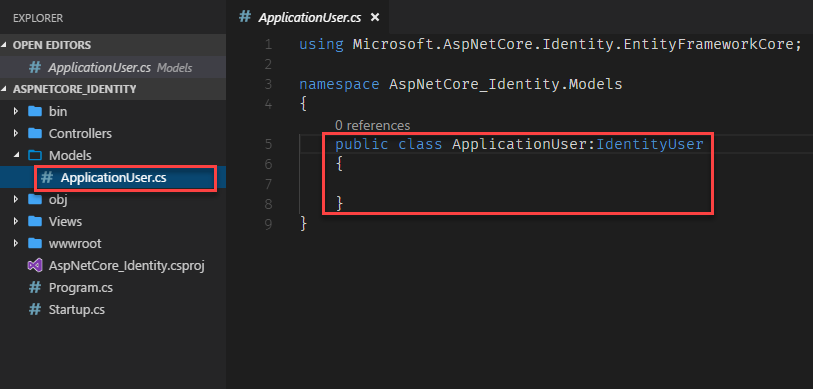
1. Add the identity package to the .csproj file. Add the following packages

Microsoft.AspNetCore.Identity : 1.1.2

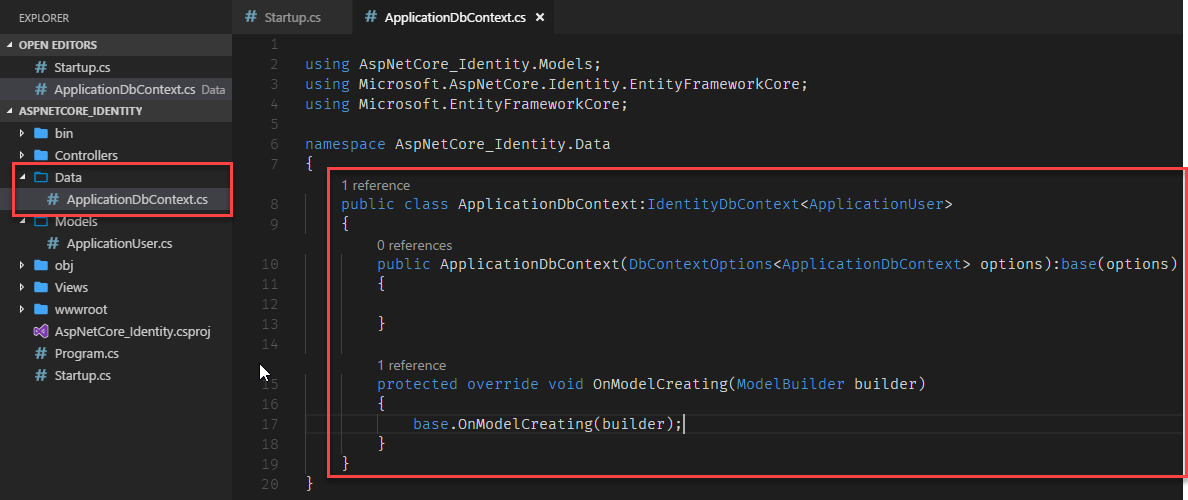
Microsoft.AspNetCore.Identity.EntityFrameworkCore:1.1.2



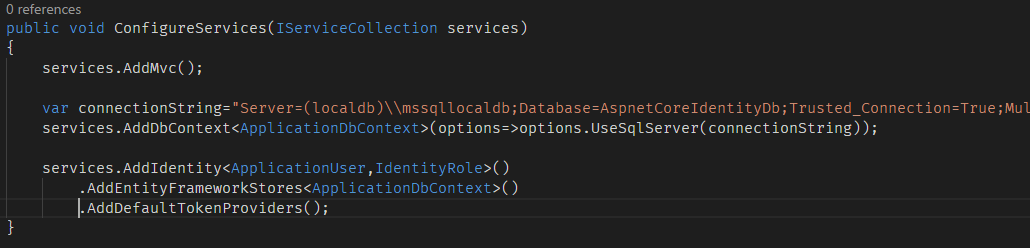
1. A user will have (at least) all the properties contained in a class named IdentityUser. You can create a subclass of IdentityUser and add more properties. Those will be present in the user’s table when we actually create the database. The common way to use Identity assumes you will be inheriting from a set of base classes, namely IdentityUser, so that you can specify what extra information you want to save about your users. Also, IdentityDbContext, which your own application’s DbContext is supposed to inherit from and then specify your own extra DbSets.
2. Create a new folder called “**Models**” in the root folder. Add a new class called “**ApplicationUser**” by inheriting the base class “**IdentityUser**”.



1. Add a new folder in the root folder named “**Data**” and add a new class called “**ApplicationDbContext**” by inheriting “**IdentityDbContext**” class.



1. Now we need to update the **Startup** class for IoC configuration for Identity management. Add the following code in the **ConfigureServices()** method.

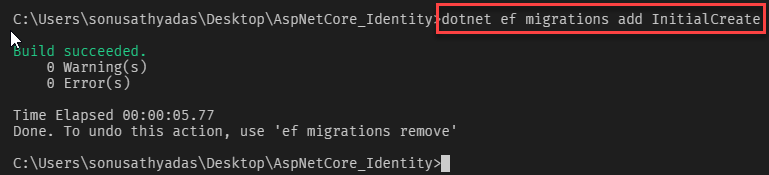


1. Next, we need to update the request pipeline. Add the Identity middleware in the request pipeline by calling app.UseIdentity() method. We need to call the Identity middleware before service static files or executing MVC actions.

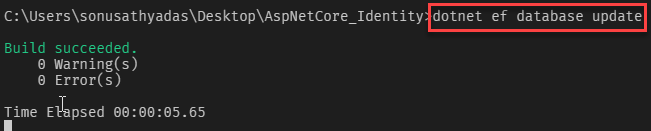


1. Enable database code first migrations by running the following command.

**dotnet ef migrations add InitialCreate**



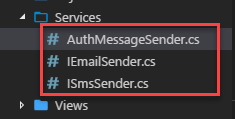
1. It creates a migrations folder and migration file. Now you need to generate the database tables. Run the command “**dotnet ef database update**” to generate database.



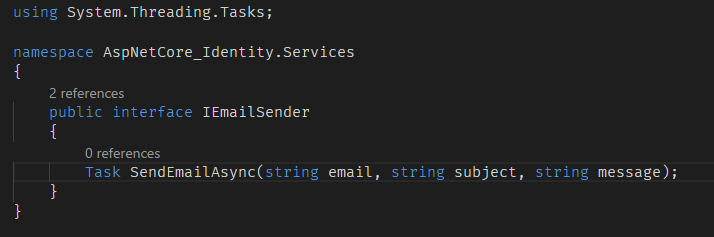
1. You can run the project in development mode by using the command

**dotnet run --environment "Development”**

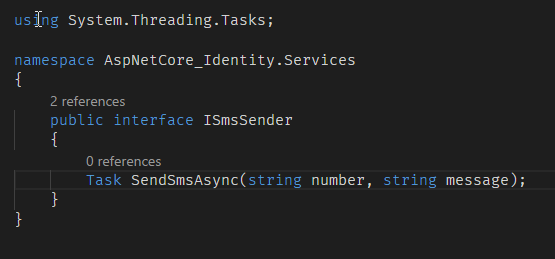
1. Create a new folder called “**Services**” and add to interfaces for Sending Sms and Email verification messages. Also, add a class to implement both the interfaces.



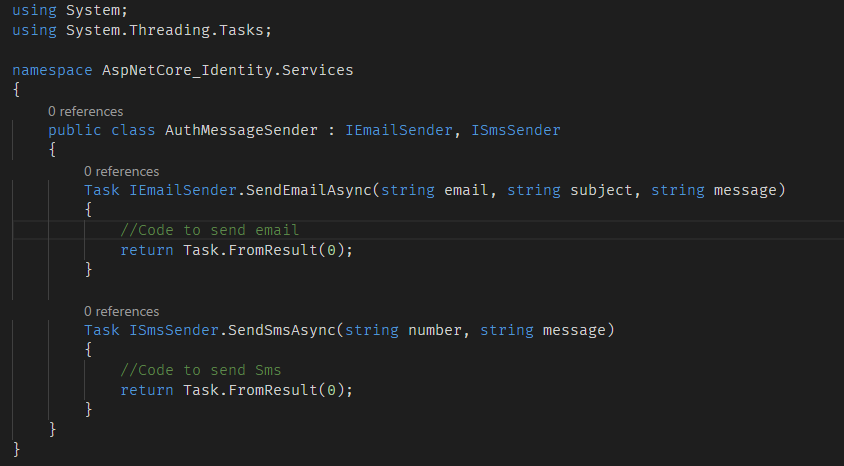
1. Add the following code to the “**IEmailSender**” interface.



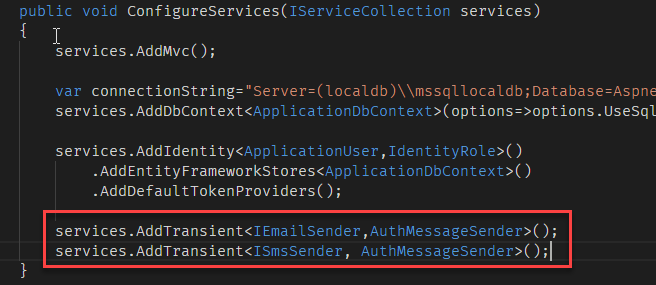
1. Add the following code to “**ISmsSender**” interface



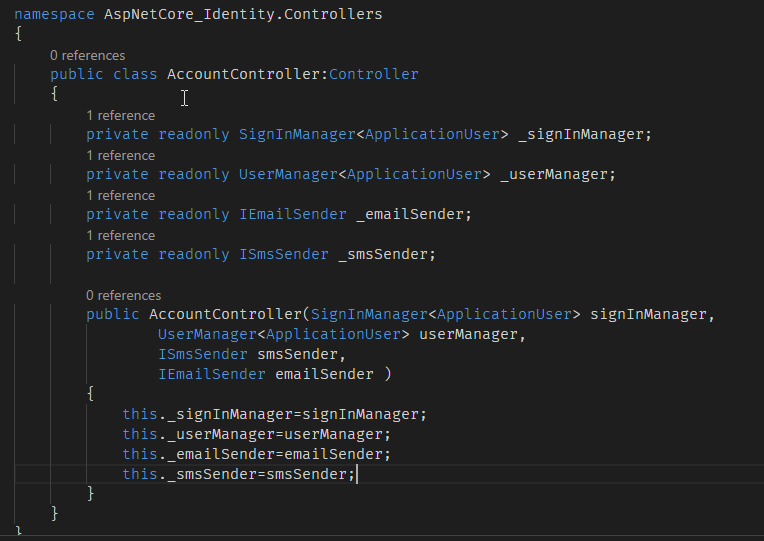
1. Add the following code to the “**AuthMessageSender**” class



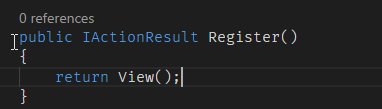
1. Register the EmailService and SmsService implementations in Startup class.



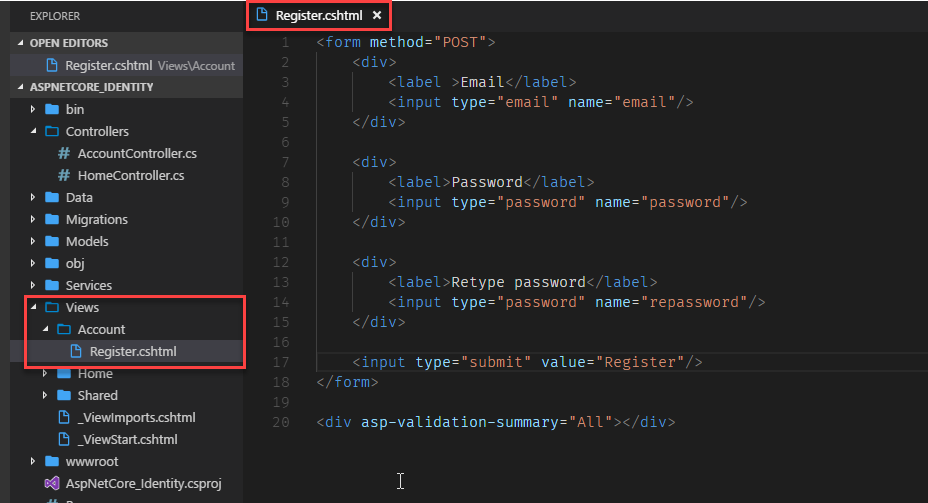
1. Create an **AccountController** in the **Conrollers** folder and Inject the **SignInManager**, **UserManager**, **IEmailSender** and **ISmsSender** services to the constructor.



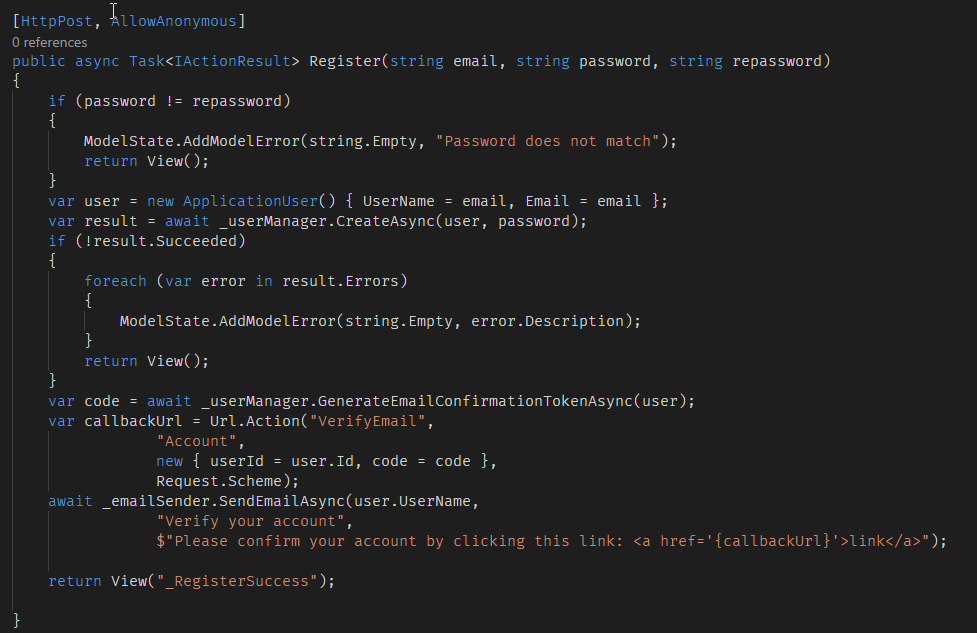
1. Now, you need to create a user registration page. For that, add an action to the **AccountController** and name it as Register.



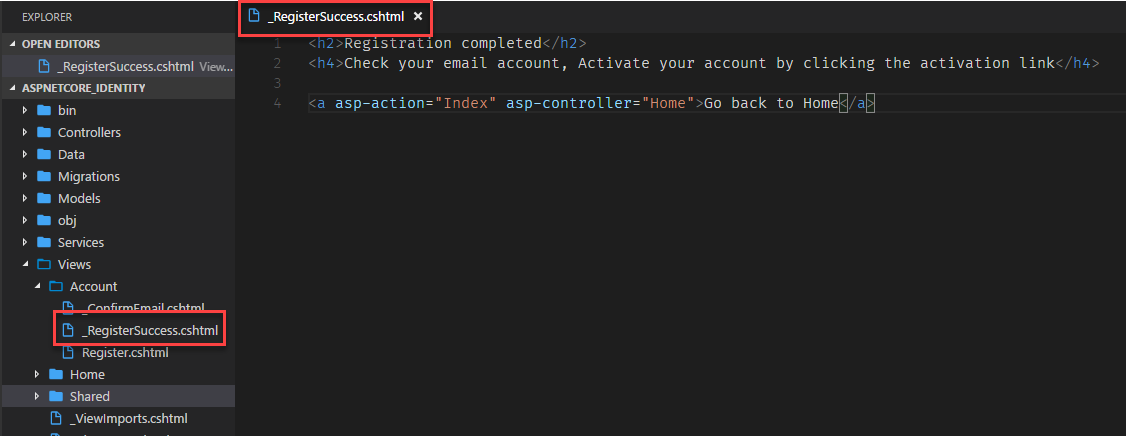
1. Create a registration View for the **Register** action. To do so, Create a new folder under **Views** folder and name it as “**Account**”. Create a view page named “**Register.cshtml**” inside the **Account** views folder.



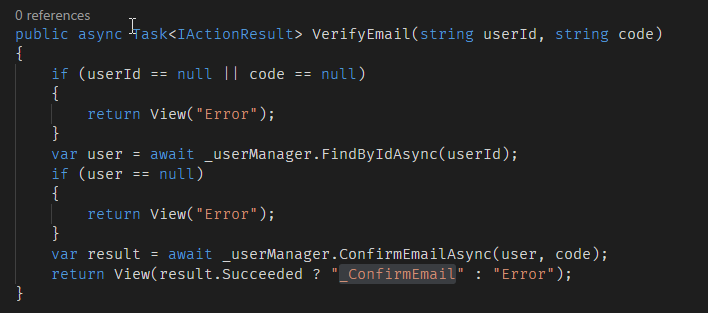
1. We are now ready to handle the submitted form values. To handle the POST request, create an **[HttpPost] Register** action in the **AccountController**.



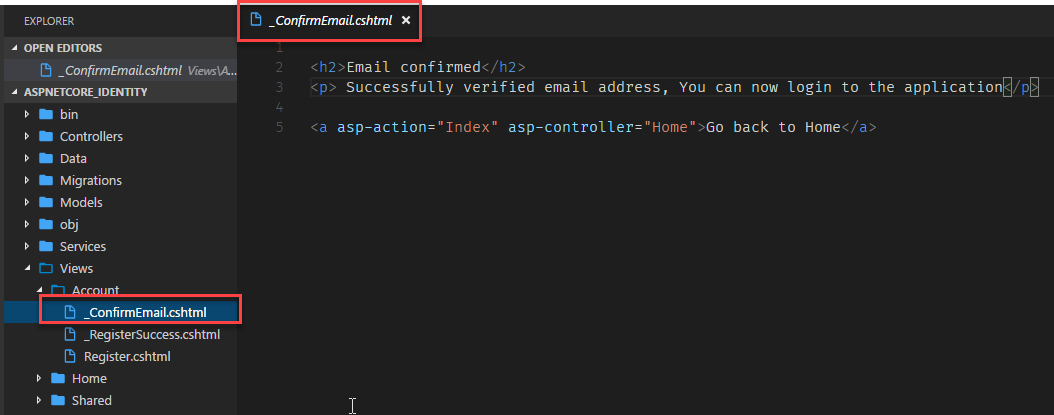
The above POST action accepts the submitted values and creates a new user. If there is any error, it returns back to the Register view. If registration is success, it send an activation mail to the users email Id. After sending the email, it redirects to the **RegisterSuccess** view. For that you need to create a **\_RegisterSuccess.cshtml** in the Account view folder. When the user clicks on the activation link send to you email id it will be send a request to **VerifyEmail** action in the **AccountController**. So, you need to write a **VerifyEmail** action also inside the **AccountController**.

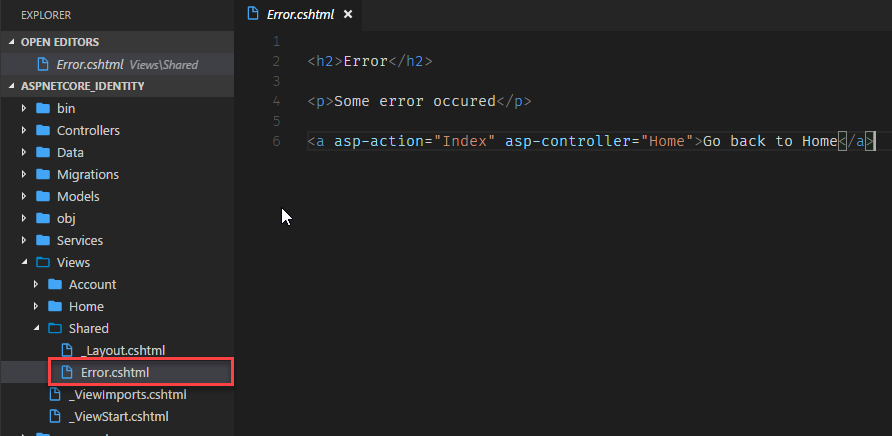


1. Add the following code to the VerifyEmail action.

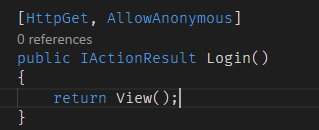


Once the email verification is completed successfully, it redirects the user to the **\_ConfirmEmail.cshtml** view. In case of error, it returns the **Error** view. Error view is in the **Shared** views folder.

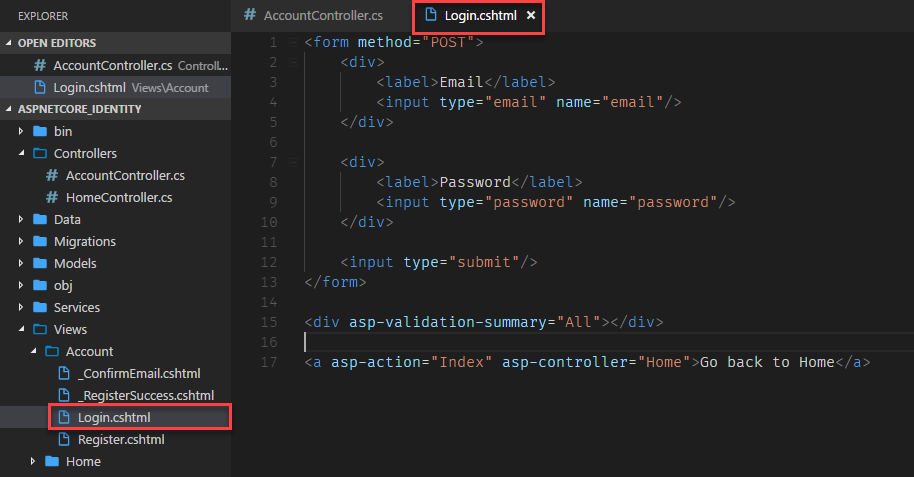




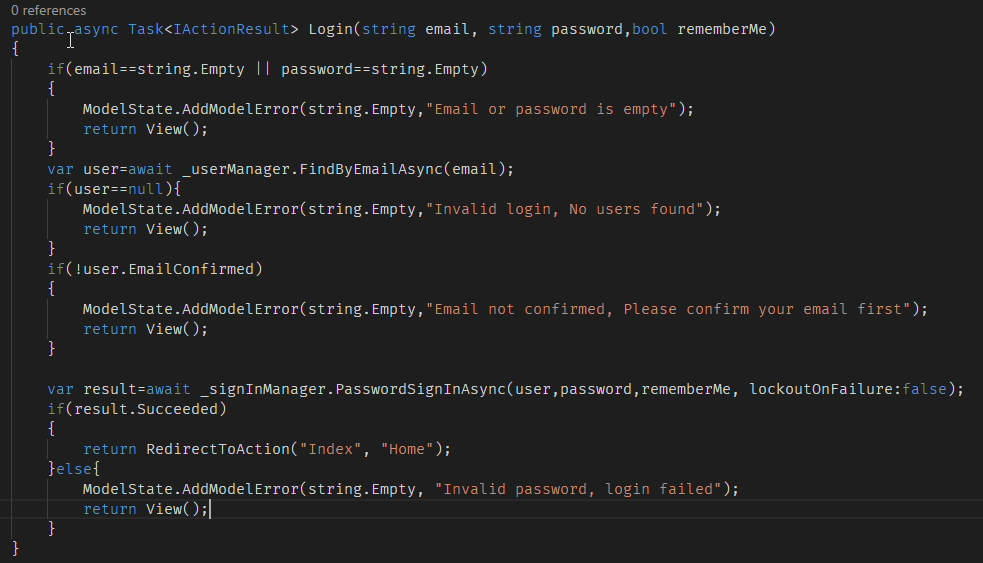
1. Now, we can create sign in page for the user. Add a Login action to the AccountController.



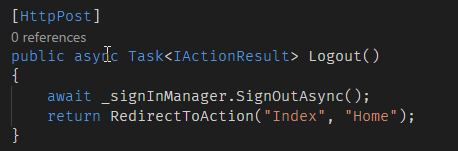
1. Create **Login.cshtml** in the **Account** views folder and add the following code.



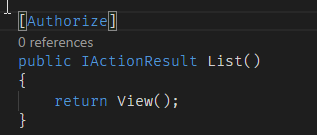
1. Write a HttpPost action for the Login page. Add a POST Login action in the AccountController.

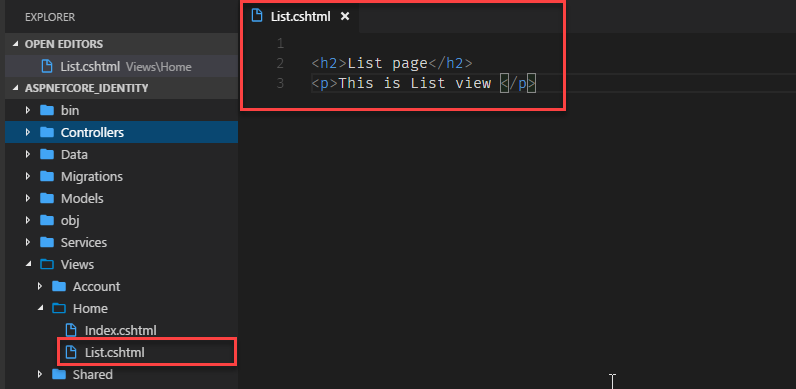


1. Run the code and test the Register and Sign In actions.
2. Now we can add the Logout functionality to the AccountController. Add an action named Logout.

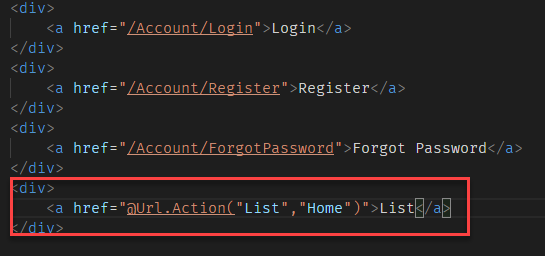


1. Add a new action “List” to the Home controller and set the [Authorize] attribute to the action. Also create a View –“List.cshtml” in the Home views folder.

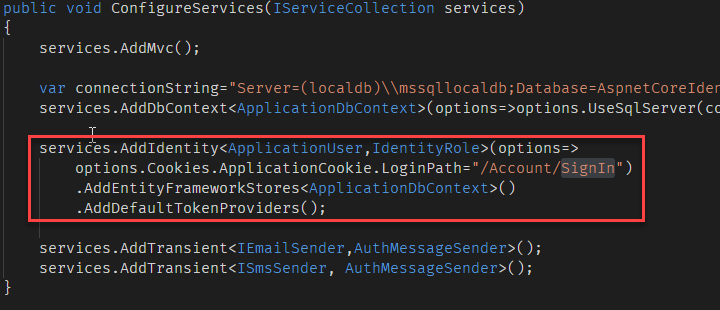




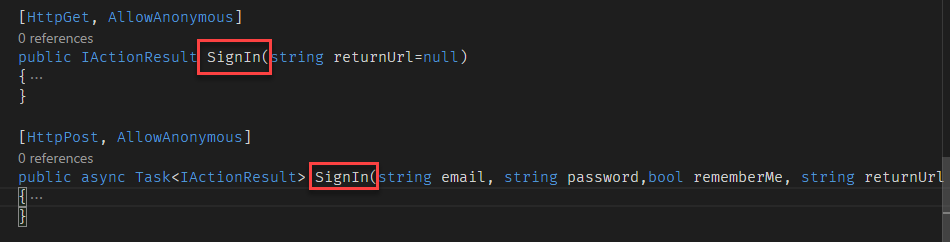
1. Update the **Index** view and add a new hyperlink to the List action. When you click on the link, If the user is not signed in it redirects to the Login view.



1. If the Login method name is something different from “**Login**”, ie: “**SignIn**” then this will not work, because the default Login action configuration in Identity management is set as “**Account/Login**”. If you change the controller or action name, it will not redirect to the **SignIn** action which you created, when the user is not authorized to access and action, it always redirect to “**Accout/Login**” only.
2. To override this default configuration, while registering the Identity in the ConfigurationServices() method of the Startup class, you also need to specify the login path.



1. Now you can change the Logic GET and POST action names to “SignIn” and test.



1. Don’t forget to rename the view as “**SignIn.cshtml**”.
2. It is also possible to configure the Password policy. You can configure it in the **ConfigureServices()** method in **Startup** class.

