Cross Site Scripting (XSS) in ASP .NET Core

## **What is Cross Site Scripting**

Cross site scripting is the injection of malicious code in a web application, usually, Javascript but could also be CSS or HTML. When attackers manage to inject code into your web application, this code often gets also saved in a database. This means every user could be affected by this. For example, if an attacker manages to inject Javascript into the product name on Amazon. Every user who opens the infected product would load the malicious code.

## **Consequences of XSS Attacks**

There are many possible consequences for your users if your website got attacked by cross site scripting:

* Attackers could read your cookies and therefore gain access to your private accounts like social media or bank
* Users may be redirected to malicious sites
* Attackers could modify the layout of the website to lure users into unintentional actions
* Users could be annoyed which will lead to damage to your reputation and probably a loss of revenue
* Often used in combination with other attacks like [cross site request forgery (CSRF)](https://www.programmingwithwolfgang.com/cross-site-request-forgery-csrf-in-asp-net-core/)

## **Best Practices against Cross Site Scripting Attacks**

Preventing XSS attacks is pretty simple if you follow these best practices:

* Validate every user input, either reject or sanitize unknown character, for example, < or > which can be used to create
* Test every input from an external source
* Use HttpOnly for cookies so it is not readable by Javascript (therefore an attacker can’t use Javascript to read your cookies)
* Use markdown instead of HTML editors

## **Cross Site Scripting in ASP .NET Core**

ASP .NET Core Is already pretty safe out of the box due to automatically encoding HTML, for example < gets encoded into &lt. Let’s have a look at two examples where XSS attacks can happen and how to prevent them. You can find the code for the demo on [GitHub](https://github.com/WolfgangOfner/MVC-XssDemo).

### **ASP .NET Core 3.1 Demo**

XSS can occur when you display text which a user entered. ASP .NET Core automatically encodes text when you use @Model, but displays the code as it if if you use @Html.Raw.

#### **Preventing XSS Attacks in forms**

The following code creates a form where the user can enter his user name. The input is displayed once in a safe way and once in an unsafe way.

|  |  |
| --- | --- |
|  | @model Customer |
|  |  |
|  | <div asp-validation-summary="All"></div> |
|  |  |
|  | <form asp-action="Index"> |
|  | <div class="form-group"> |
|  | <label asp-for="UserName">Please enter your user name</label> |
|  | <input type="text" class="form-control" asp-for="UserName" value="User"> |
|  | </div> |
|  | <button type="submit" class="mt-md-1 btn btn-primary">Submit</button> |
|  | </form> |
|  |  |
|  | <br /> |
|  |  |
|  | @if (!string.IsNullOrEmpty(Model.UserName)) |
|  | { |
|  | <div class="row"> |
|  | <p>Safe output: @(Model.UserName)</p> |
|  | </div> |
|  | <div class="row"> |
|  | <p>Unsafe output: @Html.Raw(Model.UserName)</p> |
|  | </div> |
|  | } |

[**view raw**](https://gist.github.com/WolfgangOfner/d2514af5d378090253a61e9ad4e0d8f2/raw/e6af1af04f05e79db0da35398f4262d7c564bae2/Customer.cs)[**Customer.cs**](https://gist.github.com/WolfgangOfner/d2514af5d378090253a61e9ad4e0d8f2#file-customer-cs)hosted with ❤ by [**GitHub**](https://github.com/)

When a user enters his user name everything is fine. But when an attacker enters Javascript, the Javascript will be executed when the text is rendered inside the unsafe output

tag. When you enter the following code as your name:

|  |  |
| --- | --- |
|  | <script>alert(&#8216;attacked&#8217;)</script> |

[**view raw**](https://gist.github.com/WolfgangOfner/80127fb2310a34bcf45e9dad758b99fb/raw/4c88af2c53a9ea064b403d149a84fa20bd4f9b6b/Script.cshtml)[**Script.cshtml**](https://gist.github.com/WolfgangOfner/80127fb2310a34bcf45e9dad758b99fb#file-script-cshtml)hosted with ❤ by [**GitHub**](https://github.com/)

and click submit, an alert windows will be displayed.

The injected code got executed

When you click on OK, the text will be rendered into the safe output line and nothing will be displayed in the unsafe output line because the browser interprets the Javascript.

The Javascript is displayed as text in the safe output line

#### **Preventing XSS Attacks in Query Parameters**

Another way to inject code is through query parameters. If your application ready query parameters but doesn’t sanitize them, Javascript in it will be executed. The following code contains two forms. When you click on the button a query parameter will be read and printed to an alert box.

|  |  |
| --- | --- |
|  | @using System.Text.Encodings.Web |
|  |  |
|  | @model Customer |
|  | @inject JavaScriptEncoder JavaScriptEncoder |
|  |  |
|  | <h2>Unsafe Javascript</h2> |
|  | <form asp-action="JavascriptAttack"> |
|  | @Html.HiddenFor(m => m.UserId) |
|  | <div class="form-group"> |
|  | <label for="userName">Please enter your user id</label> |
|  | <input type="text" class="form-control" id="userName" name="userName" value="User"> |
|  | </div> |
|  | <button type="submit" class="mt-md-1 btn btn-primary" onclick="alert('Saving user name for account with id: @Context.Request.Query["userId"]');">Submit</button> |
|  | </form> |
|  | <br /> |
|  | <h2>Safe Javascript</h2> |
|  | <form asp-action="JavascriptAttack"> |
|  | @Html.HiddenFor(m => m.UserId) |
|  | <div class="form-group"> |
|  | <label for="userName2">Please enter your user id</label> |
|  | <input type="text" class="form-control" id="userName2" name="userName2" value="User"> |
|  | </div> |
|  | <button type="submit" class="mt-md-1 btn btn-primary" onclick="alert('Saving user name for account with id: @JavaScriptEncoder.Encode(Context.Request.Query["UserId"])');">Submit</button> |
|  | </form> |

[**view raw**](https://gist.github.com/WolfgangOfner/c4a9259eba12d4c178f19112a13dd549/raw/bf212bf9351e9850ae0cad991bab9f47ca249538/Form.cs)[**Form.cs**](https://gist.github.com/WolfgangOfner/c4a9259eba12d4c178f19112a13dd549#file-form-cs)hosted with ❤ by [**GitHub**](https://github.com/)

The first submit button will execute Javascript whereas the second one uses the JavaScriptEncode to encode the text first. To simulate an attack replace the value of the UserId with the following code and click enter:

|  |  |
| --- | --- |
|  | %27);alert(&#8216;You got attacked&#8217;);// |

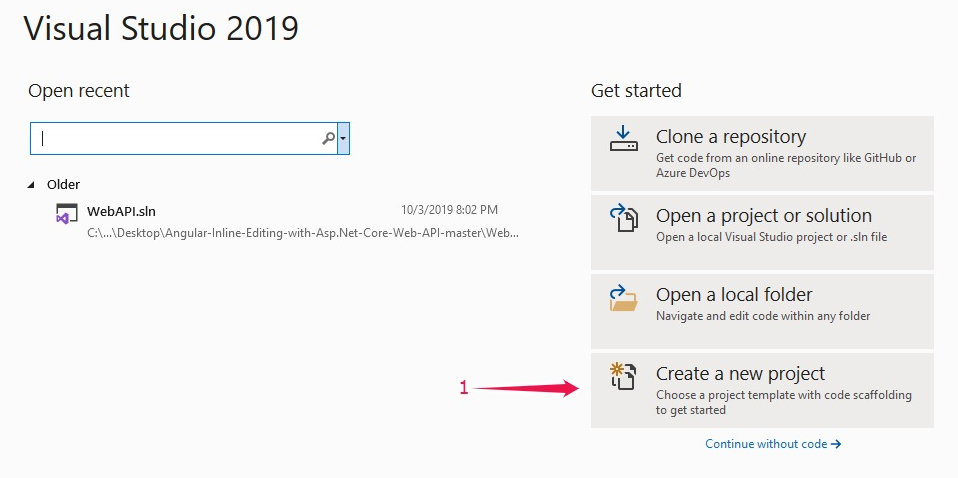
[**view raw**](https://gist.github.com/WolfgangOfner/bbe6a8a4e42bf2bae47b4a98ad3188d3/raw/f69e93a43831c8d8b8ef0e036b2ec35a379639ba/Alert.js)[**Alert.js**](https://gist.github.com/WolfgangOfner/bbe6a8a4e42bf2bae47b4a98ad3188d3#file-alert-js)hosted with ❤ by [**GitHub**](https://github.com/)

Click the submit button of the unsafe form and you will see two Javascript alerts. The first one saying “Saving user name for account with id: ” and then a second one saying “You got attacked”.

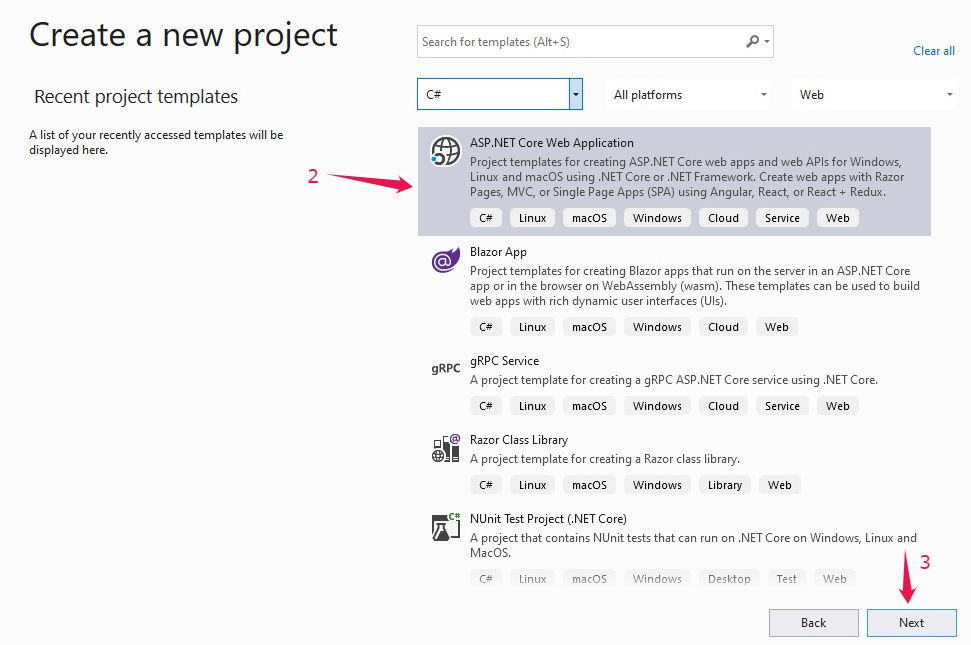
# Using Identity In ASP.NET Core MVC Authentication

**Step 1**

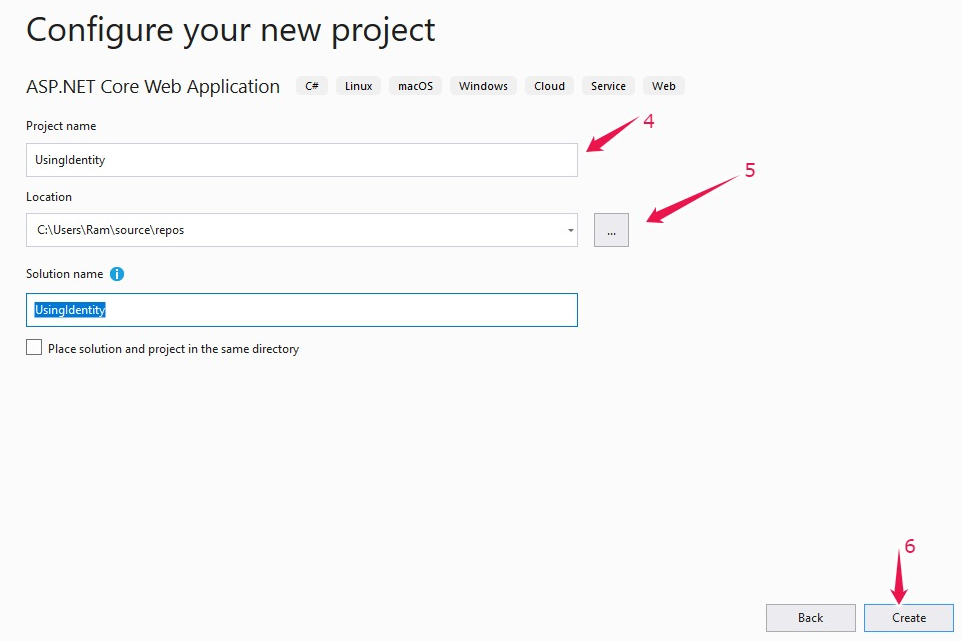
Let's create an ASP.NET Core web application. Open Visual Studio 2019 and click on Create a new project.



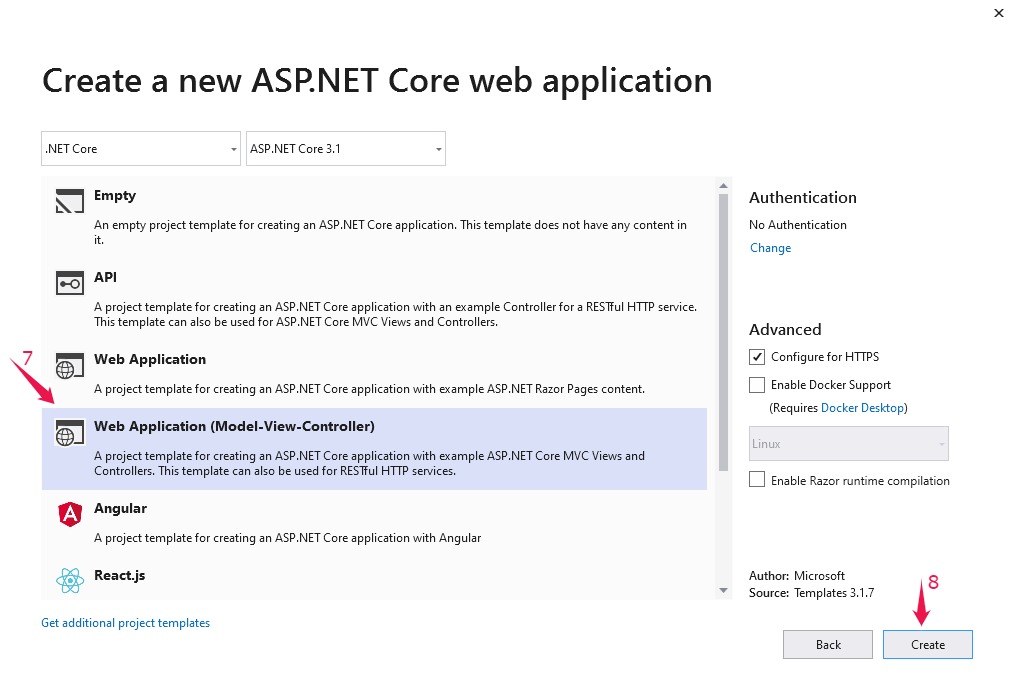
Select ASP.NET Core Web Application, and click on Next



Provide the project name of your choice, select the preferred location & click on Create

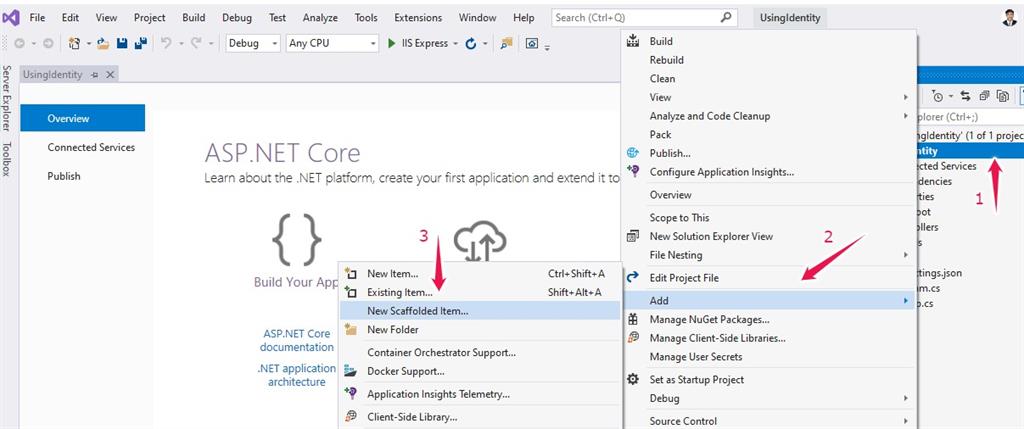


Select MVC Template and click on create, as shown below:

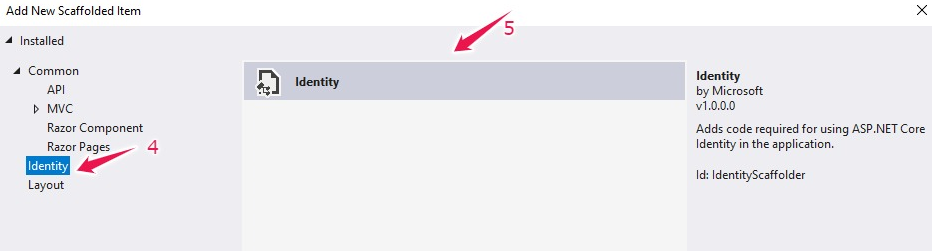


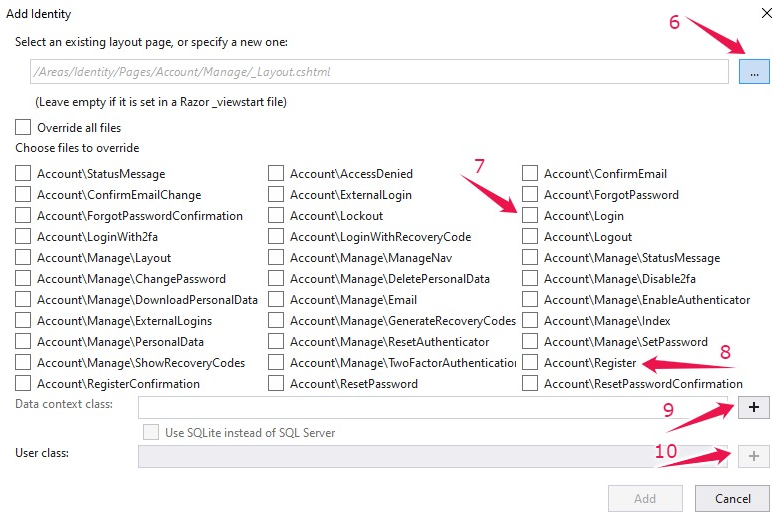
**Step 2**

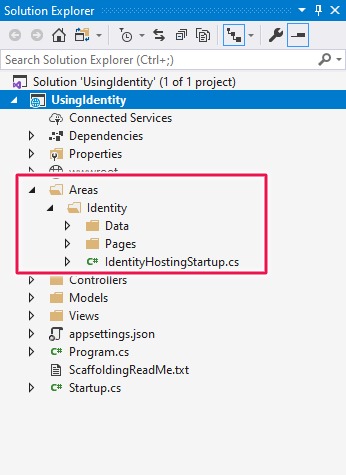
Now let's add an ASP.NET Core Identity. Select the project > Right-click > Add > click on New Scaffold item



Select Identity and click on add



Now select the layout page, as we want authentication. Let's select login and Register as shown below and provide DbContext class and user class and click on Add  
  


We can find the Areas in the application with Data & Razor pages, as shown below:  
  


**Step 3**

Now let's add user authentication to the application. Open the Startup class and Modify as shown below.  
  
In order to add support to the razor pages, we have to call the function services.AddRazorPages() and endpoints.MapRazorPages()

1. **public** **class** Startup
2. {
3. **public** Startup(IConfiguration configuration)
4. {
5. Configuration = configuration;
6. }
8. **public** IConfiguration Configuration { **get**; }
10. // This method gets called by the runtime. Use this method to add services to the container.
11. **public** **void** ConfigureServices(IServiceCollection services)
12. {
13. services.AddControllersWithViews();
14. services.AddRazorPages();
15. }
17. // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
18. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
19. {
20. **if** (env.IsDevelopment())
21. {
22. app.UseDeveloperExceptionPage();
23. }
24. **else**
25. {
26. app.UseExceptionHandler("/Home/Error");
27. // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
28. app.UseHsts();
29. }
30. app.UseHttpsRedirection();
31. app.UseStaticFiles();
33. app.UseRouting();
34. app.UseAuthentication();
36. app.UseAuthorization();
38. app.UseEndpoints(endpoints =>
39. {
40. endpoints.MapControllerRoute(
41. name: "default",
42. pattern: "{controller=Home}/{action=Index}/{id?}");
43. endpoints.MapRazorPages();
44. });
45. }
46. }

**Step 4**

Now let's start with creating the database for the application.

Open UsingIdentityUser class and add the properties and decorate with the attribute PersonalData.

1. **public** **class** UsingIdentityUser : IdentityUser
2. {
3. [PersonalData]
4. [Column(TypeName ="nvarchar(100)")]
5. **public** **string** Firstname { **get**; **set**; }
6. [PersonalData]
7. [Column(TypeName = "nvarchar(100)")]
8. **public** **string** LastName { **get**; **set**; }
9. }

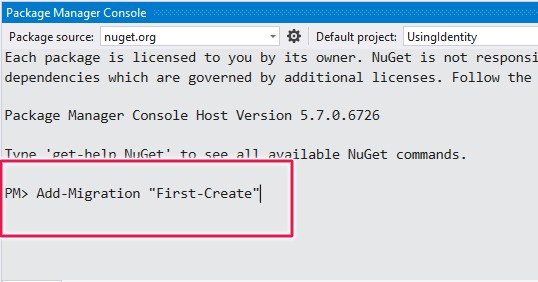
We have Dbcontext, which is also inherited from the parent class IdentityDB context. This identity db context is injected inside this identityhostingstartup class.

1. **public** **class** IdentityHostingStartup : IHostingStartup
2. {
3. **public** **void** Configure(IWebHostBuilder builder)
4. {
5. builder.ConfigureServices((context, services) => {
6. services.AddDbContext<UsingIdentityContext>(options =>
7. options.UseSqlServer(
8. context.Configuration.GetConnectionString("UsingIdentityContextConnection")));
10. services.AddDefaultIdentity<UsingIdentityUser>(options => options.SignIn.RequireConfirmedAccount = **true**)
11. .AddEntityFrameworkStores<UsingIdentityContext>();
12. });
13. }
14. }

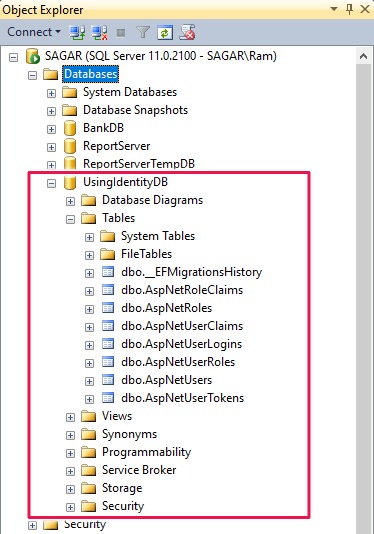
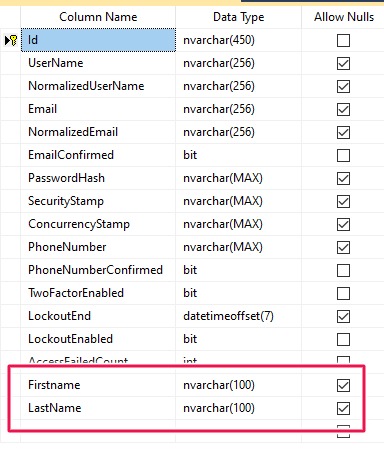
Now open appsttings.json. We can find the connection string with the name UsingIdentityContextConnection. By default, it will connect to the local DB.

1. {
2. "Logging": {
3. "LogLevel": {
4. "Default": "Information",
5. "Microsoft": "Warning",
6. "Microsoft.Hosting.Lifetime": "Information"
7. }
8. },
9. "AllowedHosts": "\*",
10. "ConnectionStrings": {
11. "UsingIdentityContextConnection": "Server=SAGAR;Database=UsingIdentityDB;Trusted\_Connection=True;MultipleActiveResultSets=true"
12. }
13. }

Now open the Package manager console and Execute the command Add-Migration "First-Create" to generate the actual physical DB

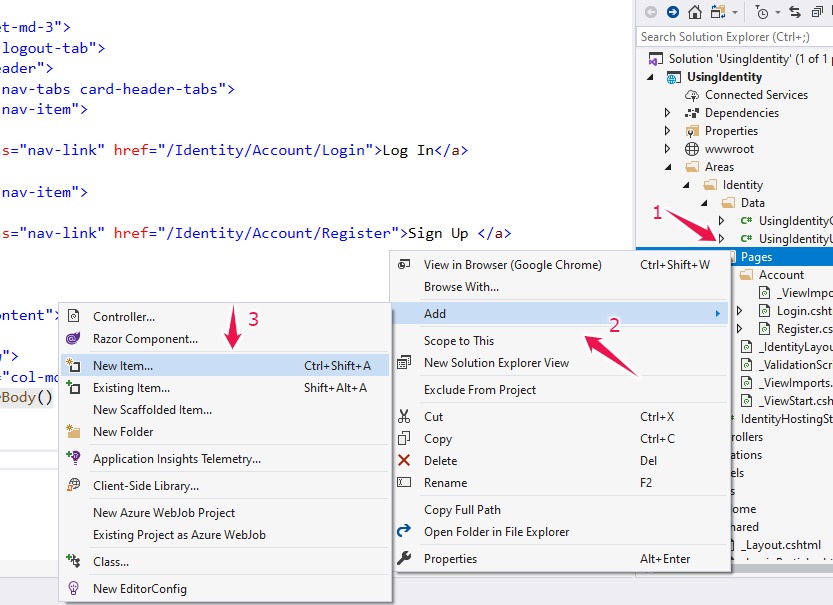


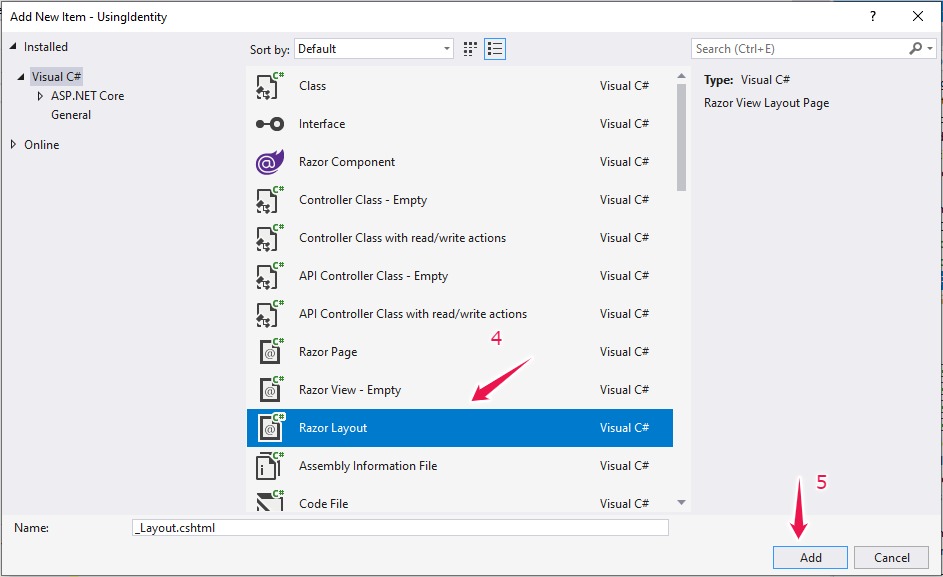
Finally, execute the command Update-Database and you can find the new database as shown below.

**Step 5**

Now let's start customizing the application. Let's add a Nested layout for tab control headers.



Select Razor layout and click on Add  
  
  
  
Customize the layout as below

1. @{
2. Layout = "/Views/Shared/\_Layout.cshtml";
3. }
4. <div **class**="row">
5. <div **class**="col-md-6  offset-md-3">
6. <div **class**="card login-logout-tab">
7. <div **class**="card-header">
8. <ul **class**="nav nav-tabs card-header-tabs">
9. <li **class**="nav-item">
11. <a **class**="nav-link" href="/Identity/Account/Login">Log In</a>
12. </li>
13. <li **class**="nav-item">
15. <a **class**="nav-link" href="/Identity/Account/Register">Sign Up </a>
16. </li>
17. </ul>
18. </div>
19. <div **class**="card-content">
21. <div **class**="row">
22. <div **class**="col-md-12 ">
23. @RenderBody()
24. </div>
25. </div>
26. </div>
28. </div>
29. </div>
30. </div>
31. @section Scripts{
33. @RenderSection("Scripts", required: **false**)
34. <script>
35. $(**function** () {
37. **var** current = location.pathname;
38. $('.nav-tabs li a').each(**function** () {
39. **var** $**this** = $(**this**);
40. **if** (current.indexOf($**this**.attr('href')) !== -1) {
41. $**this**.addClass('active');
42. }
43. })
44. })
45. </script>
46. }

Open site.css and use the below styles

1. /\*for Tab control\*/
2. div.login-logout-tab div.card.header{
3. **padding**: 0px 0px 12px 0px;
4. }
5. div.login-logout-tab li.nav-tabs{
7. **margin**: 0px 0px -12px 0px;
8. }
9. div.login-logout-tab li.nav-item{
11. **width**:50%
12. }
13. div.login-logout-tab a.nav-link{
14. **font-size**:25px;
15. **color**:#495057;
16. **text-align**:center;
17. }

Modify the Login page and register the HTML pages with our new layout.

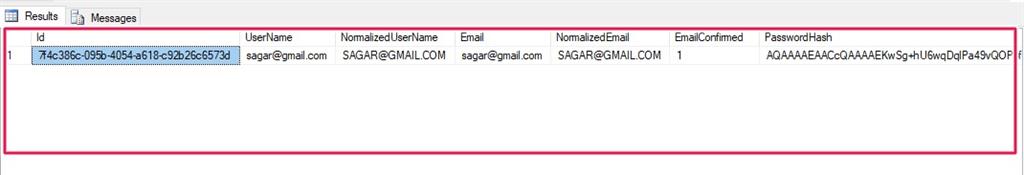
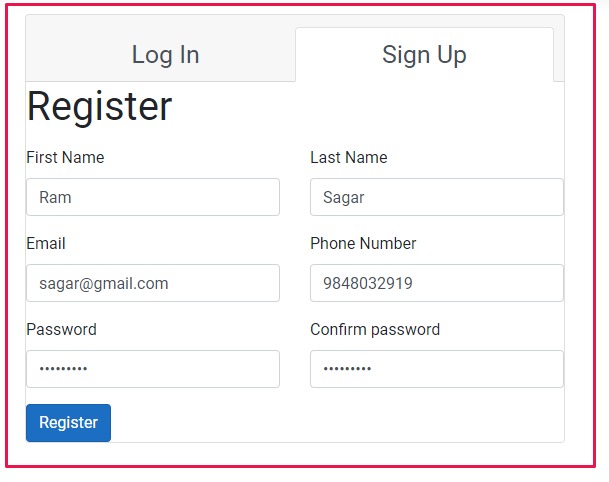
1. @page
2. @model LoginModel
4. @{
5. ViewData["Title"] = "Log in";
6. Layout = "~/Areas/Identity/Pages/\_IdentityLayout.cshtml";
7. }
9. <div **class**="col-md-10 offset-md-1">
10. <section>
11. @\*<div **class**="login-form-icon">
12. <i **class**="fas fa-user-circle fa-5x text-secondary"></i>
13. </div>\*@
14. @\*fas fa-user-circle fa-9x text-secondary\*@
15. <form id="account" method="post">
16. <div asp-validation-summary="All" **class**="text-danger"></div>
17. <div **class**="form-group">
18. <label asp-**for**="Input.Email"></label>
19. <input asp-**for**="Input.Email" **class**="form-control" />
20. <span asp-validation-**for**="Input.Email" **class**="text-danger"></span>
21. </div>
22. <div **class**="form-group">
23. <label asp-**for**="Input.Password"></label>
24. <input asp-**for**="Input.Password" **class**="form-control" />
25. <span asp-validation-**for**="Input.Password" **class**="text-danger"></span>
26. </div>
27. <div **class**="form-group">
28. <div **class**="checkbox">
29. <label asp-**for**="Input.RememberMe">
30. <input asp-**for**="Input.RememberMe" />
31. @Html.DisplayNameFor(m => m.Input.RememberMe)
32. </label>
33. </div>
34. </div>
35. <div **class**="form-group">
36. <button type="submit" **class**="btn btn-primary btn-block">Log **in**</button>
37. </div>
38. </form>
40. </section>
41. </div>
43. @section Scripts {
44. <partial name="\_ValidationScriptsPartial" />
45. }
47. @page
48. @model RegisterModel
49. @{
50. ViewData["Title"] = "Register";
51. Layout = "~/Areas/Identity/Pages/\_IdentityLayout.cshtml";
52. }
54. <h1>@ViewData["Title"]</h1>

57. <form asp-route-returnUrl="@Model.ReturnUrl" method="post">
58. <div asp-validation-summary="All" **class**="text-danger"></div>
59. <div **class**="row">
60. <div **class**="col-md-6">
61. <div **class**="form-group">
62. <label asp-**for**="Input.FirstName"></label>
63. <input asp-**for**="Input.FirstName" **class**="form-control" />
64. <span asp-validation-**for**="Input.FirstName" **class**="text-danger"></span>
65. </div>
66. </div>
68. <div **class**="col-md-6">
69. <div **class**="form-group">
70. <label asp-**for**="Input.LastName"></label>
71. <input asp-**for**="Input.LastName" **class**="form-control" />
72. <span asp-validation-**for**="Input.LastName" **class**="text-danger"></span>
73. </div>
74. </div>
75. </div>
76. <div **class**="row">
77. <div **class**="col-md-6">
78. <div **class**="form-group">
79. <label asp-**for**="Input.Email"></label>
80. <input asp-**for**="Input.Email" **class**="form-control" />
81. <span asp-validation-**for**="Input.Email" **class**="text-danger"></span>
82. </div>
83. </div>
84. <div **class**="col-md-6">
85. <div **class**="form-group">
86. <label asp-**for**="Input.PhoneNumber"></label>
87. <input asp-**for**="Input.PhoneNumber" **class**="form-control" />
88. <span asp-validation-**for**="Input.PhoneNumber" **class**="text-danger"></span>
89. </div>
90. </div>
91. </div>
93. <div **class**="row">
94. <div **class**="col-md-6">
95. <div **class**="form-group">
96. <label asp-**for**="Input.Password"></label>
97. <input asp-**for**="Input.Password" **class**="form-control" />
98. <span asp-validation-**for**="Input.Password" **class**="text-danger"></span>
99. </div>
100. </div>
101. <div **class**="col-md-6">
102. <div **class**="form-group">
103. <label asp-**for**="Input.ConfirmPassword"></label>
104. <input asp-**for**="Input.ConfirmPassword" **class**="form-control" />
105. <span asp-validation-**for**="Input.ConfirmPassword" **class**="text-danger"></span>
106. </div>
107. </div>
108. </div>
110. <button type="submit" **class**="btn btn-primary">Register</button>
111. </form>
113. @section Scripts {
114. <partial name="\_ValidationScriptsPartial" />
115. }

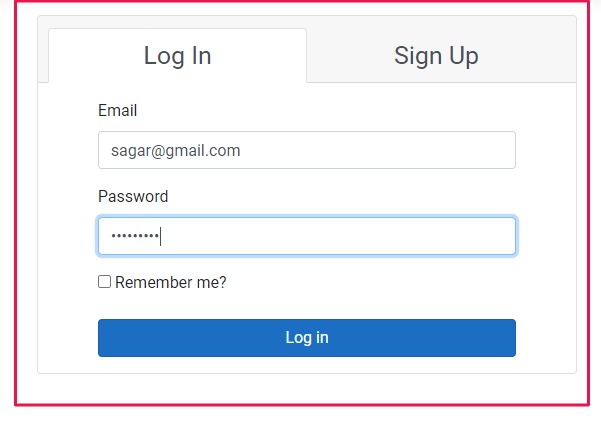
In order to display the active tab, we need to add an active class, as shown below.

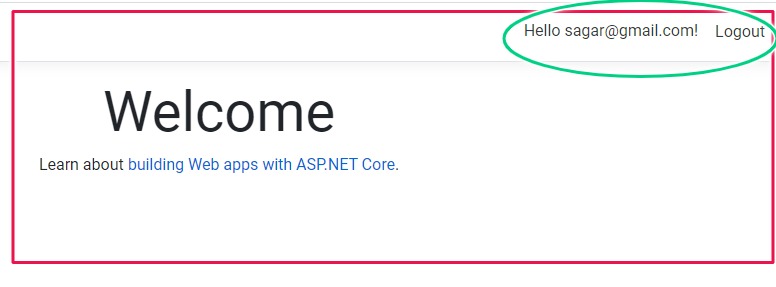
1. @section Scripts{
3. @RenderSection("Scripts", required: **false**)
4. <script>
5. $(**function** () {
7. **var** current = location.pathname;
8. $('.nav-tabs li a').each(**function** () {
9. **var** $**this** = $(**this**);
10. **if** (current.indexOf($**this**.attr('href')) !== -1) {
11. $**this**.addClass('active');
12. }
13. })
14. })
15. </script>
16. }

Let's see the output and register a user.



Now let's login with the registered user.





<https://www.c-sharpcorner.com/article/create-identity-in-simple-ways-using-asp-net-mvc-5/>