# Github Repository: https://github.com/Zayyad-Butt/Asp.NetCore-Identity

# Tutorial # 1

## **Goal:**

* In this tutorial we will cover how we can use Asp.Net Core Identity to authenticate users.
* We will use Database tables provided by Asp.net core identity to store user’s info.
* We will make a signup page and a login page and implement it using Asp.Net Core Identity.

## **Prerequisites:**

* Entity Framework
* .Net 6.0
* Tag Helpers
* MVC Framework
* Html, CSS, Bootstrap

## **What is Asp.Net Core identity?**

ASP.NET Core Identity:

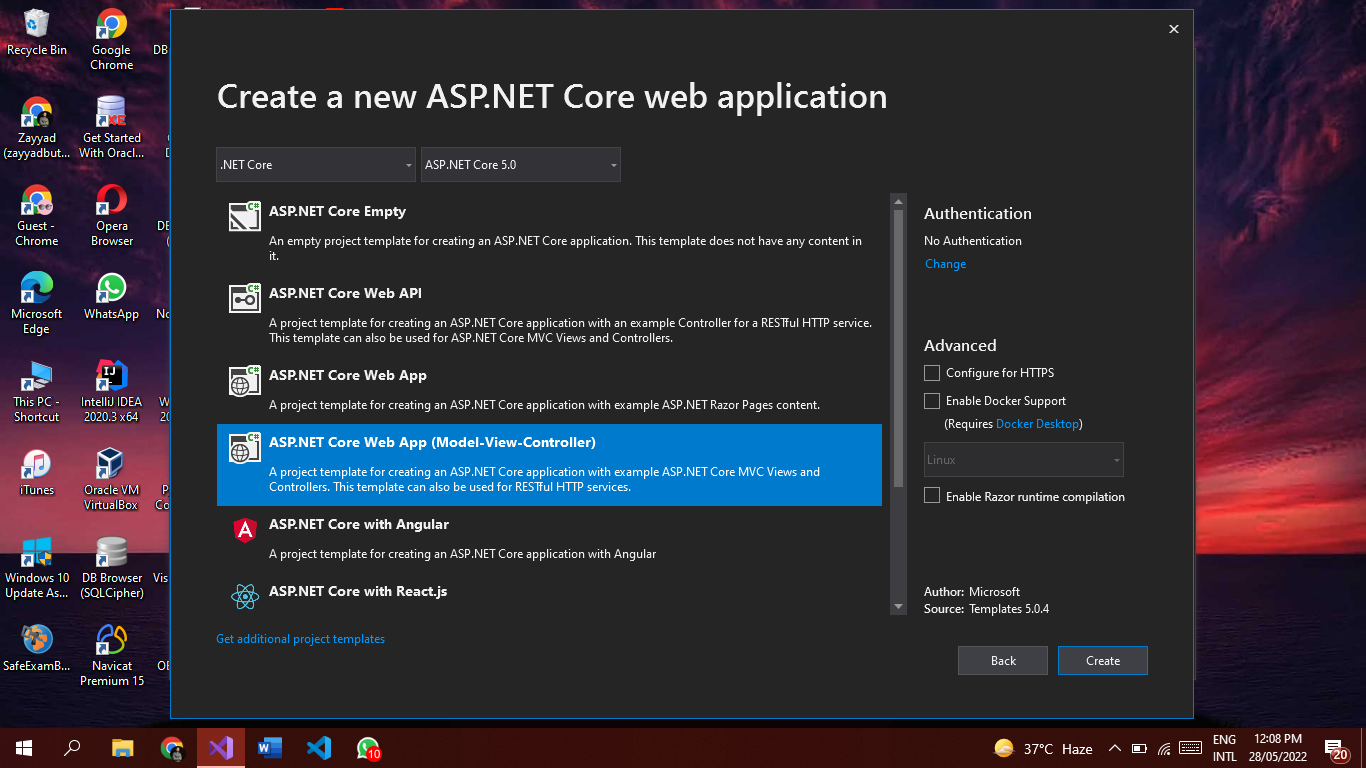
* supports user interface (UI) login functionality.
* Is designed to be used together with IdentityServer.
* Manages users, passwords, profile data, roles, claims, email confirmation, and more.

Users can create an account with the login information stored in Identity or they can use an external login provider. Supported external login providers include Facebook, Google etc.

## **Implementation:**

### *Making Project and connection with Asp.NetCoreIdentity Tables*

* Open Visual Studio and make Asp.Net core Web Application MVC Project



* Install these packages.

| Microsoft.EntityFrameworkCore.Tools Microsoft.EntityFrameworkCore.sqlserver Microsoft.aspnetcore.identity.EntityFramework |
| --- |

**=>** EntityFrameworkCore.Tools is for enabling commands lke Add-Migration, Update-Database etc.

**=>** EntityFrameworkCore.Sqlserver is for enabling connection with sql server database

**=>** ASP.NET Core Identity provider that uses Entity Framework Core.

* Go to Models Folder and make a class named **AppDbContext** and following code there.

| public class AppDbContext: IdentityDbContext {  public AppDbContext(DbContextOptions<AppDbContext>options):base(options)   { }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  base.OnModelCreating(modelBuilder);  } } |
| --- |

* Add these header files too.

| using Microsoft.AspNetCore.Identity.EntityFrameworkCore; using Microsoft.EntityFrameworkCore; |
| --- |

Here we have inherited the AppDbContext class from IdentityDbContext which is going to be used to create necessary tables for Identity Users.

Also we have override method OnModelCreate which uses a set of conventions to build a model based on the shape of your entity classes (in our case Identity user classes). You may use it to seed data into your tables.

* Now **Create Db Connection**, for this purpose you first need to make your connection string in **appsettings.json** file
* Add following connection string in appsetting.json file

| "ConnectionStrings": {  "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=UserDbTest;Trusted\_Connection=T  rue;MultipleActiveResultSets=true"  } |
| --- |

Here you can give your local db server name and database name as defined above.

* Go to **Program.cs file**  and update as follows:

| *//Getting Connection string* string connString=builder.Configuration.GetConnectionString("DefaultConnection"); *//Getting Assembly Name* var migrationAssembly=typeof(Program).Assembly.GetName().Name;  *// Add services to the container.* builder.Services.AddDbContext<AppDbContext>(options =>  options.UseSqlServer(connString, sql => sql.MigrationsAssembly(migrationAssembly));  builder.Services.AddIdentity<IdentityUser, IdentityRole>()  .AddEntityFrameworkStores<AppDbContext>(); |
| --- |

Now you are ready to make migrations to make tables provided by Asp.net core identity in Db.

* Open Package Manager Console and add the following command.

| Add-Migration InitialDbContext |
| --- |

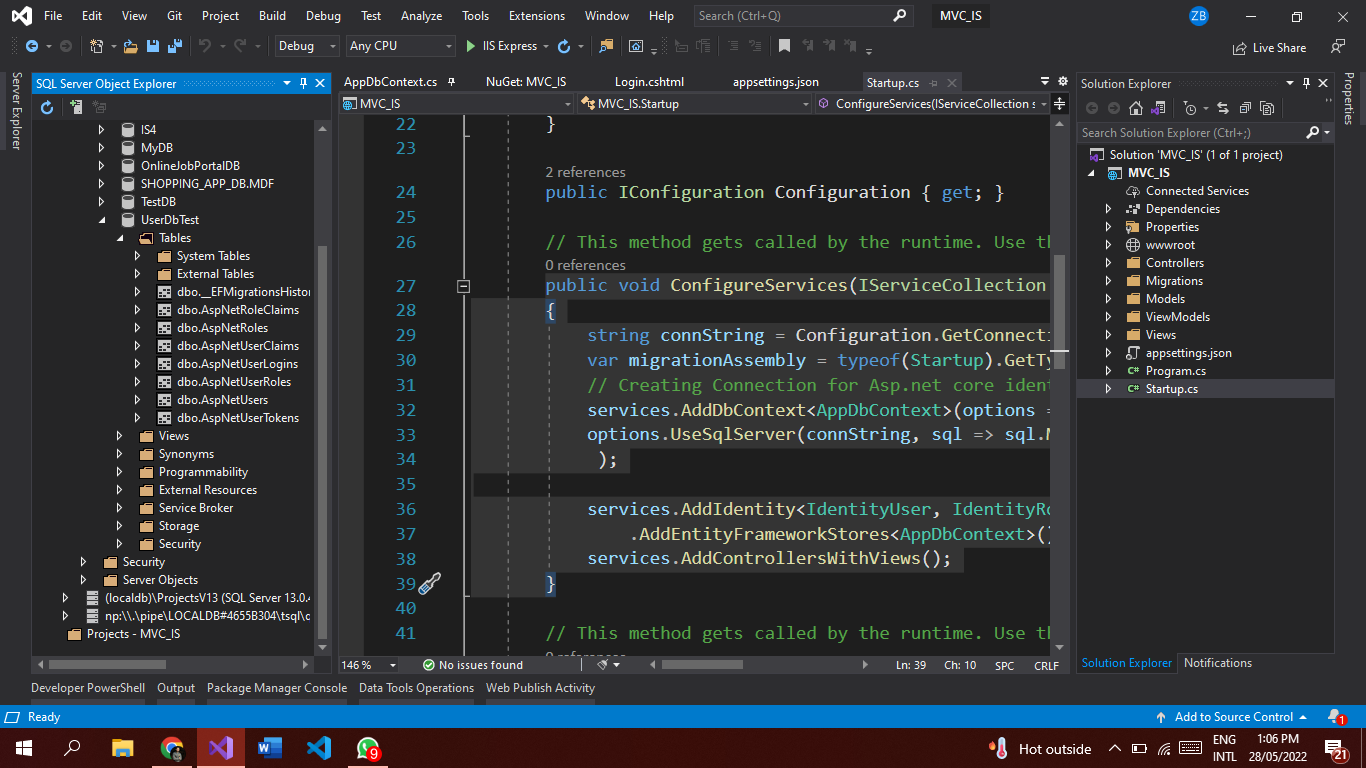
InitialDbContext is the name of our migration.

After this you can see Migrations Folder in your project and where you can see class where table names and fields are given.

For making tables and following command.

| Update-Database |
| --- |

* Now go to SqlServerObject and search for your db and explore it. You can see tables have added there.



### *Integrating Asp.NetCoreIdentity Functionality to Signup And Login Pages*

#### *SignUp Page Functionality*

* Go to Controllers Folder and make a new controller named AccountController and make a function as follows.

| [HttpGet] public IActionResult Register()  {  return View(); } |
| --- |

* Make a new Folder ViewModels and make following properties for User Signup.

| public class RegisterViewModel {  [Required]  [StringLength(20,ErrorMessage ="Length should not exceed 20  characters")]  public string Username { get; set; }  [Required]  [EmailAddress]  public string Email { get; set; }   [Required]  [DataType(DataType.Password)]  public string Password { get; set; }  [Required]  [DataType(DataType.Password)]  [Compare("Password", ErrorMessage = "Password must match with  confirmation password")]  public string ConfirmPassword { get; set; } } |
| --- |

* Now make a view against this controller, in view folder and add following code there.

| @model MVC\_IS.ViewModels.RegisterViewModel <h2>Register User</h2> <div asp-validation-summary="All" class="text-danger"></div> <form method="post" asp-action="Register">  <div class="mb-3 form-group">  <label asp-for="Username" class="form-label"></label>  <input asp-for="Username" placeholder="Username"  class="form-control" id="username">  <span asp-validation-for="Username" class="text-danger"></span>  </div>  <div class="mb-3 form-group">  <label asp-for="Email" class="form-label"></label>  <input asp-for="Email" placeholder="Email" class="form-control"  id="email">  <span asp-validation-for="Email" class="text-danger"></span>  </div>    <div class="mb-3 form-group">  <label asp-for="Password" class="form-label"></label>  <input asp-for="Password" type="password" placeholder="Password"  class="form-control" id="exampleInputPassword1">  <span asp-validation-for="Password" class="text-danger"></span>  </div>  <div class="mb-3 form-group">  <label asp-for="ConfirmPassword" class="form-label"></label>  <input asp-for="ConfirmPassword" type="password"  placeholder="Confirm Password" class="form-control"  id="exampleInputPassword2">  <span asp-validation-for="ConfirmPassword"  class="text-danger"></span>  </div>   <button type="submit" class="btn btn-primary">Register</button> </form> |
| --- |

* Make Objects of UserManager and SignInManager classes in Account Controller and make Constructor

| private readonly UserManager<IdentityUser> userManager;  private readonly SignInManager<IdentityUser> signInManager;  public AccountController(UserManager<IdentityUser> uManager,  SignInManager<IdentityUser> sManager)  {  userManager = uManager;  signInManager = sManager; } |
| --- |

UserManager is used to manage user functionalities like adding and deleting user etc.

SignInManager is used to manage functionalities related to signin and signout etc.

* Now add controller for HttpPost Request.

| [HttpPost]  public async Task<IActionResult> Register(RegisterViewModel model)  {  if (ModelState.IsValid)  {  var user = new IdentityUser  {  UserName = model.Username,  Email = model.Email,  EmailConfirmed = true,  LockoutEnabled = false,  };  var result = await userManager.CreateAsync(user,  model.Password);   if (result.Succeeded)  {  if (signInManager.IsSignedIn(User))  {  return RedirectToAction("index", "home");  }  return RedirectToAction("login", "account");  }  foreach (var error in result.Errors)  {  ModelState.AddModelError("", error.Description);  }  }  return View();  } |
| --- |

You can see we are accepting RegisterViewModel as a parameter in function. In function we are making an Instance of User. and assigning values to its properties which we got as a parameter in function.

CreateAsync method is used to create that user in database table. Which accepts two parameters, password and a user, then we are checking if user is added in table then check if someone is already Signed in then do not signin the registered user, in short,

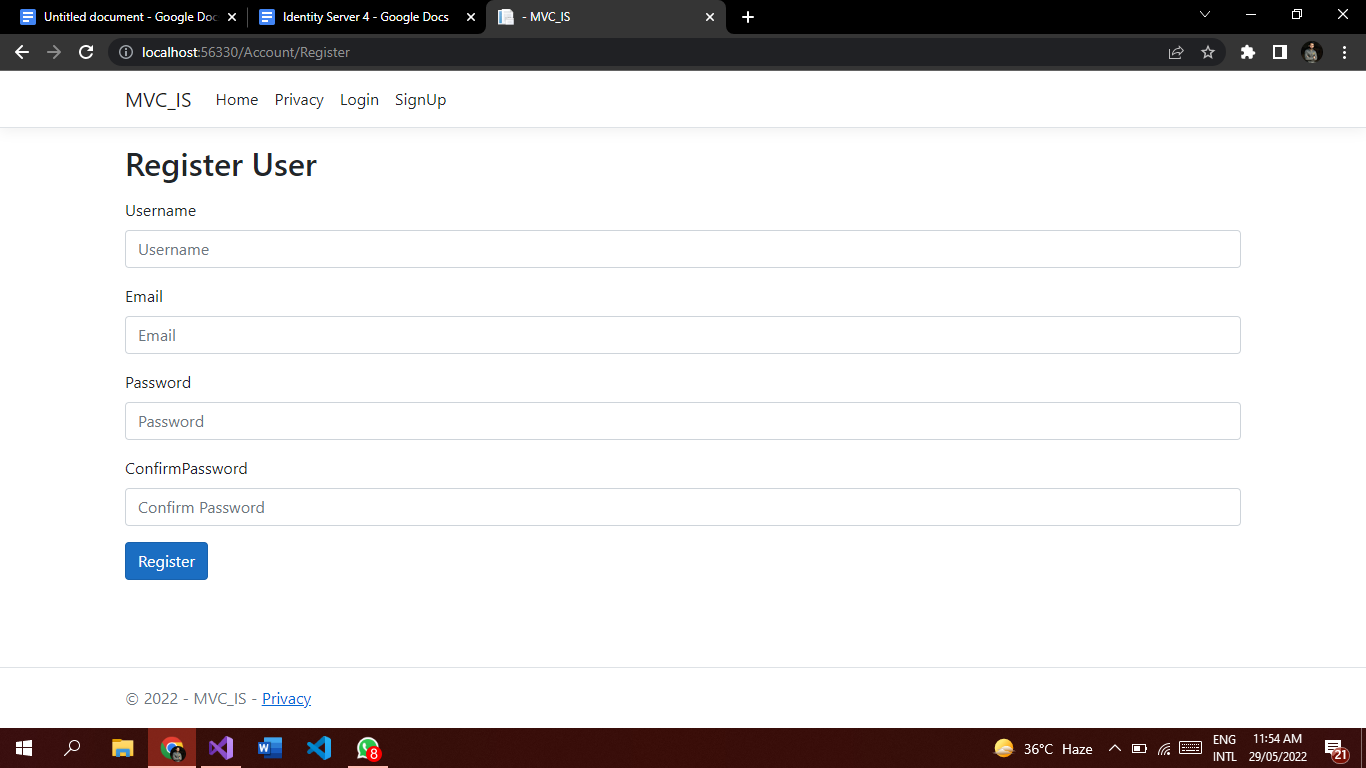
\_signInManager.SignInAsync(user, isPersistent: false);

Function is used to signing user. As our intention is only to add user in table not logging in, so we have the check of IsUserSignedIn.

You can also see i have set LockOutEnabled: false, because if i set it to true then it will block this user , and does not allow it to enter in page, if credentials are correct.

And in this tutorial our main focus is not confirming email, so that i hard codedly set it to true. But practically we make it sure that the email is provided is correct or not.

* Run project and add user



#### *Login Page Functionality*

* Make HttpGet Request for Login page

| [HttpGet]  public IActionResult Login()  {  return View();  } |
| --- |

* Create View Models for Login page as follows.

| public class LoginViewModel {  [Required]  [StringLength(20)]  public string Username { get; set; }   [Required]  [DataType(DataType.Password)]  public string Password { get; set; } } |
| --- |

* Add View Against Login

| @model MVC\_IS.ViewModels.LoginViewModel  <h2>User Sign In</h2> <div asp-validation-summary="All" class="text-danger"></div>  <form method="post" style="padding-left:200px ;padding-right:200px;" class="mt-5 container">  <div class="mb-3 form-group">  <label asp-for="Username" class="form-label"></label>  <input asp-for="Username" type="text" placeholder="Username" class="form-control" id="username">  <span asp-validation-for="Username" class="text-danger"></span>  </div>   <div class="mb-3 form-group">  <label asp-for="Password" class="form-label"></label>  <input asp-for="Password" type="password" placeholder="Password" class="form-control" id="exampleInputPassword1">  <span asp-validation-for="Password" class="text-danger"></span>  </div>  <button type="submit" class="btn btn-primary">Log In</button> </form> |
| --- |

* Add HttpPost Request for Login page

| [HttpPost] public async Task<IActionResult> Login(LoginViewModel model,  string returnUrl) {  if (ModelState.IsValid)  {  var result = await  signInManager.PasswordSignInAsync(model.Username,  model.Password, false, false);  if (result.Succeeded)  {  return RedirectToAction("index", "home");   }  ModelState.AddModelError(string.Empty, "Invalid Username or  Password");   }  return View(model); } |
| --- |

* Run Project and check login functionalities againsgt the user you have added in db.

#### *LogOut Functionality*

* Add Logout functionality against user.
* Go to Account controller and add this function there

| [HttpGet] public async Task<IActionResult> Logout() {  await signInManager.SignOutAsync();  return RedirectToAction("index", "home"); } |
| --- |