# Projects

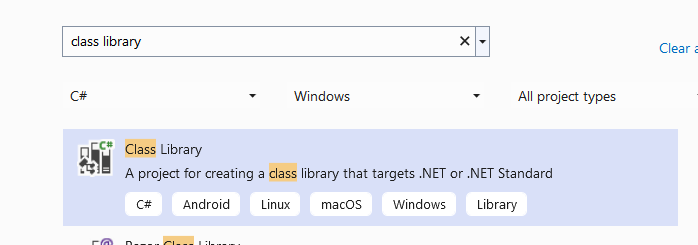
DB Project – MVC .NET Core

TestME – Console .NET Core

HousingModel – Class Library .NET Core

HousingBlazorApp – The front end using Blazor

# Models Project - HousingModel



A screenshot of a computer

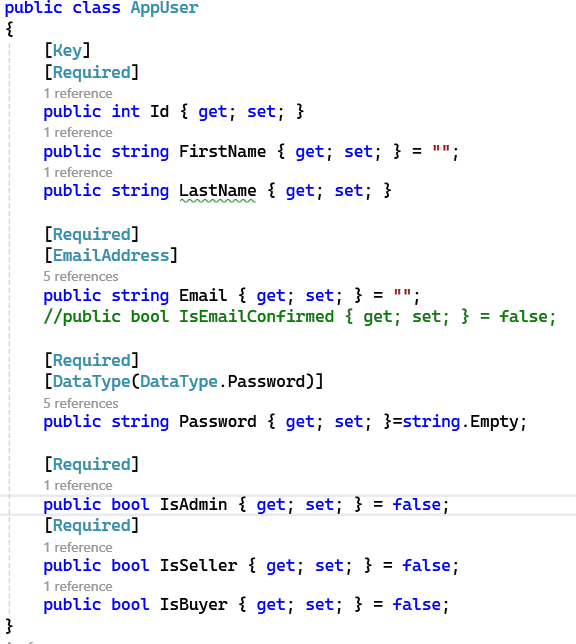
Description automatically generated

A screenshot of a computer

Description automatically generated

## Add Model C# classes:





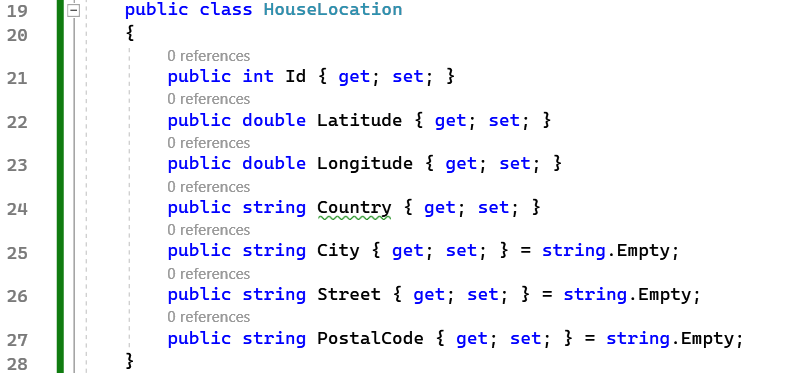
A computer code with blue text

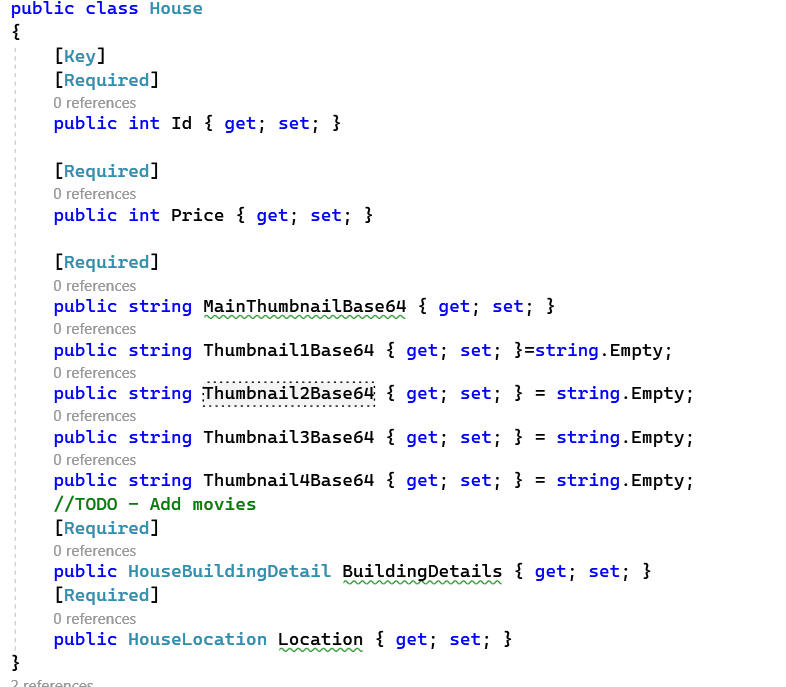
Description automatically generated

House:

A screenshot of a computer program

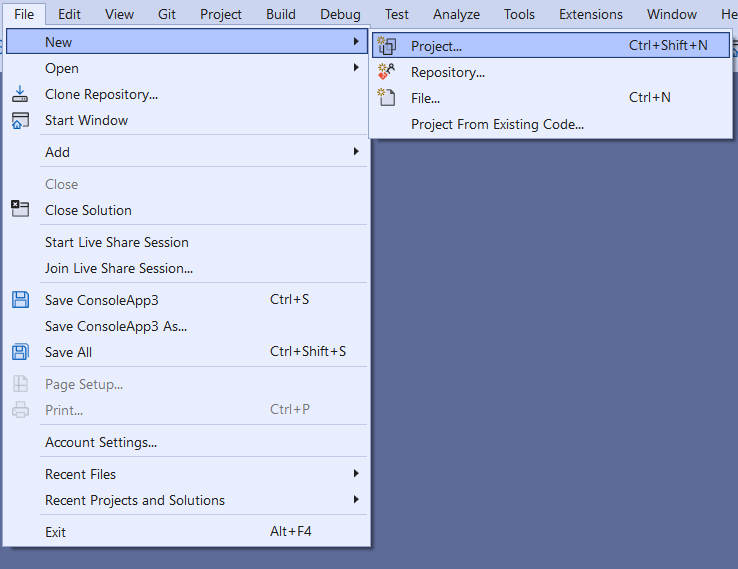
Description automatically generated

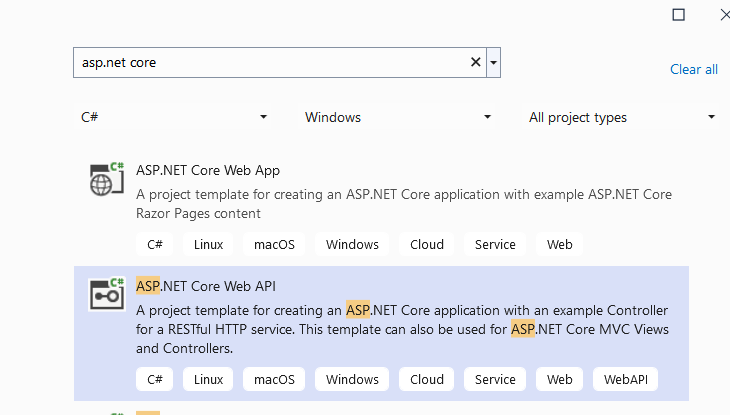




…

# Create New DB Project





A screenshot of a computer

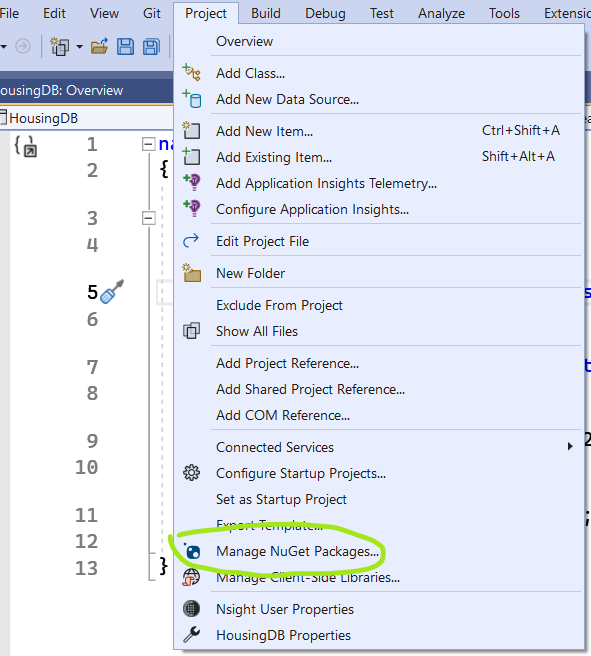
Description automatically generated

A screenshot of a computer

Description automatically generated

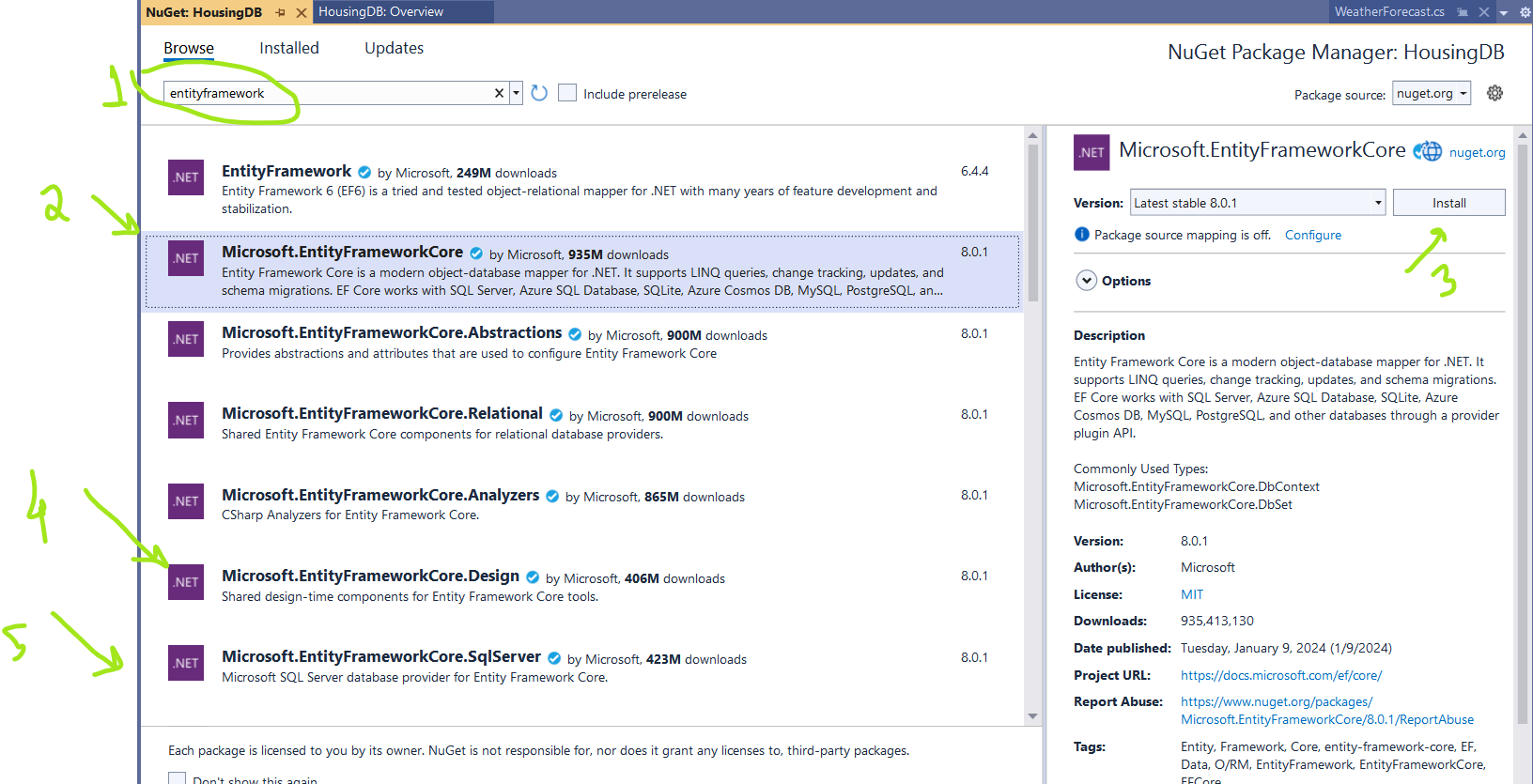
## Add Dependencies:

1. Drag HousinngModel to HousingDB project
2. Add the following

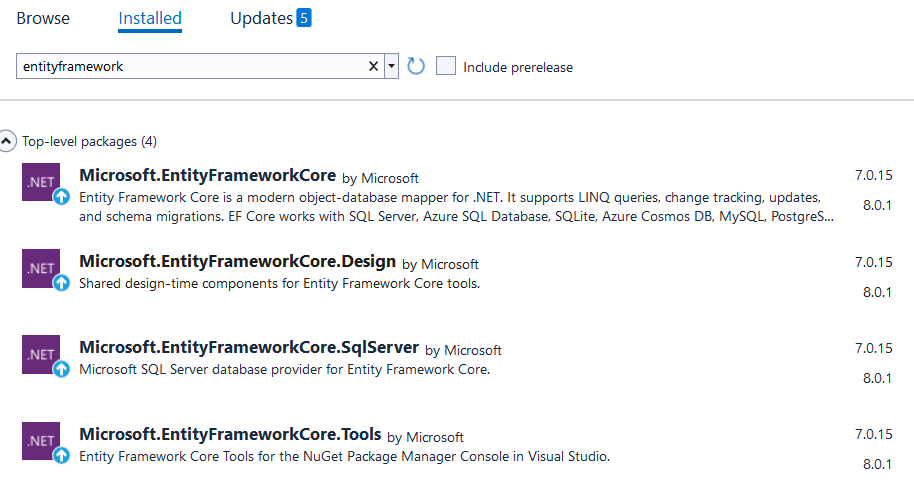


A screenshot of a computer

Description automatically generated



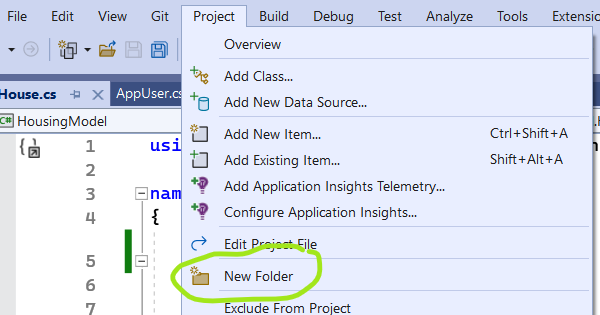
Also Tools



Note: Version of .NET Core should match the version of the dependency’s packages.

## Create DB

* In the HousingDB project add new folder called: App\_Data



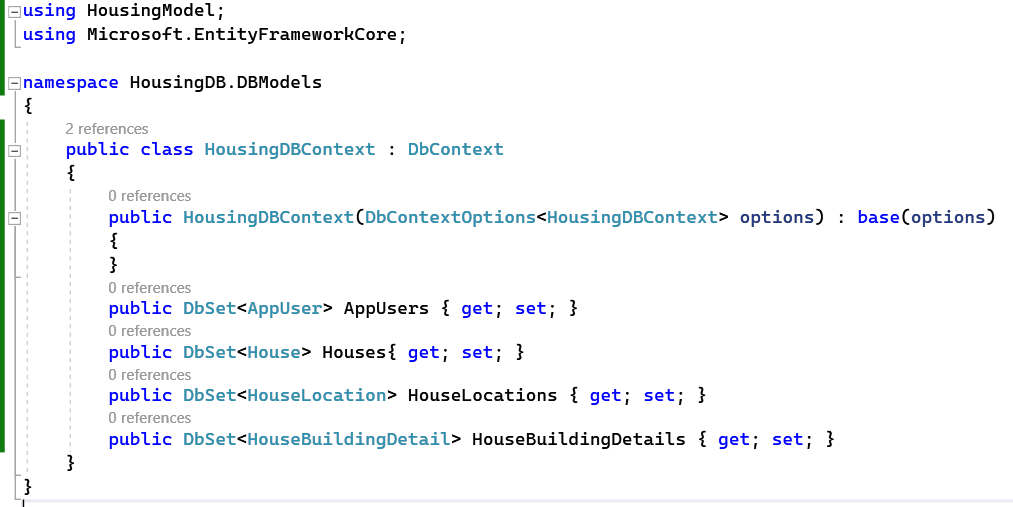
* Add the following to the end of appsettings.json

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;AttachDBFilename=[DataDirectory]\\App\_Data\\housing.mdf;Initial Catalog=housing;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

* Add new folder called: DBModels under HousingDB project
* Add class: HousingDBContext extends from DbContext under DBModels folder



* Modify program.cs to include the following **before: builder.Build()**

string path = Directory.GetCurrentDirectory();

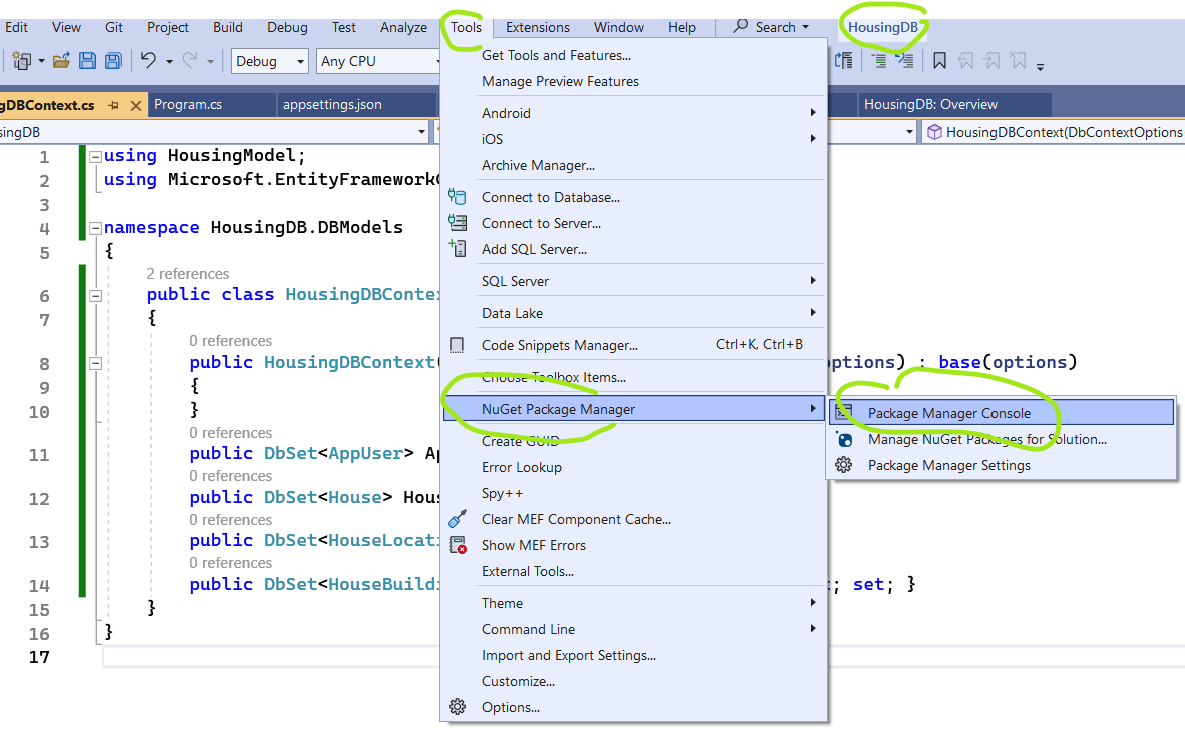
builder.Services.AddDbContext<HousingDBContext>(options =>

options.UseSqlServer(

builder.Configuration.GetConnectionString("DefaultConnection")

.Replace("[DataDirectory]", path)));

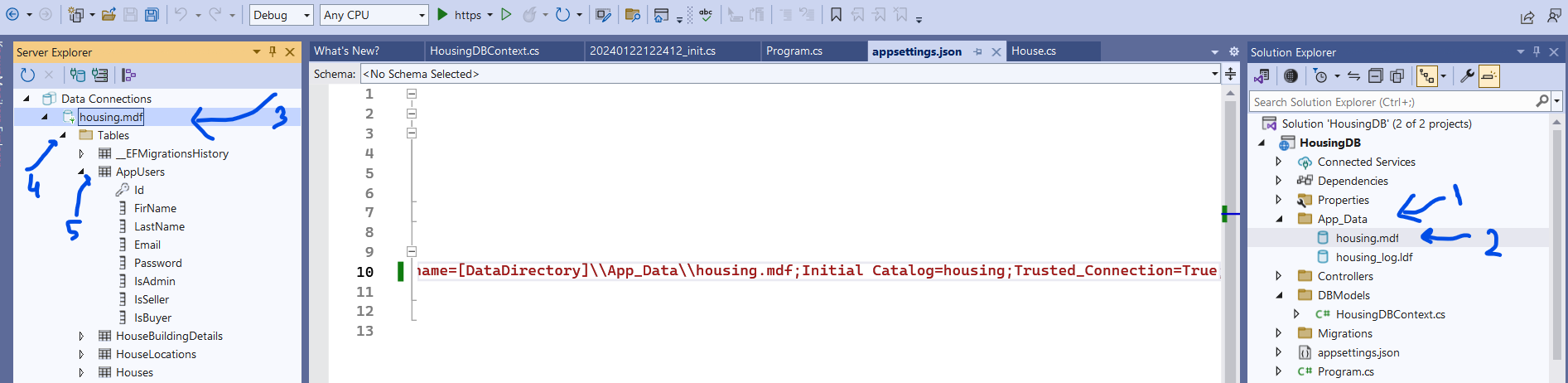
* Create DB with Package Manager Console (PMC)



* In the console type: add-migration "init"

Make sure it successful

* In the console type: Update-Database
* NOW: make sure you have housing.mdf file under App\_Data folder
  + You may double-click on it and see the DB with the tables defined before.



## Controllers

Add new class: AppUserController.cs under Controllers folder

namespace HousingDB.Controllers

{

using Microsoft.AspNetCore.Mvc;

namespace HousingDB.Controllers

{

[ApiController]

[Route("[controller]")]

public class AppUserController : **ControllerBase**

{

}

}

}

Add methods – Login, Register to the class as follows:

public class AppUserController : ControllerBase

{

HousingDBContext context;

public AppUserController(HousingDBContext context)

{

this.context = context;

}

[HttpPost(Name ="Login")]

public ActionResult<AppUser?> Login([FromBody] AppLogin data)

{

AppUser? user = context.AppUsers.Where(u => u.Email.ToLower()==data.Email.ToLower() && u.Password==data.Password).FirstOrDefault();

if(user == null)

{

return NotFound();

}

return user;

}

[HttpPost(Name = "Login2")]

public ActionResult<int> Login2(string email, string password)

{

AppUser? user = context.AppUsers.Where(u => u.Email.ToLower() == email.ToLower() && u.Password == password).FirstOrDefault();

if(user == null)

{

return NotFound();

}

return user.Id;

}

[HttpPost(Name = "Register")]

public ActionResult Register([FromBody]AppUser user)

{

//First check if user exists - if so, issue an error

AppUser? appuser = context.AppUsers.Where(u => u.Email.ToLower() == user.Email.ToLower() &&

u.Password == user.Password).FirstOrDefault();

//Not exists - OK, add him

if (appuser == null)

{

context.AppUsers.Add(user);

context.SaveChanges();

return Ok();

}

else

{

return Problem("User already exists");

}

}

}

### Multiple Get/Set

.NET Core default the GET/Set to one per controller, if you want to have multiple Get/Post on same controller, modify the header of the class controller to:

[Route("api/[controller]/[action]")]

In tnis case, all your methods will be: api/<controller>/<method>

\*\*controller: is the class name of the controller (w/o the Controller suffix)

\*\*method: is the method name or the annotation: [HttpGet(Name=”**????**”)]

# TestMe

This project is meant to run sanity/unity tests quickly w/o need to the UI.

**OR – You can use Swagger**

### Add Dependencies

Add the HousingModel as dependencies to this one. (D&D HousingModel to TestMe)

### Add test

In the Main:

using HousingModel;

using System.Net.Http.Json;

public partial class Program

{

public static async Task Main(string[] args)

{

Console.ReadKey();

HttpClient client = new HttpClient();

var resp = await client.GetAsync("https://localhost:7262/api/AppUser/Test");

string sresp = await resp.Content.ReadAsStringAsync();

Console.WriteLine(sresp);

//Login should fail

HttpClient client2 = new HttpClient();

var data = new AppLogin() { Email = "ahmad", Password = "password" };

var content = JsonContent.Create(data);

var resp2 = await client2.PostAsync("https://localhost:7262/api/AppUser/Login", content);

string sresp2 = await resp2.Content.ReadAsStringAsync();

client.Dispose();

Console.WriteLine(sresp2);

//Register dummy user

HttpClient client3 = new HttpClient();

var data3 = new AppUser() { Email="dummy@gmail.com", Password= "password", FirstName="Dummy",LastName="Dummy",IsBuyer=false,IsSeller=false, IsAdmin=false };

var content3 = JsonContent.Create(data3);

var resp3 = await client3.PostAsync("https://localhost:7262/api/AppUser/Register", content3);

string s3 = await resp3.Content.ReadAsStringAsync();

Console.WriteLine(s3);

//Login should go OK

HttpClient client4 = new HttpClient();

var data4 = new AppLogin() { Email = "dummy@gmail.com", Password = "password" };

var content4 = JsonContent.Create(data4);

var resp4 = await client4.PostAsync("https://localhost:7262/api/AppUser/Login", content4);

string sresp4 = await resp4.Content.ReadAsStringAsync();

client.Dispose();

Console.WriteLine(sresp4);

Console.ReadKey();

}

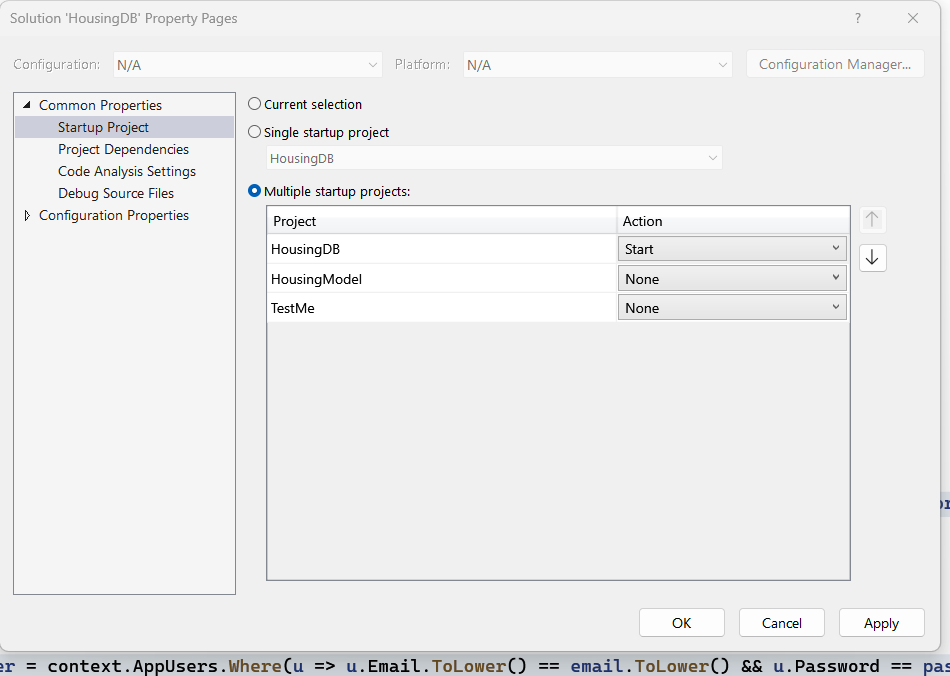
}

# Run and Test

Configure the run:

Right click on the **Solution**

Define the **Startup Project** as multiple



\*\*Later we will add the Blazor server to the list

\*\*If you want to test the Console App then change TestMe to Start

Click OK and run

# New APIs

Add the following to the AppUserController

[HttpPost(Name = "ChangePassword")]

public ActionResult ChangePassword([FromForm] string email, [FromForm] string oldPassword, [FromForm] string newPassword)

{

//check if user exists

AppUser? appuser = context.AppUsers.Where(u => u.Email.ToLower() == email.ToLower() &&

u.Password == oldPassword).FirstOrDefault();

if (appuser != null)

{

appuser.Password = newPassword;

context.AppUsers.Update(appuser);

context.SaveChanges();

return Ok();

}

else

{

return NotFound();

}

}

Add the following in the Program.cs in TestME project:

HttpClient client5 = new HttpClient();

MultipartFormDataContent c = new MultipartFormDataContent

{

{ new StringContent("email"), "email" },

{ new StringContent("password"), "oldPassword" },

{ new StringContent("password1"), "newPassword" }

};

var resp5 = await client5.PostAsync("https://localhost:7262/api/AppUser/ChangePassword", c);

string sresp5 = await resp5.Content.ReadAsStringAsync();

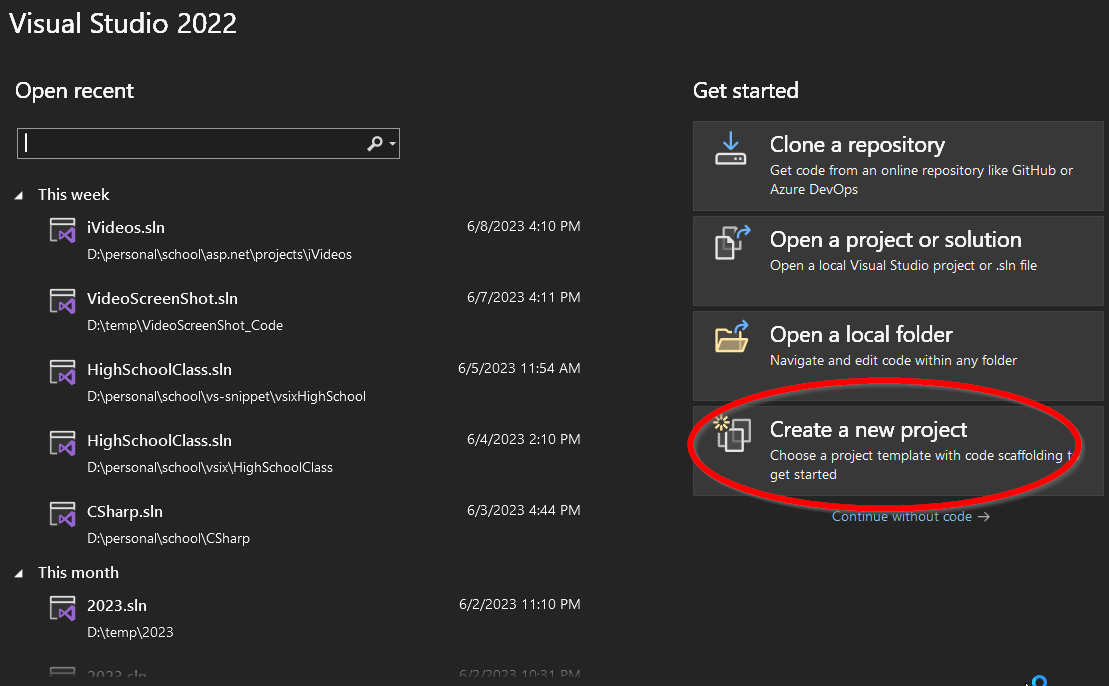
client.Dispose();

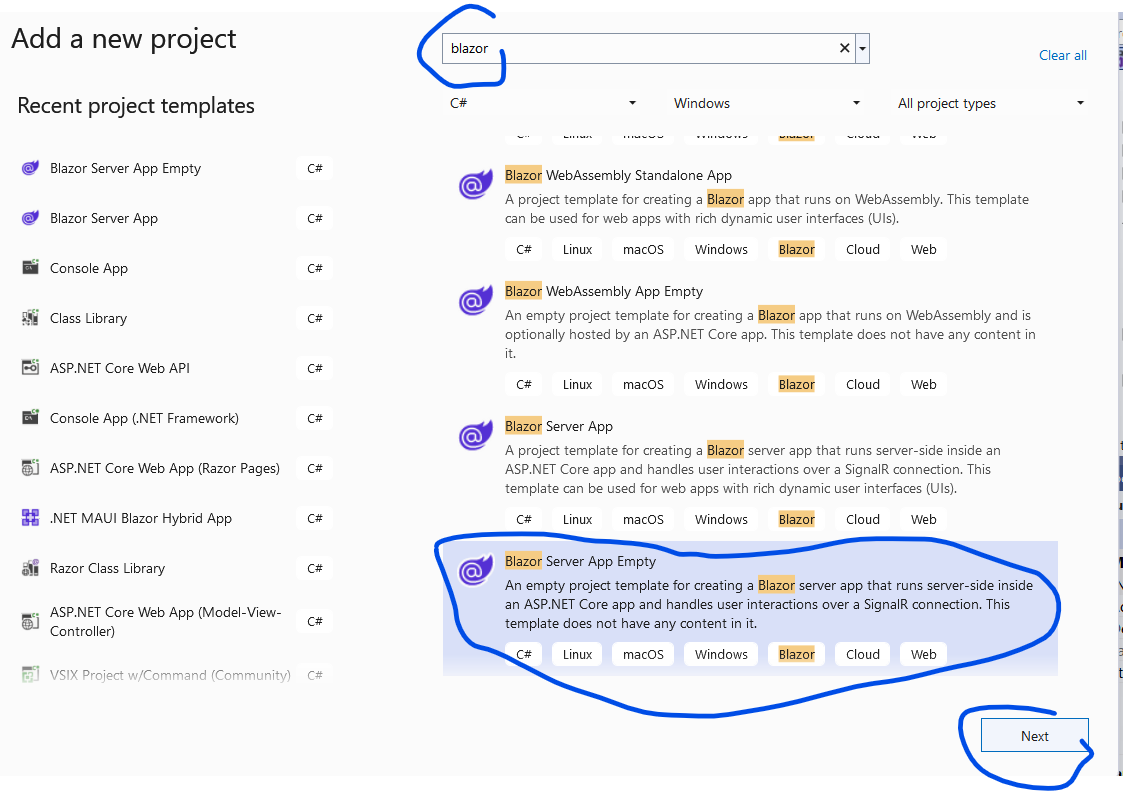
Console.WriteLine(sresp5);

\*\*Check now it is working.

# Blazor App

New project: Blazor Server





\*\*Or you can choose: Blazor Server App  
Then it will be created with some layout and Fetch data page sample

A screenshot of a computer

Description automatically generated

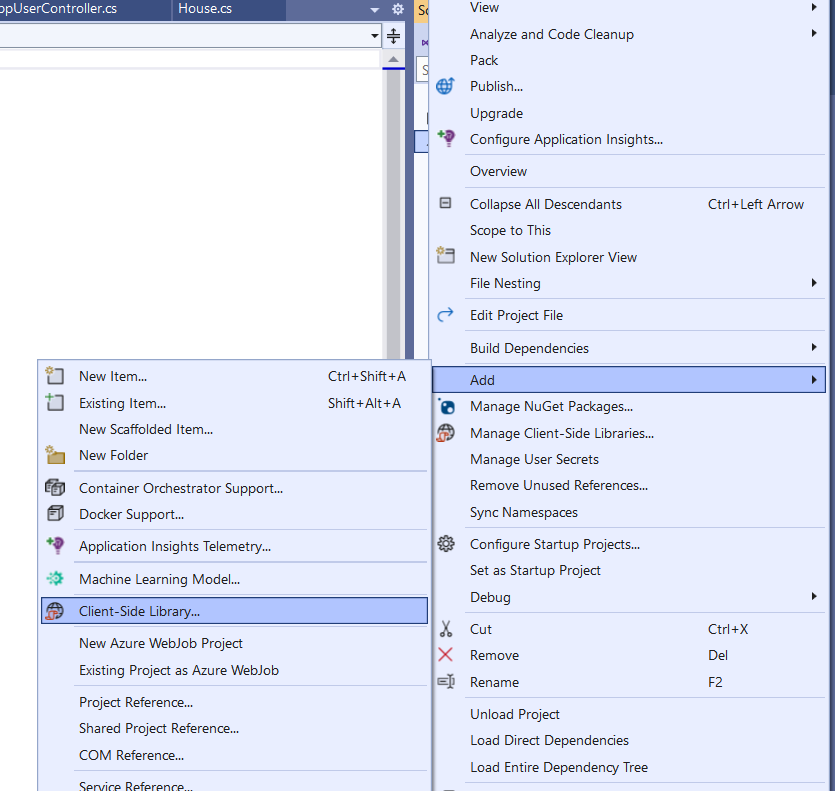
\*\*Call it: HousingBlazorApp

Nex… & Create…

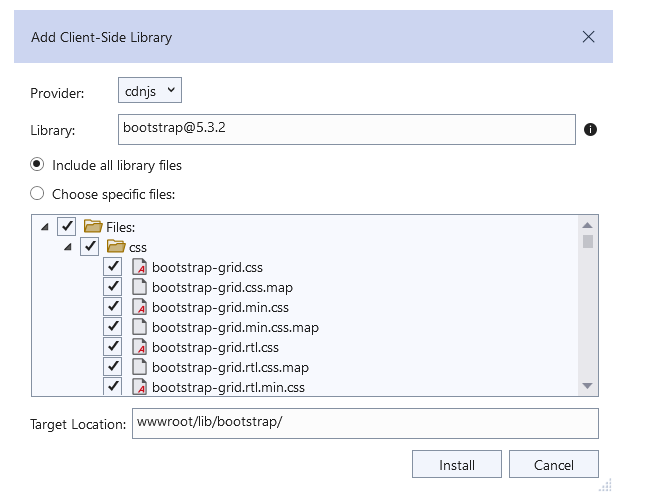
\*\* Once created, Drag HousingModel project into HousingApp project (Dependencies)

Add bootstrap

Right click on “HousingBlazorApp”



In the “Add Client Side Library” type: bootstrap



Click: Install

Open Pages\\_Host.cshtml file and add:   
<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />  
to the head section (just before any other css)

Add favicon to the website

Pick your favorite icon for the website, add the icon (image) to the wwwroot\images\favicon.png

Add the following to the Pages\\_Host.cshtml under the head section:

<link rel="icon" type="image/png" href="/images/favicon.png"/>

\*\*\*Make sure you have the following under head section:

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<**base** href="~/" />

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

<link href="css/site.css" rel="stylesheet" />

<link href="HousingBlazorApp.styles.css" rel="stylesheet" />

<link rel="icon" type="image/png" href="/images/favicon.png" />

<**component** **type**="typeof(HeadOutlet)" **render-mode**="ServerPrerendered" />

## Home page

Add HomePageComponent.razor under Pages

Add whatever content, for example:

@page "/home"

<h3>Home Page</h3>

<h2>Welcome to my housing web site</h2>

@code {

}

## User Pages

Add “User” folder under Pages folder

Right click on User folder -> Add -> Blazor Component -> name is: LoginComponent.razor

Right click on User folder -> Add -> Item -> name is: LoginComponent.razor.css

Add your html code under razor component and style under the razor.css file

Here an example of Login page: <https://www.w3schools.com/howto/howto_css_login_form.asp>

### Example

Add avatar.jpg image under wwwroot\images (to be used in the login page)

### LoginComponent.razor

@page "/login"

@using HousingModel

@inject NavigationManager navigationManager

<**PageTitle**>Login</**PageTitle**>

<h2>Login to Housing</h2>

<div class="main">

<form>

<div class="imgcontainer">

<img src="/images/avatar.jpg" alt="Avatar" class="avatar">

</div>

<div class="container">

<label for="uname"><b>Username</b></label>

<input type="text" placeholder="Enter Username" name="uname" @bind="uname" required>

<label for="psw"><b>Password</b></label>

<input type="password" placeholder="Enter Password" name="psw" @bind="psw" required>

<button type="button" @onclick="Login">Login</button>

<label>

<input type="checkbox" name="remember" @bind="rememberMe"> Remember me

</label>

</div>

<div class="container" style="background-color:#f1f1f1">

<button type="reset" class="cancelbtn">Cancel</button>

<span class="psw">Forgot <a href="#">password?</a></span>

</div>

</form>

@if (error.Length > 0)

{

<div class="text-danger">@error</div>

}

</div>

@code{

string uname;

string psw;

bool rememberMe=false;

string error = "2";

async Task Login()

{

error = "";

HttpClient client = new HttpClient();

var data = new AppLogin() { Email = uname, Password = psw };

var content = JsonContent.Create(data);

//TODO the URL should be configured.

var resp = await client.PostAsync("https://localhost:7262/api/AppUser/Login", content);

if (resp.IsSuccessStatusCode)

{

//go to home page or any other page

navigationManager.NavigateTo("/home");

}

else

{

//display error message

//error = await resp.Content.ReadAsStringAsync();

error = "One of the fields are wrong! Try again";

}

client.Dispose();

StateHasChanged();

}

}

LoginComponent.razor.css:

body {

font-family: Arial, Helvetica, sans-serif;

}

/\* Full-width input fields \*/

input[type=text], input[type=password] {

width: 100%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

box-sizing: border-box;

}

/\* Set a style for all buttons \*/

button {

background-color: #04AA6D;

color: white;

padding: 14px 20px;

margin: 8px 0;

border: none;

cursor: pointer;

width: 100%;

}

button:hover {

opacity: 0.8;

}

/\* Extra styles for the cancel button \*/

.cancelbtn {

width: auto;

padding: 10px 18px;

background-color: #f44336;

}

/\* Center the image and position the close button \*/

.imgcontainer {

text-align: center;

margin: 24px 0 12px 0;

position: relative;

}

img.avatar {

width: 20%;

border-radius: 50%;

}

.container {

padding: 16px;

}

span.psw {

float: right;

padding-top: 16px;

}

/\* The Modal (background) \*/

.modal {

display: none; /\* Hidden by default \*/

position: fixed; /\* Stay in place \*/

z-index: 1; /\* Sit on top \*/

left: 0;

top: 0;

width: 100%; /\* Full width \*/

height: 100%; /\* Full height \*/

overflow: auto; /\* Enable scroll if needed \*/

background-color: rgb(0,0,0); /\* Fallback color \*/

background-color: rgba(0,0,0,0.4); /\* Black w/ opacity \*/

padding-top: 60px;

}

/\* Modal Content/Box \*/

.modal-content {

background-color: #fefefe;

margin: 5% auto 15% auto; /\* 5% from the top, 15% from the bottom and centered \*/

border: 1px solid #888;

width: 80%; /\* Could be more or less, depending on screen size \*/

}

/\* The Close Button (x) \*/

.close {

position: absolute;

right: 25px;

top: 0;

color: #000;

font-size: 35px;

font-weight: bold;

}

.close:hover,

.close:focus {

color: red;

cursor: pointer;

}

/\* Add Zoom Animation \*/

.animate {

-webkit-animation: animatezoom 0.6s;

animation: animatezoom 0.6s

}

@-webkit-keyframes animatezoom {

from {

-webkit-transform: scale(0)

}

to {

-webkit-transform: scale(1)

}

}

@keyframes animatezoom {

from {

transform: scale(0)

}

to {

transform: scale(1)

}

}

/\* Change styles for span and cancel button on extra small screens \*/

@media screen and (max-width: 300px) {

span.psw {

display: block;

float: none;

}

.cancelbtn {

width: 100%;

}

}

.main {

width: 50%;

margin: auto;

}

## Run & Test the application

Go to the URL: <https://localhost:7236/login>

type wrong credentials and see the error message

type correct credentials and see that we switch to “home” page

## Save session / User information

@inject ProtectedSessionStorage ProtectedSessionStore

\*\*add @using Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage

First, in the LoginComponent.razor, add the following at the top:

@using Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage

@inject ProtectedSessionStorage protectedSessionStore

Then in the @code section, after login was successfully, update the session by calling:

await protectedSessionStore.SetAsync("appuser.email", uname);

This is an example, you can store objects, whatever.

Now, in HomePageComponent.razor, replace the code to this one:

@page "/home"

@using HousingModel

@using Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage

@inject ProtectedSessionStorage protectedSessionStore;

<h1>Hello, world!</h1>

@if (username != null)

{

<h2>@username</h2>

}

else

{

<h2>You were not signed in. <a href="/login">Login</a></h2>

}

@code {

string? username = null;

protected override async Task OnAfterRenderAsync(bool firstRender)

{

if (firstRender)

{  
 //get data from storage in case user is logged in

var res = await protectedSessionStore.GetAsync<string>("appuser.email");

if (res.Success)

{

username = res.Value;

StateHasChanged(); //Call refresh

}

}

}

}

# General Services

This project is a collection of services that can be used in several projects, such as Blazor App and MAUI and maybe others,

## Services as ClassLibrary

Create the project as class library

Add HousingModels as dependencies to this one (I.e. HousingServices depends on HousingModels)

# **Send an Email from C#**

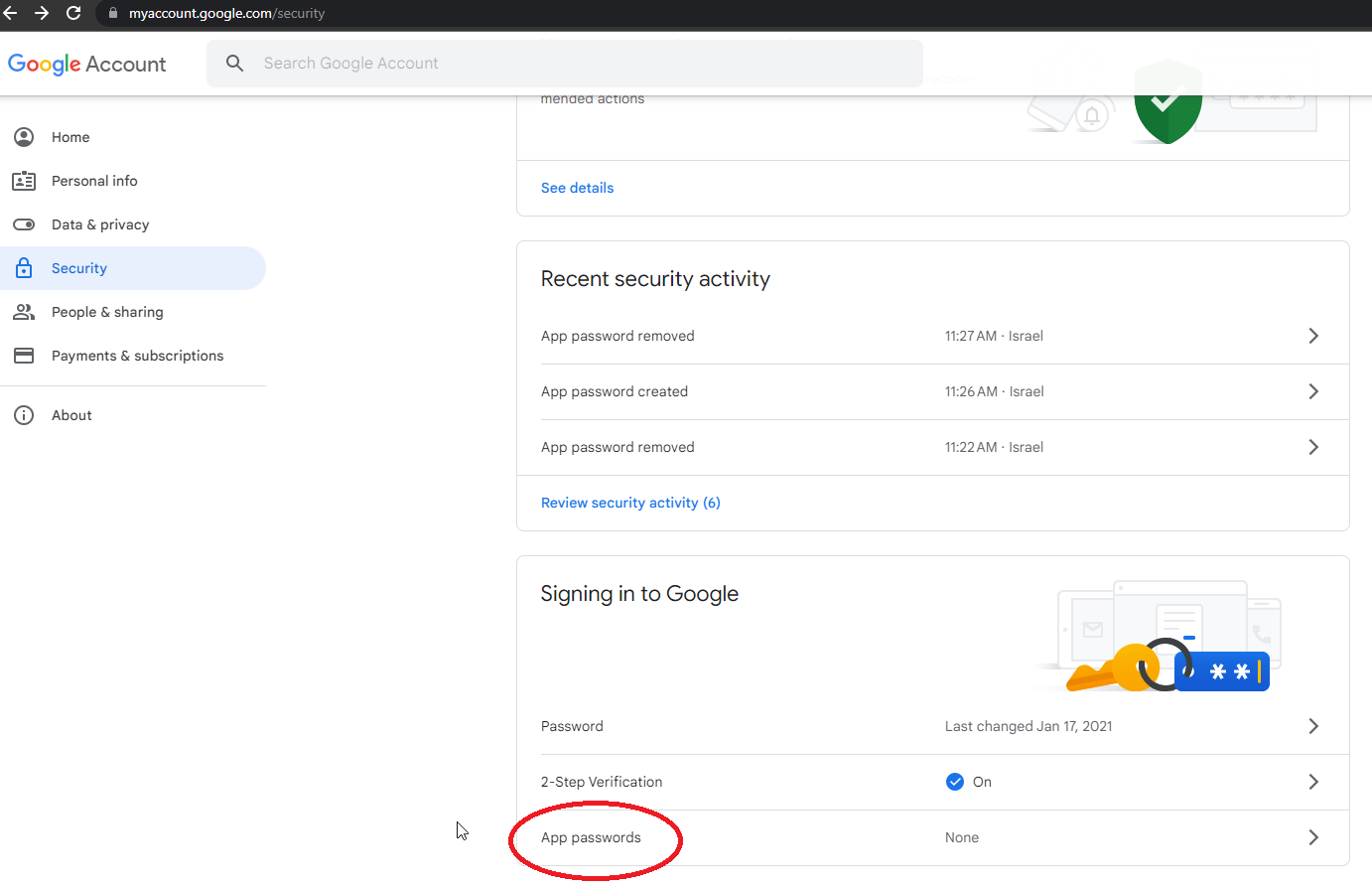
You may watch a short video explaining the details: <https://youtu.be/Ms_FDe2VzFU>

We are going to utilize the gmail smtp server to send emails. However, first we need to do some configuration in our gmail account.

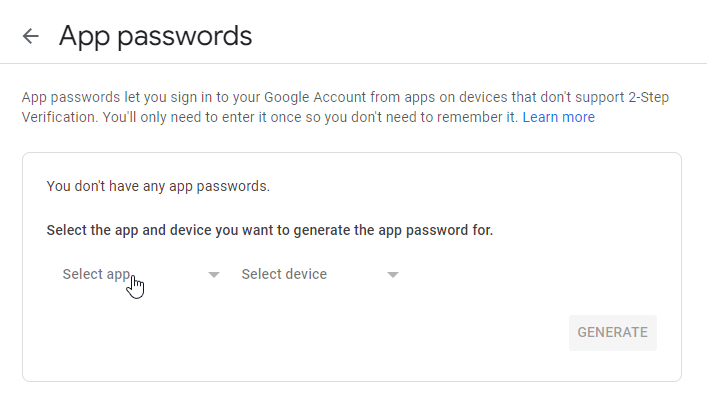
## **Configure your gmail**

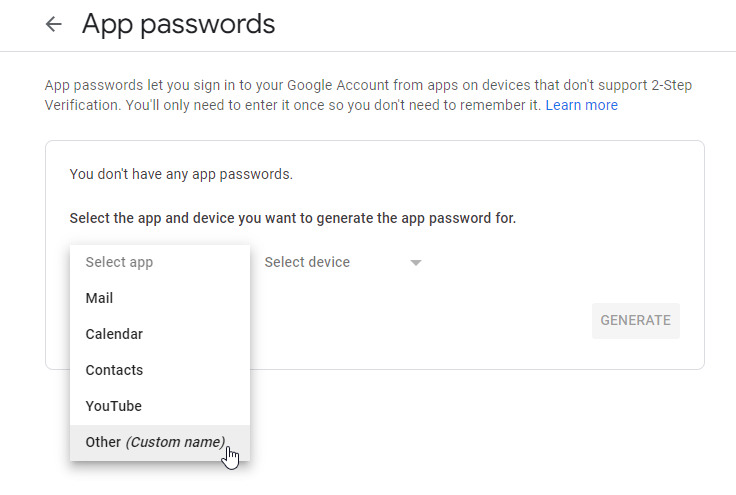
Visit <https://myaccount.google.com/security>

Under Signing in to Google, click: App Passwords

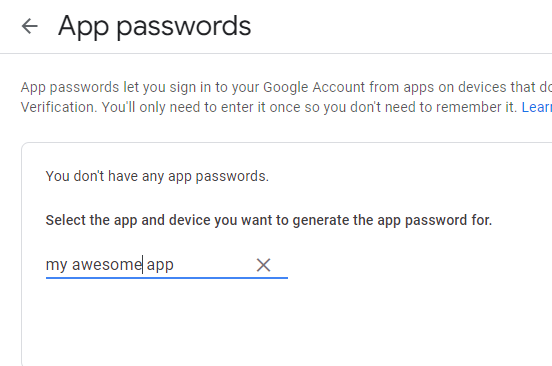


Select app → Select “Other (Custom name)”

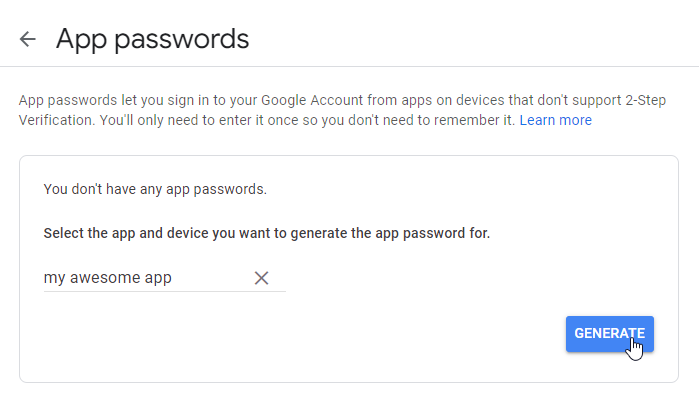




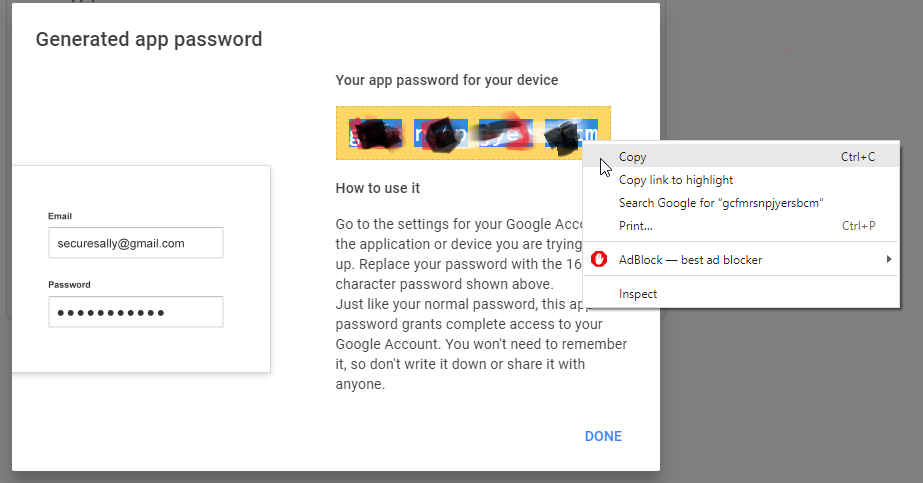
Type your app name (Any free name)



Click Generate



Copy the password from the popup (Paste it somewhere - we need it later)



Click on “Done” button

## **Configure Visual Studio Project**

### **Option 1: hardcode mail credentials.**

### **Send Email Code**

Copy and Paste the following code to some class (could be new class - e.g., EmailUtils):

using System.Net;

using System.Net.Mail;

public static async Task SendEmail(string to, string subject, string body, bool isBodyHTML = true)

{

    string from = "YOUR GMAIL";

    string display = "YOUR NAME";

    string emailPassword = "APP PASSWORD - from previous step";

    var fromEmail = new MailAddress(from, display);

    var toEmail = new MailAddress(to);

    var smtp = new SmtpClient

    {

        Host = "smtp.gmail.com",

        Port = 587,

        EnableSsl = true,

        Timeout = 60000, //60 seconds

        DeliveryMethod = SmtpDeliveryMethod.Network,

        UseDefaultCredentials = false,

        Credentials = new NetworkCredential(fromEmail.Address, emailPassword)

    };

    using var message = new MailMessage(fromEmail, toEmail)

    {

        Subject = subject,

        Body = body,

        IsBodyHtml = isBodyHTML,

    };

    try

    {

        await smtp.SendMailAsync(message);

    }

    catch

    {

    }

}

### **Option 2: configure mail credentials.**

We want to read the email credentials from some configuration file. In our case we are going to use **appsettings.json,** open the file and add the following:

"Email": {

    "from": {

      "mail": "**YOUR GMAIL**@gmail.com",

      "display": "**YOUR NAME**"

    },

    "password": "**App password from previous section**"

  }

Note: modify the piece of code above to meet your details. Modify the following:

YOUR GMAIL

YOUR NAME

App password from previous section

Save file

### **Send Email Code**

Same as previous one (hardcode mail) but this time we will take the credentials from the appsettings.json using IConfiguration:

using System.Net;

using System.Net.Mail;

namespace HousingDB.Services

{

//See https://docs.google.com/document/d/1-naW1AW-xv\_bhJFC1KHo\_QbQrtiu\_YCzgkOY2fn\_GEM/edit#heading=h.51lh3iio1qpt

public class EmailService

{

private IConfiguration configuration;

public EmailService(IConfiguration configuration)

{

configuration = configuration;

}

public async Task SendEmail(string to, string subject, string body, bool isBodyHTML = true)

{

IConfigurationSection config = configuration.GetSection("Email");

if(config!=null)

{

//read from the config from.mail, from.display, password

string from = config["from:mail"];

string display = config["from:display"];

string emailPassword = config["password"];

if (string.IsNullOrEmpty(emailPassword) || string.IsNullOrEmpty(from))

throw new Exception("Missing Email section in the appsettings");

var fromEmail = new MailAddress(from, display);

var toEmail = new MailAddress(to);

var smtp = new SmtpClient

{

Host = "smtp.gmail.com",

Port = 587,

EnableSsl = true,

Timeout = 5000,

DeliveryMethod = SmtpDeliveryMethod.Network,

UseDefaultCredentials = false,

Credentials = new NetworkCredential(fromEmail.Address, emailPassword)

};

using var message = new MailMessage(fromEmail, toEmail)

{

Subject = subject,

Body = body,

IsBodyHtml = isBodyHTML,

};

try

{

await smtp.SendMailAsync(message);

}

catch

{

}

}

}

}

}

Now, we want to register the service as singleton so we can use it in our code:

In **Program**.cs, before builder.Build add the following line:

builder.Services.AddSingleton<EmailService>();

### **Use SendEmail**

Locate the place you want to send email and type the following - this is an example, you may modify the code based on your use case. Assume we are willing to send the email once the user is Registered:

Add parma in the CTOR: EmailService emailService;

public async Task<IActionResult> Register(AppUser model)

{

//save the user in the db

…

//send him an email

string subject = "Welcome to zzz Store.";

string body = "<br/><br/><h3>Welcome, user.FirstName + ", " + user.LastName!</h3><br/>" +

    "Thanks for your registration to our web site." +

    "<br/><br/>If you didn't register to our site, please send us an email"

await emailService.SendEmail(user.Email, subject, body, true);

//In case of passing configuration

//await EmailUtils.SendEmail(this.configuration, user.Email, subject, body, true);

}

# New House

### New House Model

In HouseModels project, make sure to have House Model classes:

public class House

{

[Key]

[Required]

public int Id { get; set; }

[Required]

public int Price { get; set; }

[Required]

public string MainThumbnailBase64 { get; set; }

public string Thumbnail1Base64 { get; set; }=string.Empty;

public string Thumbnail2Base64 { get; set; } = string.Empty;

public string Thumbnail3Base64 { get; set; } = string.Empty;

public string Thumbnail4Base64 { get; set; } = string.Empty;

//TODO - Add movies

[Required]

public HouseBuildingDetail BuildingDetails { get; set; }

[Required]

public HouseLocation Location { get; set; }

}

public class HouseLocation

{

public int Id { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

[Required]

public string Country { get; set; }

public string City { get; set; } = string.Empty;

public string Street { get; set; } = string.Empty;

public string PostalCode { get; set; } = string.Empty;

}

public class HouseBuildingDetail

{

[Required]

public int Id { get; set; }

[Required]

public int SquareRoot { get; set; }

[Required]

public int NumOfRooms { get; set; }

public int NumOfBathrooms { get; set; }

public bool HasSwimmingPool { get; set; } = false;

public bool IsPrivate { get; set; }=false;

public int Level { get; set; }

}

### House DB

In HousingDB project, make sure to have the following members in the HousingDBContext:

public DbSet<House> Houses{ get; set; }

public DbSet<HouseLocation> HouseLocations { get; set; }

public DbSet<HouseBuildingDetail> HouseBuildingDetails { get; set; }

If you didn’t have the above members, don’t forget to update the database (See above sections)

### Blazor Pages

In HousingBlazorApp project, under Pages folder, create new folder called: House (Optional)

Under House folder, right-click🡪Add🡪Blazor Component🡪Razor Component  
Call it: NewHouseComponent.razor and then click on Add

@page "/house/new"

@using HousingModel

@using Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage

@using Microsoft.AspNetCore.Components.Forms

@inject NavigationManager navigationManager

@inject ProtectedSessionStorage protectedSessionStore

<div class="new-house-form">

<h2>Create New House</h2>

<form>

<!-- Display error message - just in case, with red, large bold font-->

@if (error!=null&&error.Length > 0)

{

<div class="text-danger">@error</div>

}

@if (NewHouse != null)

{

<div class="form-group">

<label for="price">Price:</label>

<input type="number" id="price" @bind="@NewHouse.Price" required/>

</div>

<div class="form-group">

<label for="description">Description:</label>

<textarea class="form-control" id="description" @bind="@NewHouse.Description"></textarea>

</div>

<div class="form-group">

<label for="mainThumbnail">Main Thumbnail:</label>

<**InputFile** id="mainThumbnail" class="shadow bg-gradient" title="House Thumbnail" **OnChange**="HandleMainThumbnailUpload" accept=".png,.jpg,.jpeg"/>

</div>

<div class="form-group">

<label for="thumbnail1">Thumbnail 1:</label>

<**InputFile** id="thumbnail1" class="shadow bg-gradient" title="House Thumbnail" **OnChange**="HandleThumbnail1Upload" accept=".png,.jpg,.jpeg" />

</div>

<div class="form-group">

<label for="thumbnail2">Thumbnail 2:</label>

<**InputFile** id="thumbnail2" class="shadow bg-gradient" title="House Thumbnail" **OnChange**="HandleThumbnail2Upload" accept=".png,.jpg,.jpeg" />

</div>

<div class="form-group">

<label for="thumbnail3">Thumbnail 3:</label>

<**InputFile** id="thumbnail3" class="shadow bg-gradient" title="House Thumbnail" **OnChange**="HandleThumbnail3Upload" accept=".png,.jpg,.jpeg" />

</div>

<div class="form-group">

<label for="thumbnail4">Thumbnail 4:</label>

<**InputFile** id="thumbnail4" class="shadow bg-gradient" title="House Thumbnail" **OnChange**="HandleThumbnail4Upload" accept=".png,.jpg,.jpeg" />

</div>

<h3>Building Details</h3>

<div class="form-group">

<label for="squareRoot">Square Root:</label>

<input type="number" class="form-control" id="squareRoot" @bind="@NewHouse.BuildingDetails.SquareRoot" />

</div>

<div class="form-group">

<label for="numOfRooms">Number of Rooms:</label>

<input type="number" class="form-control" id="numOfRooms" @bind="@NewHouse.BuildingDetails.NumOfRooms" />

</div>

<div class="form-group">

<label for="numOfBathrooms">Number of Bathrooms:</label>

<input type="number" class="form-control" id="numOfBathrooms" @bind="@NewHouse.BuildingDetails.NumOfBathrooms" />

</div>

<div class="form-group">

<label for="isPrivateToggle">Is Private:</label>

<div class="toggle-switch">

<input type="checkbox" id="isPrivateToggle" class="toggle-input" @bind="NewHouse.BuildingDetails.IsPrivate" />

<label for="isPrivateToggle" class="toggle-label"></label>

</div>

</div>

<div class="form-group">

<label for="hasSwimmingPoolToggle">Has Swimming Pool:</label>

<div class="toggle-switch">

<input type="checkbox" id="hasSwimmingPoolToggle" class="toggle-input" @bind="NewHouse.BuildingDetails.HasSwimmingPool" />

<label for="hasSwimmingPoolToggle" class="toggle-label"></label>

</div>

</div>

<div class="form-group">

<label for="level">Level:</label>

<input type="number" class="form-control" id="level" @bind="@NewHouse.BuildingDetails.Level" />

</div>

<h3>Location</h3>

<div class="form-group">

<label for="country">Country:</label>

<input type="text" class="form-control" id="country" @bind="@NewHouse.Location.Country" />

</div>

<div class="form-group">

<label for="city">City:</label>

<input type="text" class="form-control" id="city" @bind="@NewHouse.Location.City" />

</div>

<div class="form-group">

<label for="street">Street:</label>

<input type="text" class="form-control" id="street" @bind="@NewHouse.Location.Street" />

</div>

<div class="form-group">

<label for="postalCode">Postal Code:</label>

<input type="text" class="form-control" id="postalCode" @bind="@NewHouse.Location.PostalCode" />

</div>

<div class="form-group align-content-center">

<button type="button" class="btn btn-primary" @onclick="CreateNewHouse">Create</button>

</div>

}

else

{

<p>Preparing...</p>

}

</form>

</div>

@code {

private House NewHouse = null;

private string error = null;

protected override void OnAfterRender(bool firstRender)

{

base.OnAfterRender(firstRender);

if (firstRender)

{

NewHouse = new House();

error = "";

StateHasChanged();

}

}

private bool ValidateFormValues()

{

if (NewHouse.Price < 1)

{

error = "Price cannot be negative.";

return false;

}

//main thumbnail should be valid - not empty

if (string.IsNullOrEmpty(NewHouse.MainThumbnailBase64))

{

error = "Main Thumbnail is required.";

return false;

}

//num of rooms and num of bath rooms should be positive

if (NewHouse.BuildingDetails.NumOfRooms < 1)

{

error = "Number of rooms should be positive.";

return false;

}

//num of bath rooms should be positive

if (NewHouse.BuildingDetails.NumOfBathrooms < 1)

{

error = "Number of bathrooms should be positive.";

return false;

}

//country and city should be valid

if (string.IsNullOrEmpty(NewHouse.Location.Country) || string.IsNullOrEmpty(NewHouse.Location.City))

{

error = "Country and City are required.";

return false;

}

return true;

}

private async Task CreateNewHouse()

{

error = "";

// Perform validation if needed

if (!ValidateFormValues())

{

StateHasChanged(); //display error message

return;

}

// Then save NewHouse to the database or perform other actions

// Example: dbContext.Houses.Add(NewHouse); dbContext.SaveChanges();

//navigate to house list page

navigationManager.NavigateTo("/house");

}

private async Task<string?> HandleThumbnailUpload(InputFileChangeEventArgs e)

{

var thumbnailFile = e.GetMultipleFiles().FirstOrDefault();

if (thumbnailFile != null)

{

using MemoryStream ms = new();

var resized = await thumbnailFile.RequestImageFileAsync(thumbnailFile.ContentType, maxWidth: 500, maxHeight: 800);

using Stream fileStream = resized.OpenReadStream();

await fileStream.CopyToAsync(ms);

var fileString = $"data:{thumbnailFile.ContentType};base64,{Convert.ToBase64String(ms.ToArray())}";

return fileString;

}

return null;

}

private async Task HandleThumbnail4Upload(InputFileChangeEventArgs e)

{

var code = await HandleThumbnailUpload(e);

if (code != null)

NewHouse.Thumbnail4Base64 = code;

}

private async Task HandleThumbnail3Upload(InputFileChangeEventArgs e)

{

var code = await HandleThumbnailUpload(e);

if (code != null)

NewHouse.Thumbnail3Base64 = code;

}

private async Task HandleThumbnail2Upload(InputFileChangeEventArgs e)

{

var code = await HandleThumbnailUpload(e);

if (code != null)

NewHouse.Thumbnail2Base64 = code;

}

private async Task HandleThumbnail1Upload(InputFileChangeEventArgs e)

{

var code = await HandleThumbnailUpload(e);

if (code != null)

NewHouse.Thumbnail1Base64 = code;

}

private async Task HandleMainThumbnailUpload(InputFileChangeEventArgs e)

{

var code = await HandleThumbnailUpload(e);

if (code != null)

NewHouse.MainThumbnailBase64 = code;

}