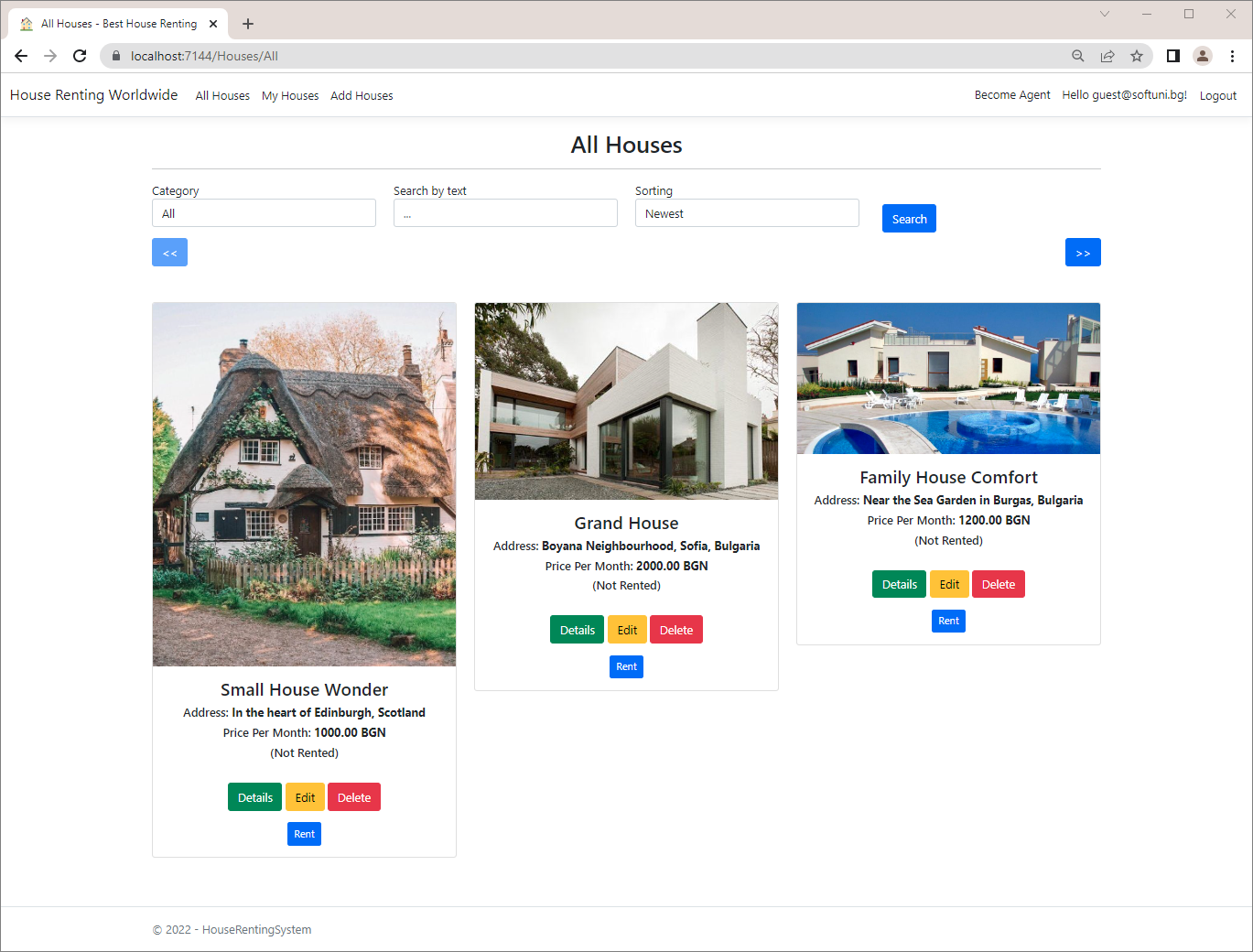
# Workshop: Project Fundamentals

Workshop for the ["ASP.NET Advanced" course @ SoftUni](https://softuni.bg/trainings/4369/asp-net-advanced-february-2024)

The "House Renting System" **ASP.NET Core MVC App** is a Web application for **house renting**. Users can look at **all** **houses** with their **details**, **rent a house** and look at **their rented houses**. They can also **become** Agents. Agents can **add houses**, see their **details** and **edit** and **delete** only **houses they added**. The Admin has **all privileges** of Users and Agents and can see **all registrations** in the app and **all made rents**.

We will implement the app during the workshops in the course.

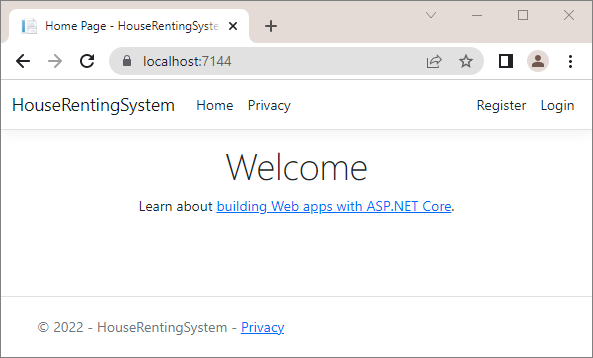


## Create the Project

Our first task is to create an **ASP.NET Core MVC application** in **Visual** **Studio**. Open Visual Studio and create a new **ASP.NET Core Web App (Model-View-Controller)** with and **Individual Accounts** **Authentication** **type**, as we want to have "**Register**" and "**Login**" functionalities. The app name should be "HouseRentingSystem".

## Examine the App in the Browser

**Run** the created app in the **browser**. It should have **four pages** for now – "Home", "Privacy", "Register" and "Login" **pages**. The "Home" **page** looks like this:

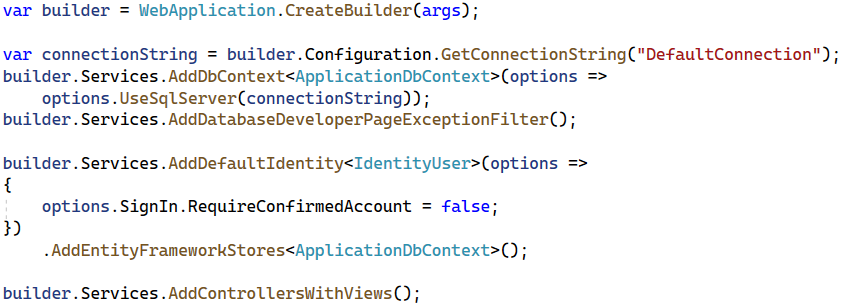
****

## Clean Project

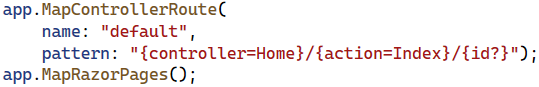
As you know, **ASP.NET Core** gives us a pretty good **MVC template** to work on. However, we should now define our **own style of writing** **and** **formatting code**, which will be used for the **whole app**. We will do this as it is important for our code to **look good** and be **cleaner** and **more readable**. In addition, in this way we will also **examine the** **project files and classes** better.

### Step 1: Modify the Program class

First, go to **Program.cs** and look at how it is written. We should remove code comments. Now improve the configuration of the services. It is good for all commands to have an empty line between them. It should look like this:



You can see that the **controller route mapping** is the **default** one.

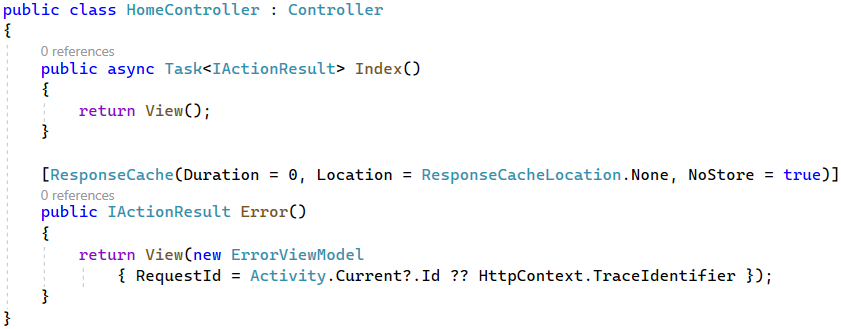


For this reason, you can replace the **MapControllerRoute(…)** method with the **MapDefaultControllerRoute()** one:



### Step 2: Modify the HomeController Class

Now let's clean the HomeController **class**. To start with, we won't need a **logging functionality** in our app, so remove the logger **property** and the **whole class constructor**, which initializes it. We can also make the **actions** with **arrow functions** and remove the "**Privacy**" page from our app, as we won't be needing it.

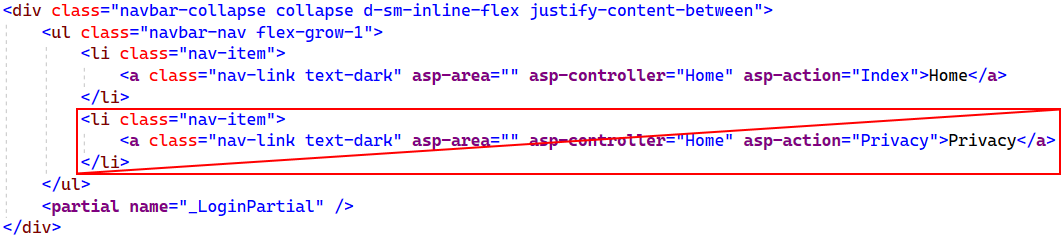


We should **remove** the "Privacy.cshtml" **view** from our project. Find it in the "/Views/Home" **folder** and **delete it**:

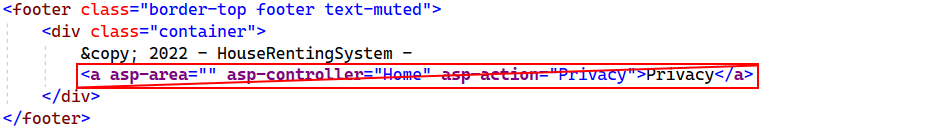
### Step 3: Modify the \_Layout.cshtml File

The "\_Layout.cshtml" **view** in our app defines how our **header** and **footer look**.

Go to the **view file** in the "/Views/Shared" **folder** and let's make some changes. Start by **removing** the "Privacy" **page links** from the **header** and the **footer**, as we deleted the page. Remove the following lines from the **header**:



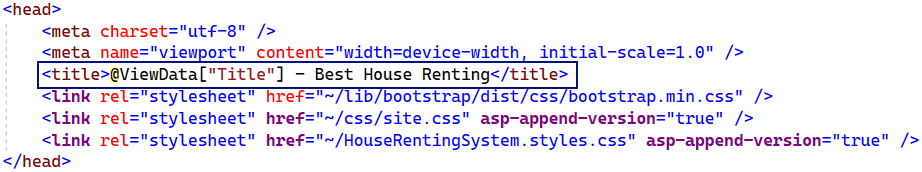
Then, **remove** the <a> **tag** with the link from the **footer**:



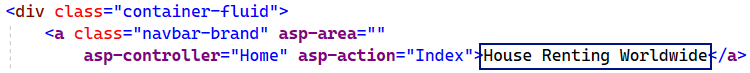
You can also replace the year with **@DateTime.Now.Year**:



Modify the **title** of the app in the <head> **tag** and **write a meaningful one**:

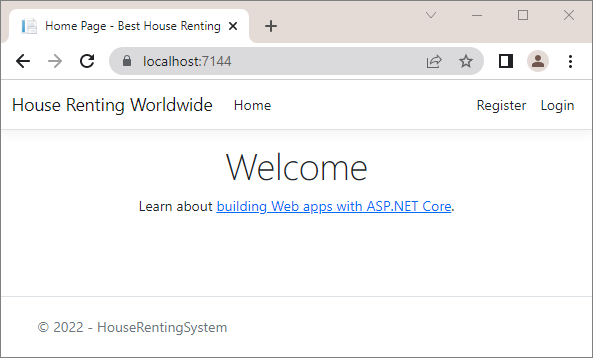


Then, **change the name** of the app in the **header**, as well:



This **name** is in an <a> **tag**, as it is a **hyperlink** to the **main page** of the app.

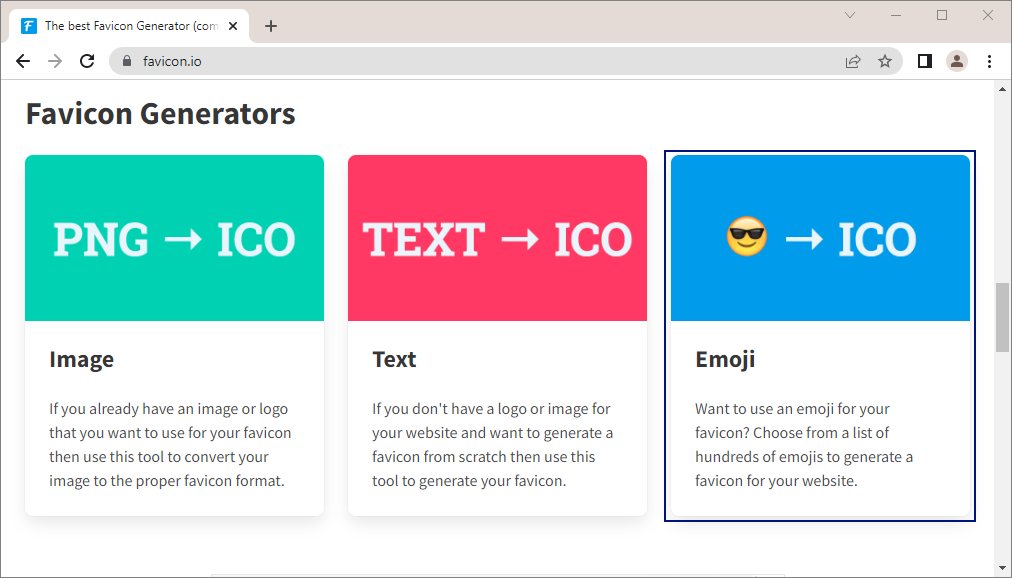
Now the **result** in the **browser** is the following:

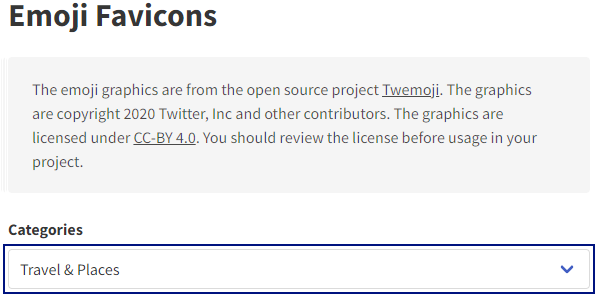


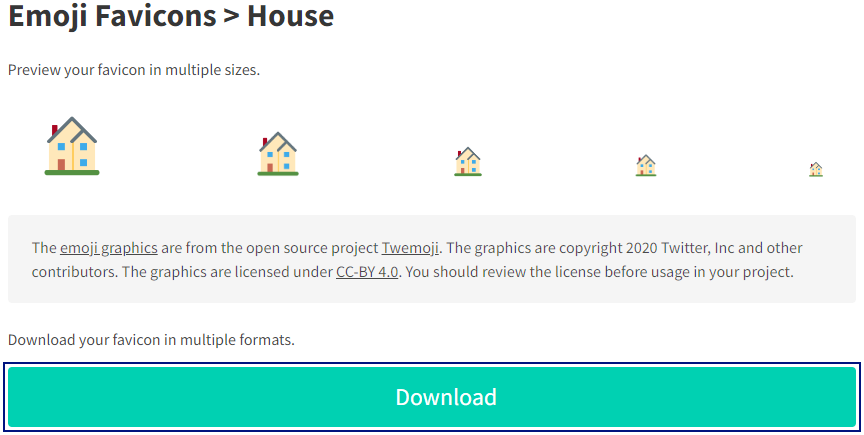
## Add Favicon



In this task, we will see how to **add a favicon** in our **site's tab** in the **browser**. To do this, we will first go to <https://favicon.io/> and **generate a favicon**. **Create** one by yourself and **download** it like this:



 🡪 

 🡪 Logo, company name

Description automatically generated

**Open** the .zip **file** and **search for** the favicon.ico **file**:

Graphical user interface, text, application

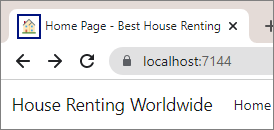
Description automatically generated

You should now **copy** and **paste this file** in the **project**. **Open the project** in File Explorer and look for the favicon.ico **file** in the "wwwroot" **directory**. **Replace the old favicon** with the **new** one:

Graphical user interface, application

Description automatically generated

When you are ready, **reload the project** in **VS** and **run it in the browser**. The **new favicon** should be visible in the **browser tab**:



## Create Entity Model Classes

Now, we will create the **data models**, which we will need for our **database**. We will have **three data model** **classes** – House, Category and Agent.

Create the above classes in the "Data" **folder** of the project. They should be in a **separate folder** from the ApplicationDbContext class – the "Models" **folder**.

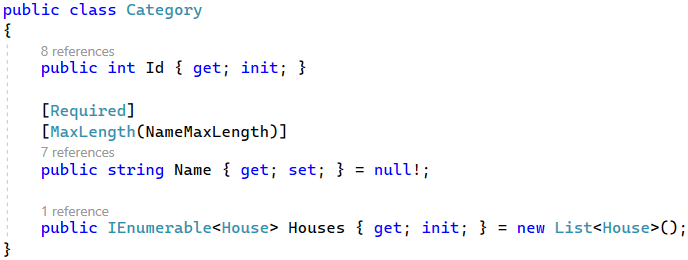
First, let's create a separate class with constants for the max and min length values. In the "**Data**" folder, create the **DataConstants** class.

### Step 1: Add Category Entity Class

The Category **class** should have the following **properties**:

* Id – a unique integer, Primary Key
* Name – a string with max length **50** (**required**)
* Houses – a **collection** of House

Our first entity class should look like this:



### Step 2: Add House Entity Class

The House class should have the following **properties**:

* Id – a unique **integer**, Primary Key
* Title – a string with min length **10** and max length **50** (**required**)
* Address – a string with min length **30** and max length **150** (**required**)
* Description – a string with min length **50** and max length **500** (**required**)
* ImageUrl – a string (**required**)
* PricePerMonth – a decimal with min value **0** and max value **2000** (**required**)
* CategoryId – an integer (**required**)
* Category – a Category object
* AgentId – an integer (**required**)
* Agent – an Agent object
* RenterId – a string

### Step 3: Add Agent Entity Class

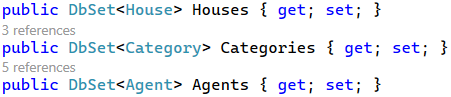
The Agent **class** should have the following **properties**:

* Id – a unique integer, Primary Key
* PhoneNumber – a string with min length **7** and max length **15** (**required**)
* UserId – a string (**required**)
* User – an IdentityUser object

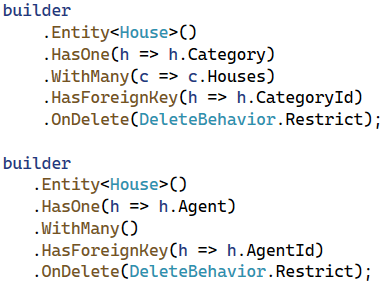
## Modify DbContext Class

As we now have **all entity classes** your app needs, use them for the database. To start with, it is a good idea to **rename** the ApplicationDbContext **class** to "HouseRentingDbContext", so that it is connected to the idea of our application.

Create DbSet **properties** for **all tables** in the **database**:



Next, we should **override** the OnModelCreating(ModelBuilder builder) **method** in the HouseRentingDbContext **class**:



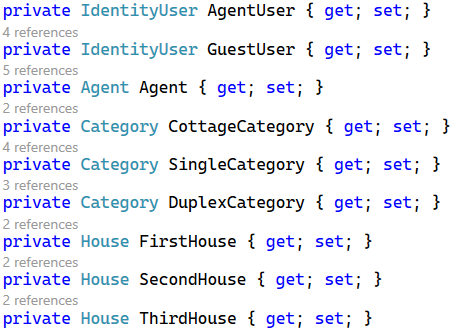
Don't forget to invoke the base **OnModelCreating()** method at the end.

Now our **database structure** is ready. If you migrate it now, however, it will be **created with empty tables**. For this reason, let's **seed some data** **to fill in the database tables**.

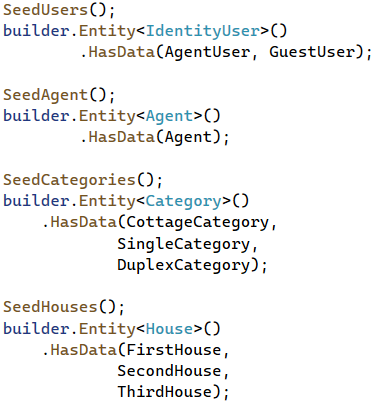
## Seed Database

Now we need to **populate the database** with an **initial set of data**. This will include **two users**, an **agent**, **three** **categories** and **three houses**.

First, **create properties** for the above **objects** in the HouseRentingDbContext **class**:



Then, we will use **separate methods** to **add data to these objects**, which will be **added to the corresponding database** **tables** in the OnModelCreating(…) **method**. Add the following lines to the method, before invoking the base one:



As you remember from the previous workshops, it is important that the above **seeding methods are invoked in the correct order**, as they **depend** on each other.

As the methods have a lot of data, you can copy the code from here:

|  |
| --- |
| private void SeedUsers()  {  var hasher = new PasswordHasher<IdentityUser>();  AgentUser = new IdentityUser()  {  Id = "dea12856-c198-4129-b3f3-b893d8395082",  UserName = "agent@mail.com",  NormalizedUserName = "agent@mail.com",  Email = "agent@mail.com",  NormalizedEmail = "agent@mail.com"  };  AgentUser.PasswordHash =  hasher.HashPassword(AgentUser, "agent123");  GuestUser = new IdentityUser()  {  Id = "6d5800ce-d726-4fc8-83d9-d6b3ac1f591e",  UserName = "guest@mail.com",  NormalizedUserName = "guest@mail.com",  Email = "guest@mail.com",  NormalizedEmail = "guest@mail.com"  };  GuestUser.PasswordHash =  hasher.HashPassword(AgentUser, "guest123");  }  private void SeedAgent()  {  Agent = new Agent()  {  Id = 1,  PhoneNumber = "+359888888888",  UserId = AgentUser.Id  };  }  private void SeedCategories()  {  CottageCategory = new Category()  {  Id = 1,  Name = "Cottage"  };  SingleCategory = new Category()  {  Id = 2,  Name = "Single-Family"  };  DuplexCategory = new Category()  {  Id = 3,  Name = "Duplex"  };  }  private void SeedHouses()  {  FirstHouse = new House()  {  Id = 1,  Title = "Big House Marina",  Address = "North London, UK (near the border)",  Description = "A big house for your whole family. Don't miss to buy a house with three bedrooms.",  ImageUrl = "https://www.luxury-architecture.net/wp-content/uploads/2017/12/1513217889-7597-FAIRWAYS-010.jpg",  PricePerMonth = 2100.00M,  CategoryId = DuplexCategory.Id,  AgentId = Agent.Id,  RenterId = GuestUser.Id  };  SecondHouse = new House()  {  Id = 2,  Title = "Family House Comfort",  Address = "Near the Sea Garden in Burgas, Bulgaria",  Description = "It has the best comfort you will ever ask for. With two bedrooms, it is great for your family.",  ImageUrl = "https://cf.bstatic.com/xdata/images/hotel/max1024x768/179489660.jpg?k=2029f6d9589b49c95dcc9503a265e292c2cdfcb5277487a0050397c3f8dd545a&o=&hp=1",  PricePerMonth = 1200.00M,  CategoryId = SingleCategory.Id,  AgentId = Agent.Id  };  ThirdHouse = new House()  {  Id = 3,  Title = "Grand House",  Address = "Boyana Neighbourhood, Sofia, Bulgaria",  Description = "This luxurious house is everything you will need. It is just excellent.",  ImageUrl = "https://i.pinimg.com/originals/a6/f5/85/a6f5850a77633c56e4e4ac4f867e3c00.jpg",  PricePerMonth = 2000.00M,  CategoryId = SingleCategory.Id,  AgentId = Agent.Id  };  } |

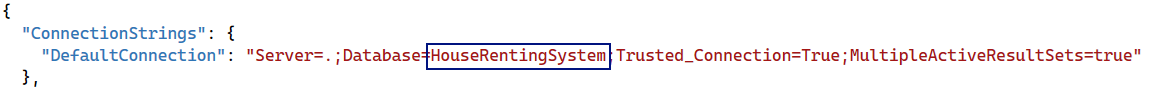
Note that only the **first house** has a RenterId, which means that it is **rented by the** GuestUser.

Now we have a db context with **seeded data** and our **database is ready to be migrated**.

## Create a Migration

We will now **create a migration** to the database. Before that, however, let's give the **database a good name**.

To do this, **edit the connection string** in the "appsettings.json" **file**. Set "Database" **name** to be "HouseRentingSystem":

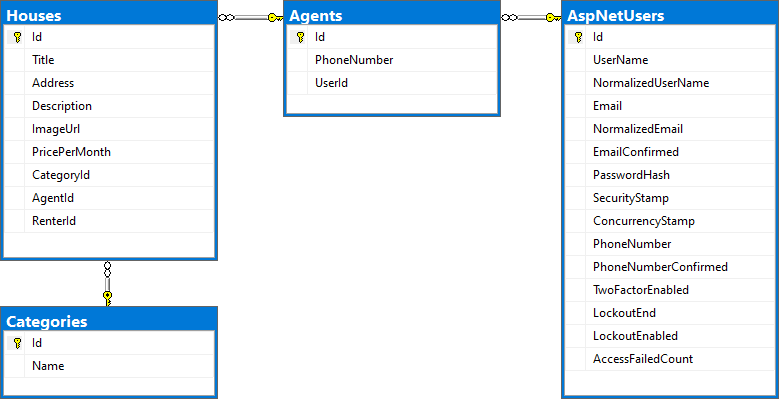


Next, open the Package Manager Console to write **commands** for **managing migrations**. In the **console**, write a command for **adding a migration** to the "Data/Migrations" **folder** with a given **name** and **press** [Enter] to **execute it**. Now you should have a **new migration** in the "Migrations" **folder**.

**Examine the tables** and its **restrictions** in the **new migration** – if something is wrong, **delete the migration** with the "Remove-Migration" **command** or **delete the migration file**. Don't forget that you should also **delete the database** from SQL Server Management Studio, or errors will appear.

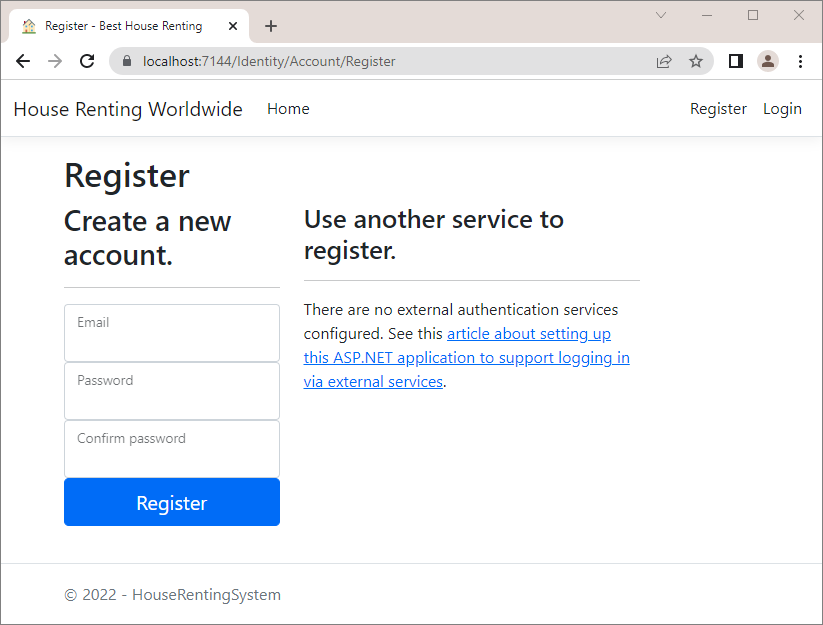
Now **run the app** in the browser – there should not be **any errors**. Then, look at the **newly-created database** in SQL Server Management Studio and **examine its tables** – all tables we created should be **present** and have the **right restrictions** and **relationships**.

Examine the **diagram of the database**, as well. It should look like this:

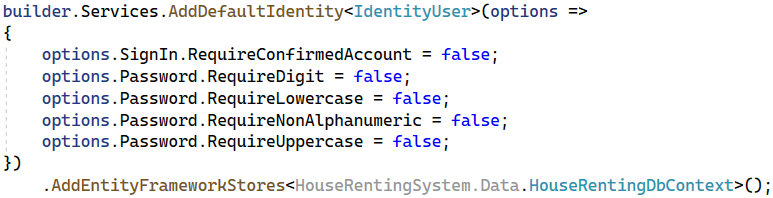


## Register and Log in the App

Go to the "Register" **page** in the "HouseRentingSystem" **app** and you should see the **registration form**:



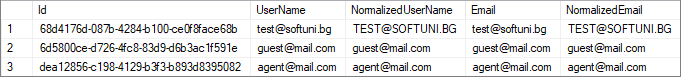
If you try to **fill in the form**, you will see that **password requirements are too strict** – change this by adding the following lines in Program.cs:



After this, **fill in** the "Register" **form** with **valid data** and you should be **logged**:



Also, the **new user** should be part of the **database** – check in the "AspNetUsers" **table**:



## Modify Navigation

Now, we should **change the navigation menu** to have **links** to the "All Houses", "My Houses", "Add House" and "Become Agent" **pages** when the **user is logged-in**.

When the **user is not logged-in**, the **navigation** should look like this:

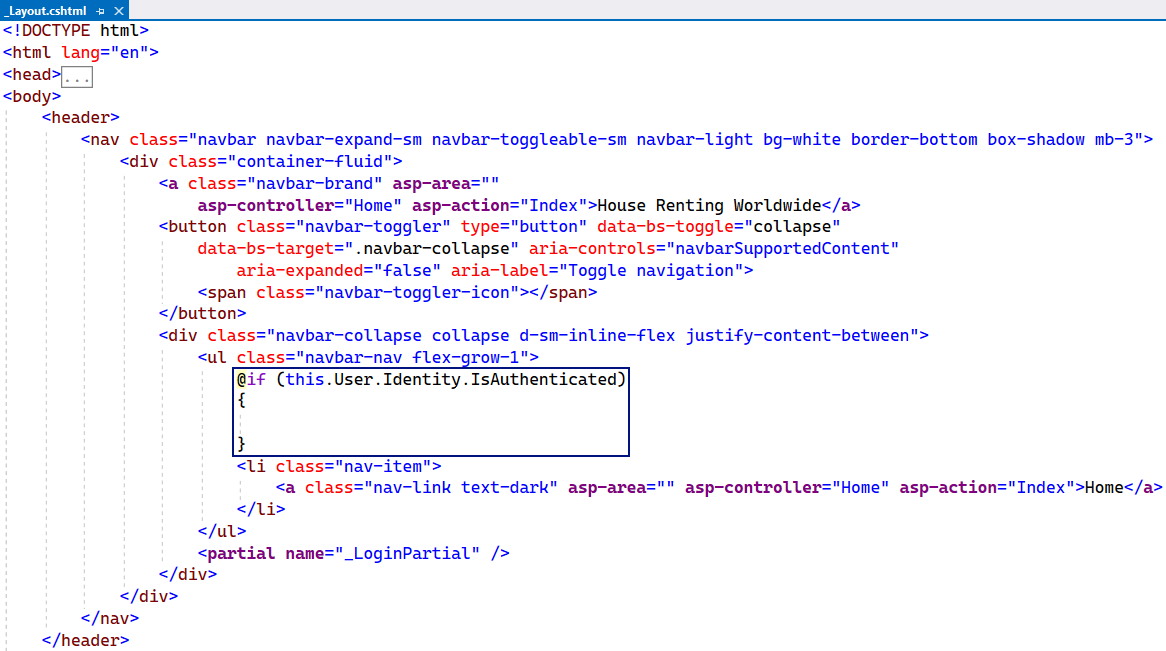


When the **user is logged-in**, it should be the following:

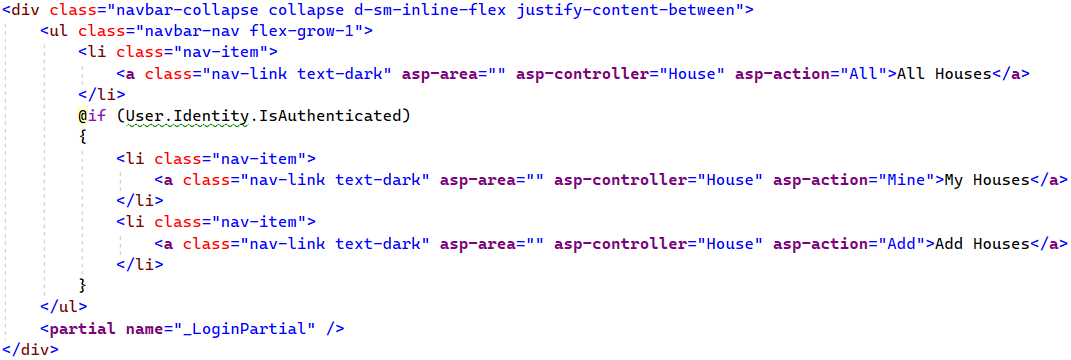


To do this, we need to **modify** the "\_Layout.cshtml" and "\_LoginPartial.cshtml" **views** in the "/Views/Shared" **folder**, as they are **responsible for the navigation menu**.

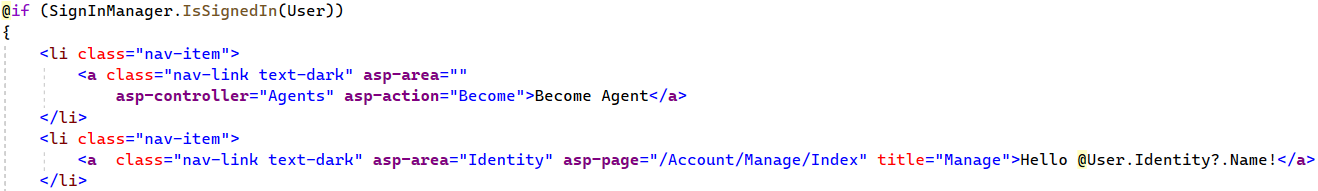
First, to **change links on the left side** of the page, go to the "\_Layout.cshtml" file. In it, we want to **add links to the** **pages** depending on whether the **user is authenticated**. Use an if **statement** in the **Razor view** and the Identity.IsAuthenticated **property** to check whether the **current user is authenticated**:



Then, outside of the if **statement**, add the "All Houses" **page** **link**, as the page should be **accessible to anyone**. In the **statement**, add **links** to the "My Houses" and "Add House" **pages**, which are **only for authenticated users**. Use the asp-controller and asp-action **tag** **helpers**, so that **links point to the correct controller action**. Don't forget to remove the link to the "**Home**" page.



Now go to the "\_LoginPartial.cshtml" **view** to add the "Become Agent" **page link**. It should be **only for logged-in** **users** (later we will see how only non-agents to see it), so find the right place in the view code and **add a link** like this:



Try out the **navigation bar links** in the browser – they should **redirect to the correct pages**.

## Create the MVC Structure of the App

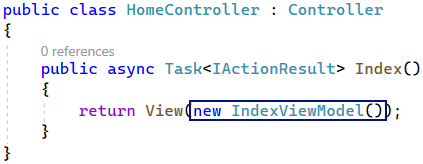
In our "HouseRentingSystem" **app** we will have **three controller classes –** HomeController, HouseController and AgentController. Now we will just **create the controller actions** with the **attributes** and **models** they need, but we **won't implement them** yet, as we will do this later in the workshop.

### Step 1: Modify the HomeController Class

We already have the HomeController in our **project** but we will modify it a bit, as we want it to **return a model**.

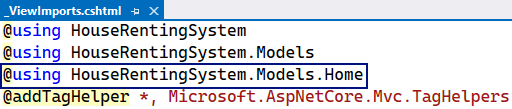
Start by creating a "Home" **folder** in the "Models" **folder** of our project – this is where the **models** for our HomeController should be created. Then, create the IndexViewModel **class** in the "/Models/Home" **folder**.

Use the **model** and **pass it to the view** in the Index() **method** of the HomeController **class**:

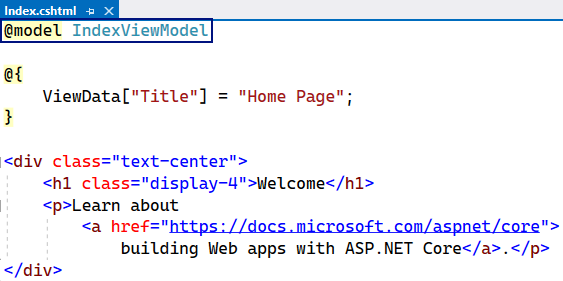


Now we should go to the "Index.cshtml" **view** in the "/Views/Home" **folder** and modify it to **accept an** IndexViewModel.

Before we write in the **view**, we need to go to the "\_ViewImports.cshtml" **file** and **add the** "HouseRentingSystem.Models.Home" **namespace**, so that the **views can use the models** in our "/Models/Home" **folder** like this:



Add the IndexViewModel to the "Index.cshtml" **view** like this:



### Step 2: Create the HouseController Class

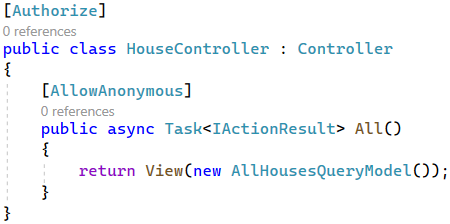
The HouseController will have all **actions** for CRUD **operations** on **houses** and **their renting**.

First, create the HouseController **class** in the "Controllers" **folder** of the "HouseRentingSystem" **project**.

The class should have some **methods,** which (for now) we will create **without implementing them fully**.

Start with writing the All() **method**, which should **return a view** with **all houses**. Create the AllHousesQueryModel **model**, which will later get information from the **request query** and use it to generate a **view**, in the "Models" **folder**. Create a "Houses" **folder** for **house models** and **add the above model class**.

We will leave the **model class** like this for now and **add properties** **later**. Go back to the HouseController and write the All() **method** to **return a view** with the newly-created **model**.



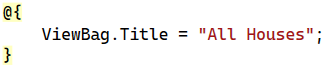
As the actions returns a view, let's **create that view**. To do this, create a "Houses" (the name of the **controller**) **folder** in the "Views" **folder** of the project and then **create an empty** Razor **view** named "All.cshtml" (the name of the **action**).

Before we write in the **view**, we need to go to the "\_ViewImports.cshtml" **file** and **add the** "HouseRentingSystem.Models.Houses" **namespace**, so that the **views can use the models** in our "/Models/Houses" **folder**.

Now, in the All **view**, use the @model **directive**, which specifies what **type** the view should accept from the **controller** **action**. In this case, we accept a AllHousesQueryModel **model**:



Then, we will **set a page title** with ViewBag and then use it in a **page heading**. For the ViewBag, we will use the @ **symbol** to **open a code block** and **write C#** in it like this:

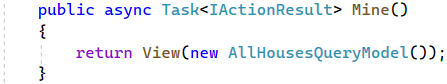


The rest of the "All.cshtml" **file** looks like this:

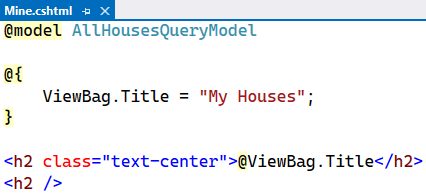


We will improve it later in the workshop.

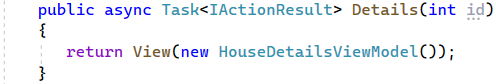
Go back to the HouseController **class** and continue with writing its **methods**. The next **action** is Mine(), which should also **return a view with houses**. Use the [Authorize] **attribute** to make the method accessible only for **authorized users**:



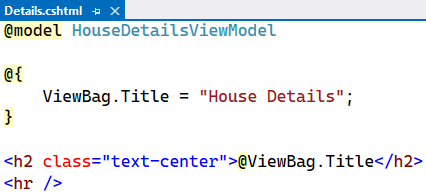
The **method** uses the AllHousesQueryModel **model**, which we have created already. **Create a view**, similar to the one we just created, which will be returned:



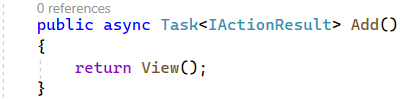
Next, add the Details(int id) **action** to the **controller class**, which should **return a view** with a **details model**:



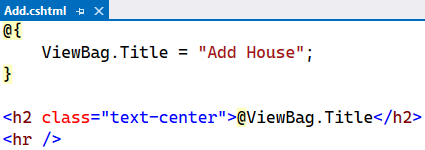
Create the HouseDetailsViewModel **class** in the "/Models/Houses" **folder** and **leave it empty** for now. Then, **write the method** to return a **view** and **create the view**, as well. The **view** should be the following:



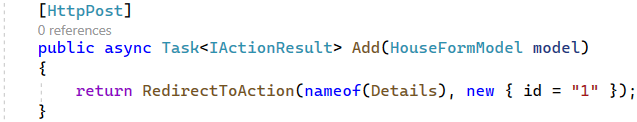
In the HouseController, add the Add() **method**, which should just **return a view**. Remember that only **authorized users** should be able to access the "Add House" **page**:



The "Add.cshtml" **view** should **contain a form for adding a new house**. However, we will **create the form later –** for now, the **view** should be the following:



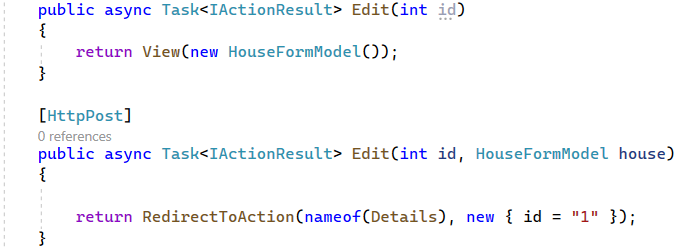
Next, write the Add(HouseFormModel house) **method**, which should **accept a model** from the **view** when we **create its form**. Create the HouseFormModel **class** in the "/Models/Houses" **folder** and **leave it empty** for now. The Add(HouseFormModel house) **method** should be invoked on a "POST" **request** and be for **authorized users** only:



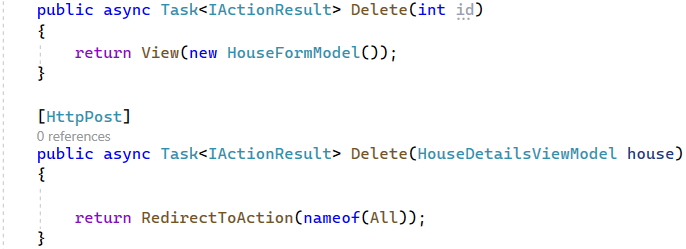
The method should **add a new house** to the **database** (which we will do later) and **redirect the user** to the "House Details" **page** of the **newly-created house**. The Details(int id) **controller** **action** accepts an id, but we will **hardcode** it for now.

Next **methods** to be implemented are for the "Edit House" **page**. The method on a "GET" **request** should accept a house id, so that we know which **house to edit** and **return a view** with the **current house information**:

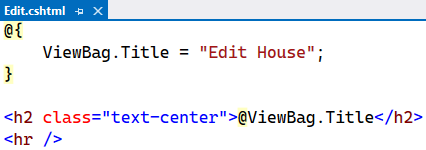
The **method** for the "POST" **request** to the "Edit House" **page** should accept a house id and a **model** and **redirect** the user to the "House Details" **page** after the house is **modified in the database**:

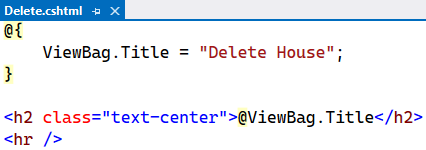


The **methods for** **deleting a house** are similar to the above methods, but they **accept** and **pass** a HouseDetailsViewModel **model**. After the **house is deleted**, the user should be **redirected** to the "All Houses" **page**. Write them like this:

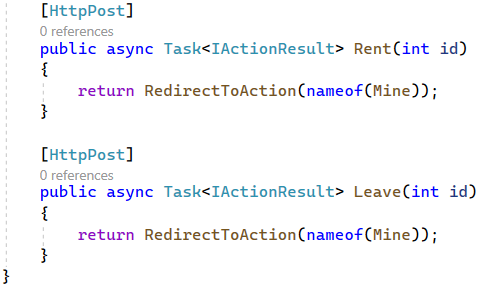


Create the "Edit.cshtml" and "Delete.cshtml" **views**, as we did with the previous ones:





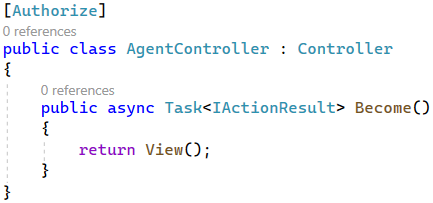
Finally, write the **methods** for the **house renting and leaving functionality**. They should be invoked on a "POST" **request** and **redirect** the user to the "My Houses" page:



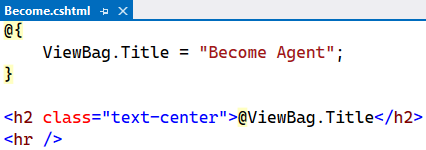
### Step 3: Create the AgentController Class

The AgentController is taking care of the **agents functionality**.

Create the AgentController in the "Controller" **folder** and write the Become() **method** like this:

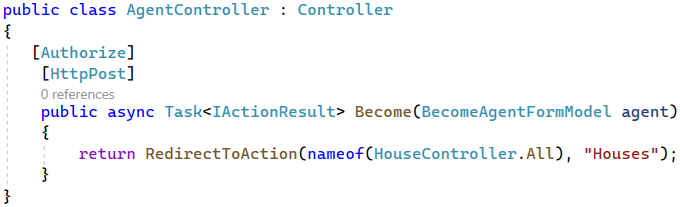


Create the "Become.cshtml" **view** in "/Views/Agents":



For the Become(BecomeAgentFormModel agent) **method** you should create the BecomeAgentFormModel in the "/Models/Agents" **folder**.

Write the **method** to be **invoked** on a "POST" **request** and **return a redirect response**:



Now we have written the **structure of our controllers** (HomeController, HouseController and AgentController) and their **actions**, as well as the **models** and **views** for our **MVC app**.

In the next workshop, we will start creating **services**, which will contain the **app's business logic** and **interact with the database**. Our **controllers** will use the **service methods** and depend only on them – they will **not have access to the database**. We will do this by creating **services with service models** and implementing the **logic in service methods**. Let's see how this is going to happen.