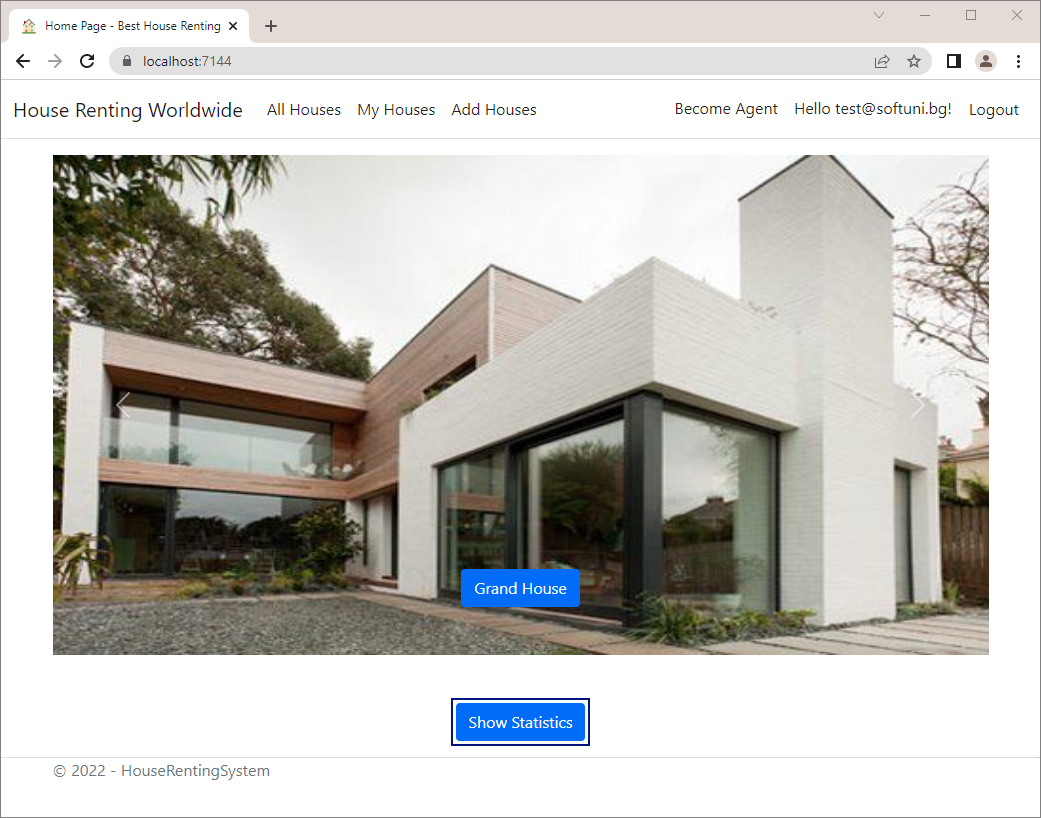
# Workshop: Security and Web API

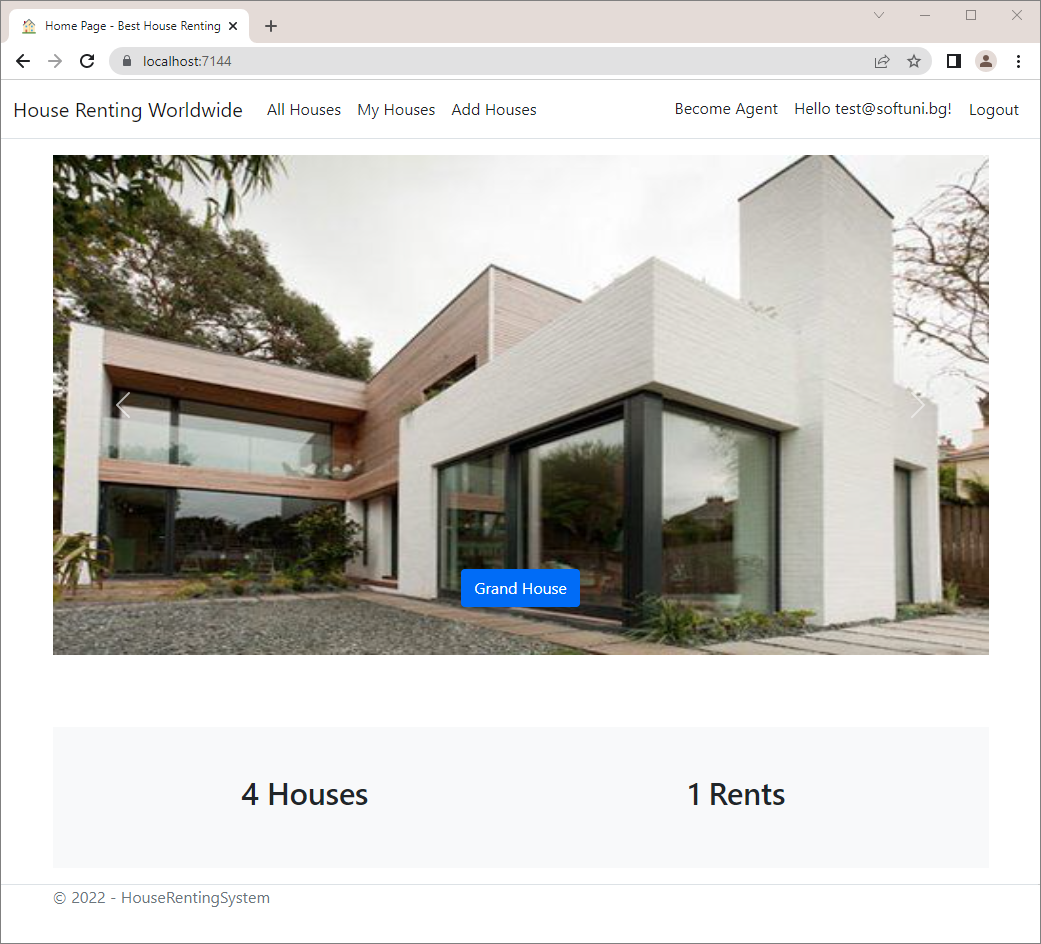
Workshop for the ["ASP.NET Advanced" course @ SoftUni](https://softuni.bg/trainings/4369/asp-net-advanced-february-2024)

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**.

## Create Web API

In this task, we want to **display statistics of houses and rents** in the bottom of the "Home" **page**. The page should have a [Show Statistics] **button**. When the **button is clicked**, the **total houses and total rents counts will be** **displayed**. It should look like this:

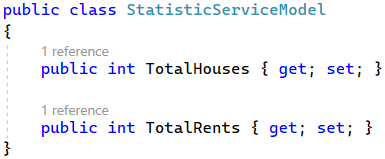




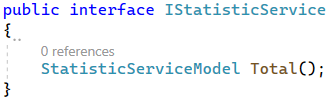
To do this, our view will use **JavaScript** **code** to send a "GET" **request**, which will **invoke an API controller method**. The **controller method** will use a **service** to **return the counts to the controller**, which will pass them to the **view**.

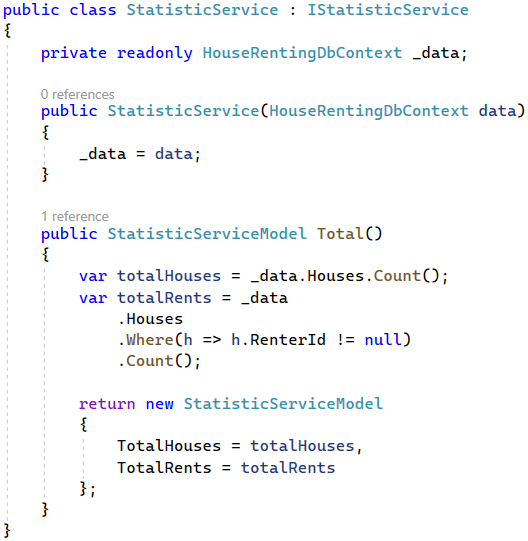
Start by **creating a service for the statistics**. Create the IStatisticsService **interface** in folder "/Contracts/Statistic and the StatisticService **class** in folder "/Services/Statistic".

They will have a **single method** to **return a model** **with** **total counts** from the database. Create the StatisticServiceModel in folder "/Models/Statistic" and **add properties** to it:

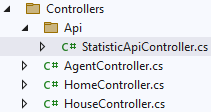


**Define** and **implement** the **service method**, which returns a StatisticServiceModel:

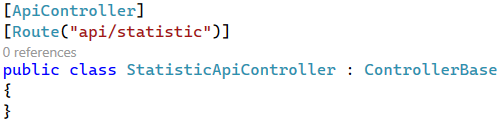




Now create the StatisticsApiController class in folder "/Controllers/Api". This will be the **API controller** **class**, which will use the **service** to **return responses on HTTP requests**:

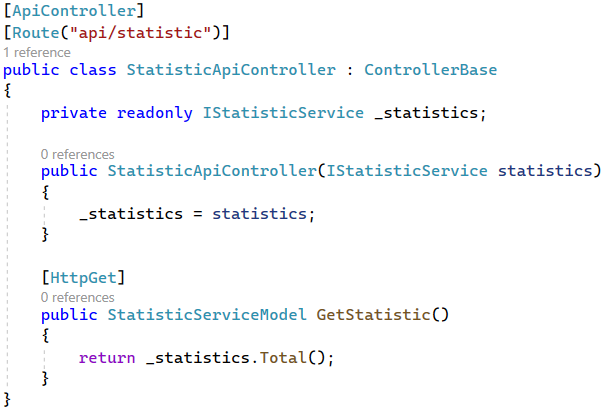
 

To make the StatisticApiController **class** work as an **API controller**, it should inherit the ControllerBase **class** and have the [ApiController] and [Route] **attributes**:



Note that we have **set the controller route** to be "api/statistic". This means that the **controller and its methods** will be **invoked on an HTTP request** to <https://localhost:44342/api/statistic>.

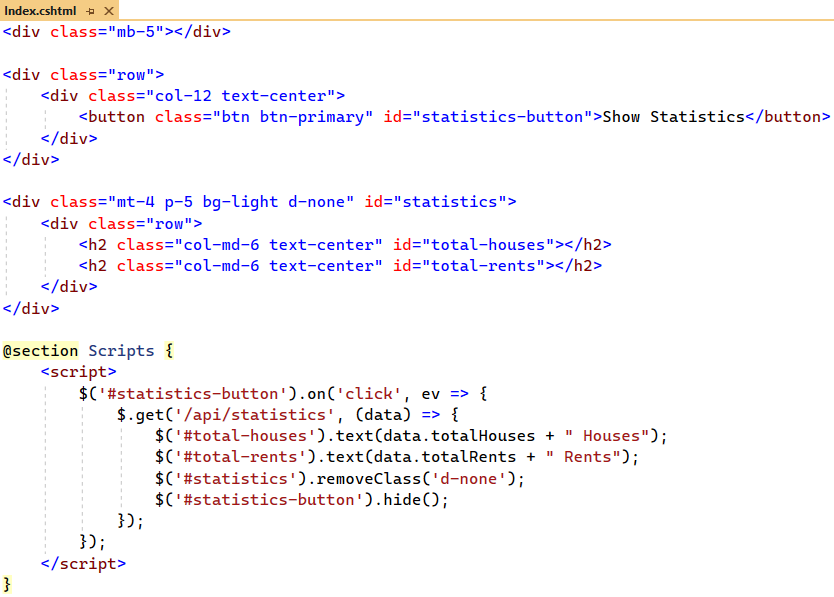
The **controller class** should use the **statistic interface** and should have a **single method** to return a StatisticServiceModel on a "GET" **request**. Do it like this:



Don't forget that you should **add the service** in Program.cs class to use it:



Finally, we should modify the Index.cshtml **view** to send a "GET" **request** to "api/statistic" and **display the** **returned data**. To do this, we will **create** a <div> and **use a JavaScript function** to **fill it with data**. Add the following code to the end of **Index.cshtml** it like this:



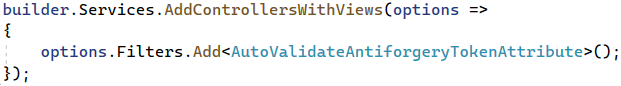
Now try out the **statistic functionality** in the browser. You should see the **total houses and rents count** when you **click** on the [Show statitics] **button**. Make sure everything works as expected.

## Secure the App Against CSRF

**Anti-forgery tokens** are a security mechanism to defend against **cross-site request forgery** (**CSRF**) **attacks**. The AutoValidateAntiforgeryTokenAttribute is a **global MVC filter** to automatically validate all appropriate action methods.

Now, we will **add a filter** to protect our app against **CSRF attacks**, **enhance** **URLs** that access the "Details" **page** and **inject** and **use services in views**.

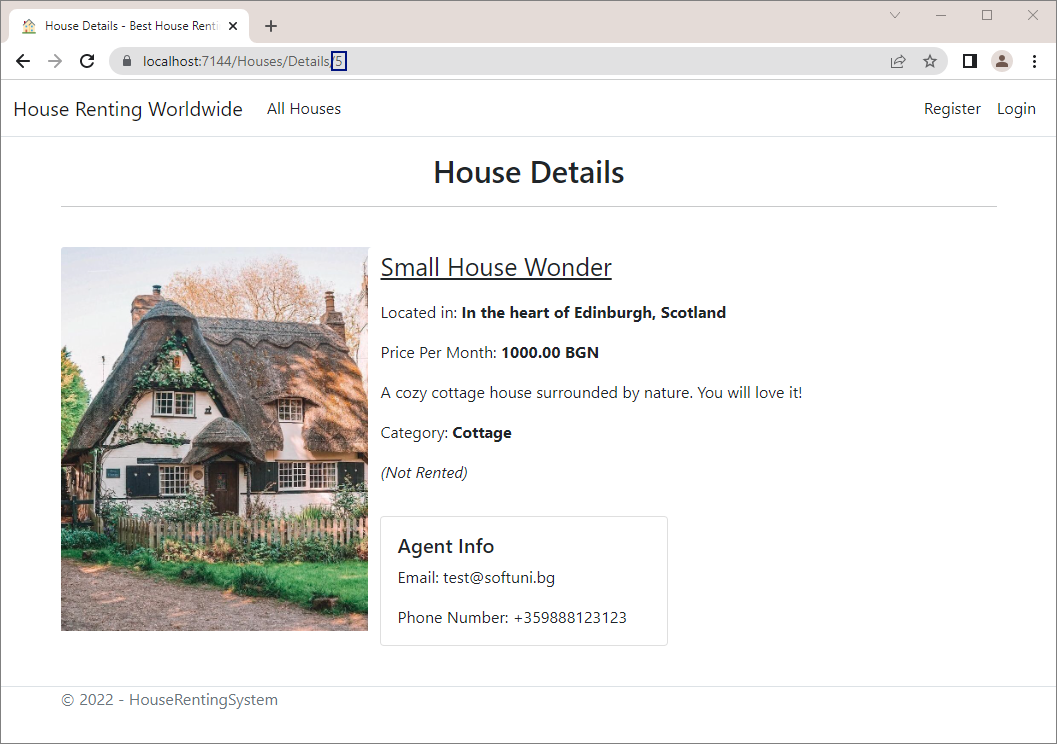
Go to the Program.cs of your app and **add the filter** to the AddControllersWithViews(…) **method** like this:



Now you are **secure against CSRF attacks**.

## Protect URLs

Our task now is to **modify the URLs** of the "Details" **page** to be **more secure** and, at the same time, more SEO-friendly. The reason for this change is that now everyone can **easily reach our pages** only by **changing the house id** **parameter** in the **URL**:



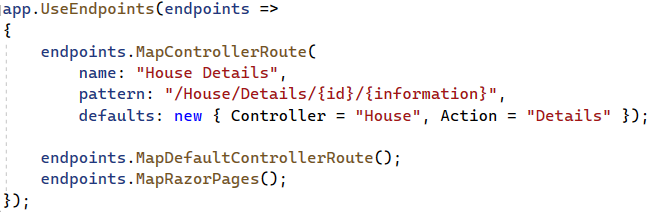
In this way, we are **not protected** from someone **stealing the information** through a **foreach**. To prevent this, we will add some **house information** (the **house title** and **part of the address**) to the **URL** like this:

Graphical user interface, website

Description automatically generated

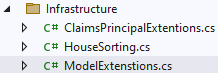
To do this, we should **modify our routes** to **include the information in the URL** and the "Details" **page** for a house to be accessed on "/House/Details/{id}/{information}". When we click on a [Details] **button**, we will **include this information**. Then, when we access the "Details" **page**, the **information will be checked**. If it is **missing** or **different**, a BadRequest will be returned.

Start by **creating a custom controller route** to include the house information. Go to the **Program class** and add the following code:



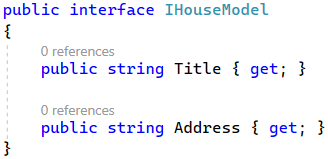
Because of this **endpoint**, now our Details(int id) **method** in the HouseController **accepts an additional parameter** **with information**. Before we **modify the method**, however, let's **create a method**, which will **generate the** **information** string.

As this is a **web-related job**, even though it will use service models, we will create an **extension class** called ModelExtensions in folder "/Infrastructure":

The class will have an **extension method** **for getting the house information**. We will need this method for **several** **models** (**HouseServiceModel**, **HouseIndexServiceModel**, etc.), but we **don't want to write methods for each one of them**. For this reason, we will **create an interface** with the **properties for the information** and **other classes will** **implement that interface**.

Create the IHouseModel **interface** in the "Contracts" folder and **define properties for title** **and** **address**:



Go back to the ModelExtensions **class** and **add a method for getting the house information**. In it, we should get the **title** and **the first three words of the address**. They should be **joined by a** **hyphen** "-" and should **not contain any** **other symbols**, except for letters, digits and hyphens. Write the class like this:

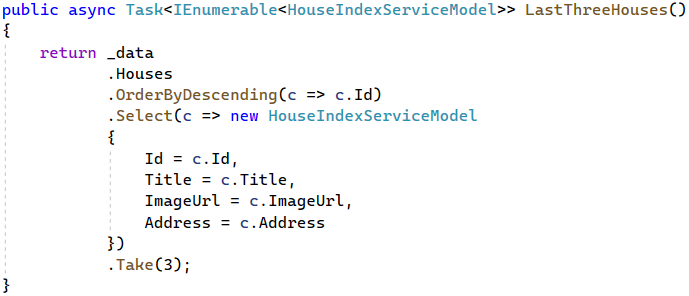


Now **modify the** [Details] **buttons** in the **views** to **add the house information** to the **URL** when they send a **request**. We should do this in the "\_HousePartial.cshtml" and "Index.cshtml" **views**. They **accept** and **pass** a HouseServiceModel and a HouseIndexServiceModel. To use the **extension method** we created, these two **model classes should implement** the IHouseModel **interface**:





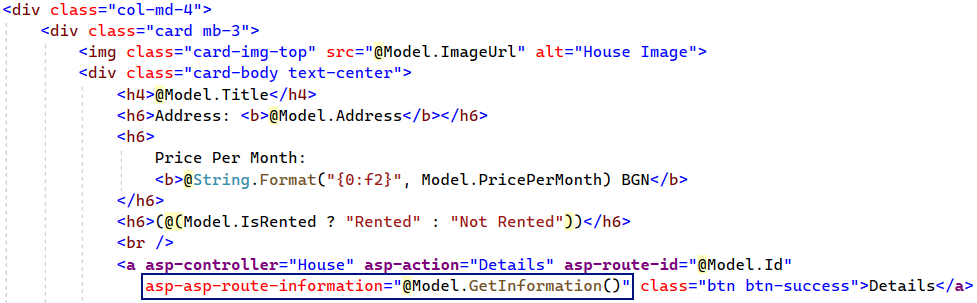
Note that we should add an Address **property** to the HouseIndexServiceModel to **implement the interface**. Don't forget to **assign a value** to the **property** in the LastThreeHouses() **method** of the HouseService **class**:



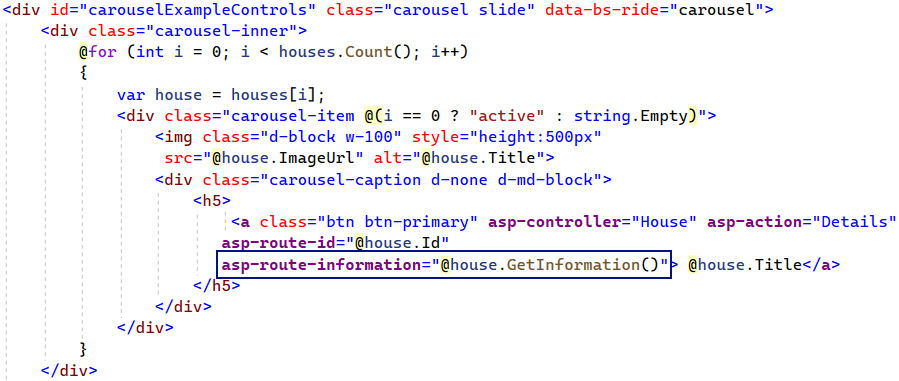
Now you can **modify the views** to **send the information when making requests**. First, go to the "\_ViewImports.cshtml" view and add the ModelExtensions **class** **namespace** to use its method:



Go to the "\_HousePartial.cshtml" **view** and **add the house information** as a **route parameter**:

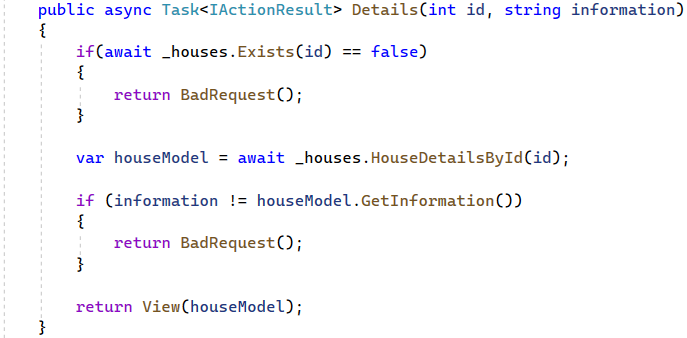


Do the same with the [Details] **button** in the "Index.cshtml" **view**:

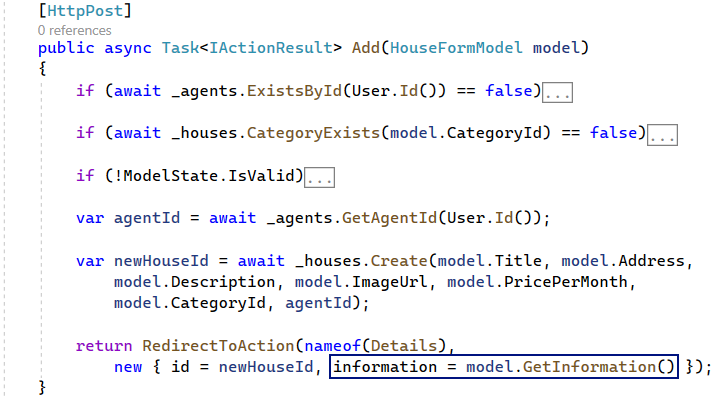


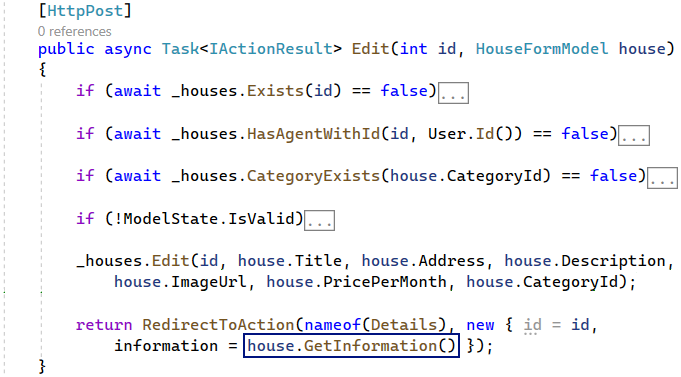
Note that here we are **not protected from unescaped characters** (to do this, we should **decode** and **encode** the **URL**). However, we won't do this here, so you can **do it on your own**, if you want.

Now we should modify the Details(int id) **method** in the HouseController to **accept the information string** and **check if it is correct**. Do it like this:



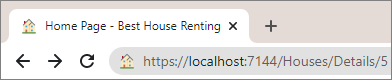
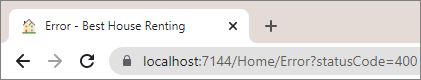
Also, we should **add the information as a parameter** when we **redirect** to the "Details" **page**. Do this in the Add(HouseFormModel model) and Edit(int id, HouseFormModel model) **methods**:



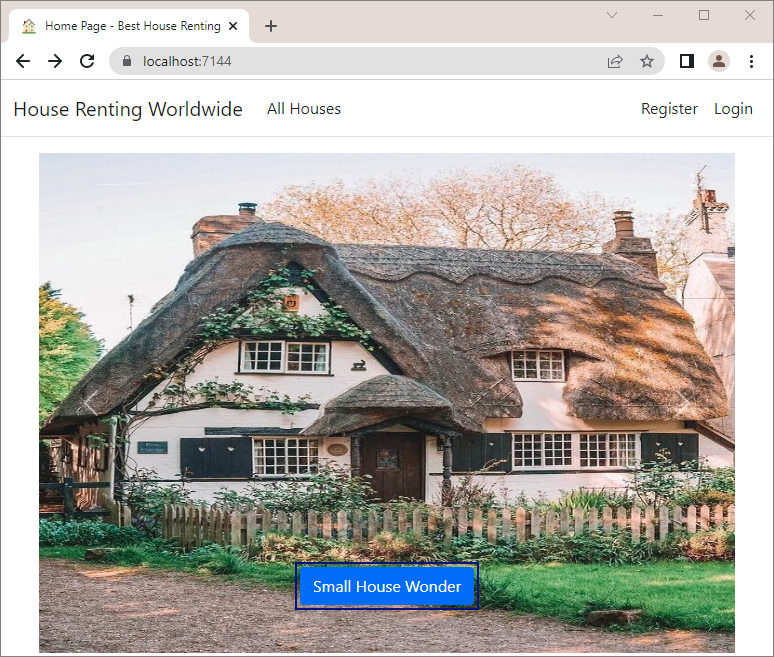


Note that the HouseFormModel should also **implement** the IHouseModel **interface** for this to work.

At last, try out the **URLs in the browser**. If you try to **access** the "Details" **page** with only an **id**, you should see the "400 Bad Request" **error** **page**:

 🡪

However, if you **click** on the [Details] **button** of **any house on any page**, you should see the **house** "Details" **page** and the **URL** should **contain the house information**:

 🡪



You should be correctly **redirected** to the "Details" **page** after **adding or editing a house**. Try it out, as well.

## Inject Services in Views

In this task, we will see how to **inject services in views**, so that we can **use service methods**. We want to do this, so that we can **show different buttons**, depending on whether the **user is an agent or not**, whether the user is **the** **agent** **of the current house**, etc.

To do this, we should first **add the service classes namespaces** to the "\_ViewImports.cshtml" **view**:



### Step 1: Modify the \_Layout.cshtml and \_LoginPartial.cshtml Views

Let's start by modifying the **navigation bar views**. Until now, we showed all buttons, no matter if the current user is an agent or not. We will change that now – when the **user is not an agent**, they should see **all buttons for logged-in** **users**, except for the [Add House] one:



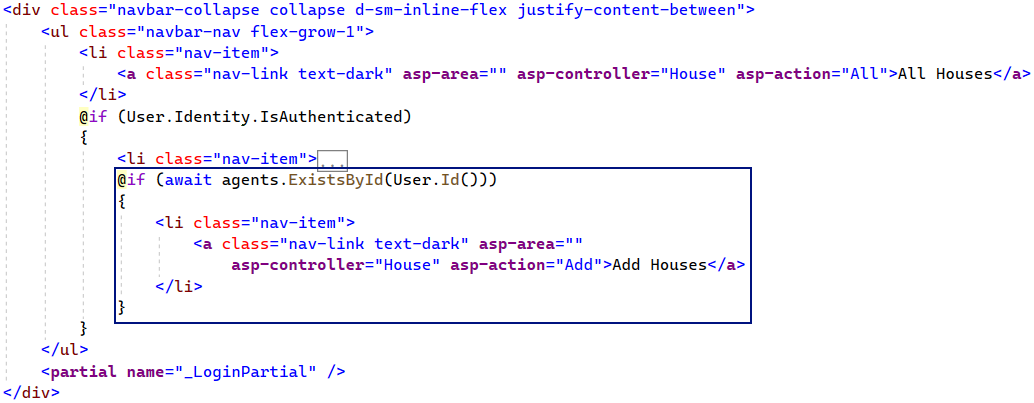
When the **user has become an agent**, they should **not see** the [Become Agent] **button** anymore, but should **see** **the** [Add House] one:



To **modify** when the [Add House] **button is displayed or not**, we should go to the "\_Layout.cshtml" **view** and **inject** **the** IAgentService, as we want to use its ExistsById(string userId) **method**. Do it like this:

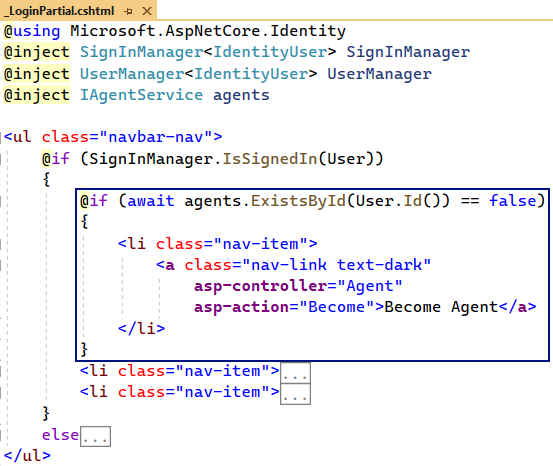


Then, **use the service methods** to check whether the **current user is an agent**. If **they are**, display the [Add House] **button**:



Try out if the **button is visible to agents** **and non-agents** in the browser.

Now let's do the same thing with the [Become Agent] **button** of the **navigation bar**. To modify when it is **displayed**, go to the "\_LoginPartial.cshtml" **view** and **modify** it like this:



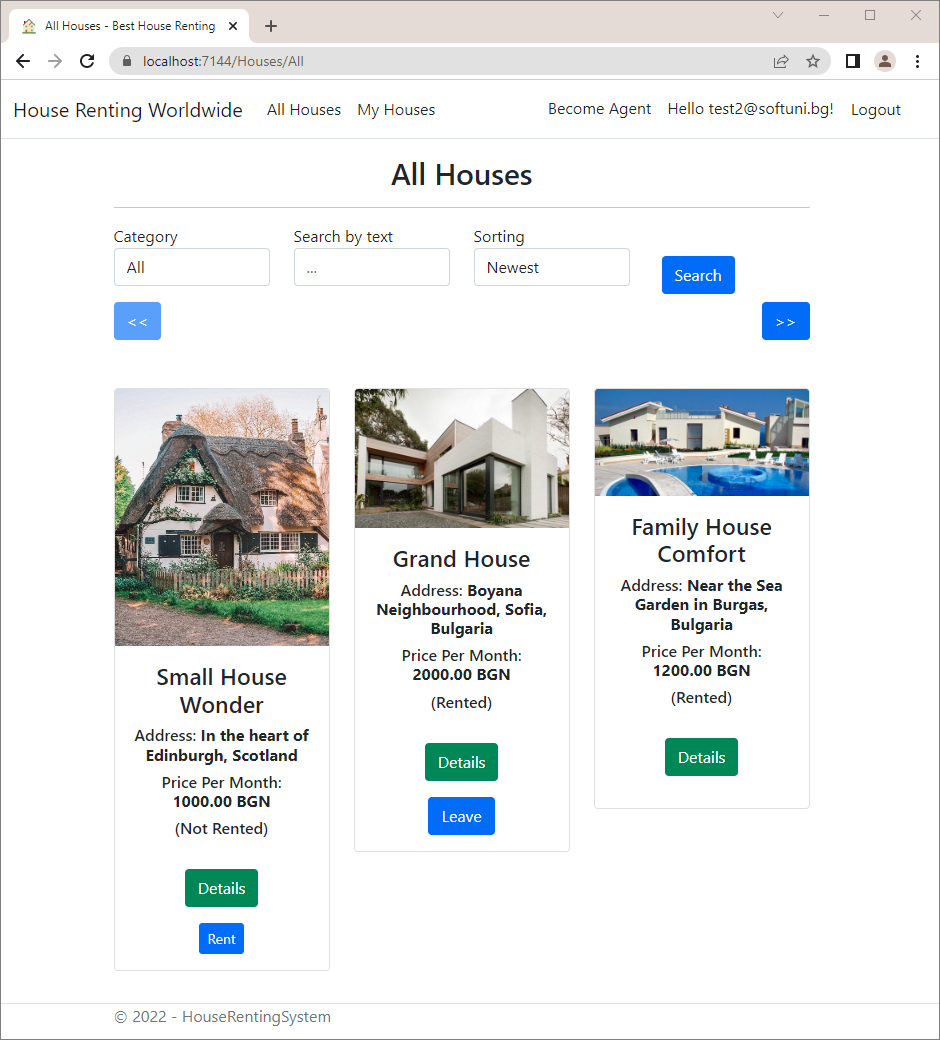
Try out the [Become Agent] **button** in the **browser**, too.

### Step 2: Modify the \_HousePartial.cshtml View

Now we will **modify** the "\_HousePartial.cshtml" **view**, so that the "All Houses" and "My Houses" **pages** show the **correct buttons** in different cases.

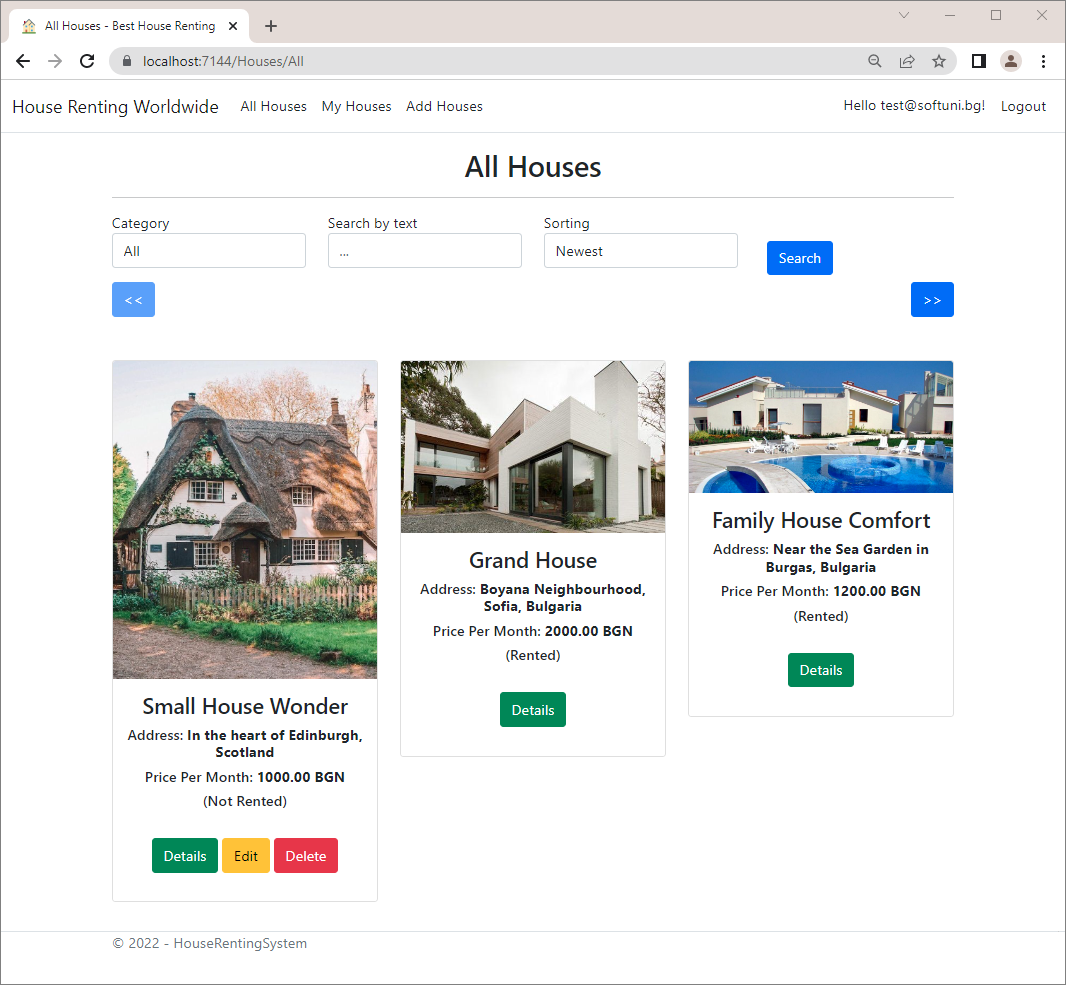
A **non-agent user** should see the [Details] and [Rent] **buttons** of **houses**, which are **not rented**. If they **rent a house**, they should see the [Details] and [Leave] **buttons**. If **another user rented a given house**, the current user should only see the [Details] **button** of the **house**.

Examine the above cases on the page below. The **current user has rented the second house** and **the** **third one is** **rented by another user**. The **first one is not rented**:

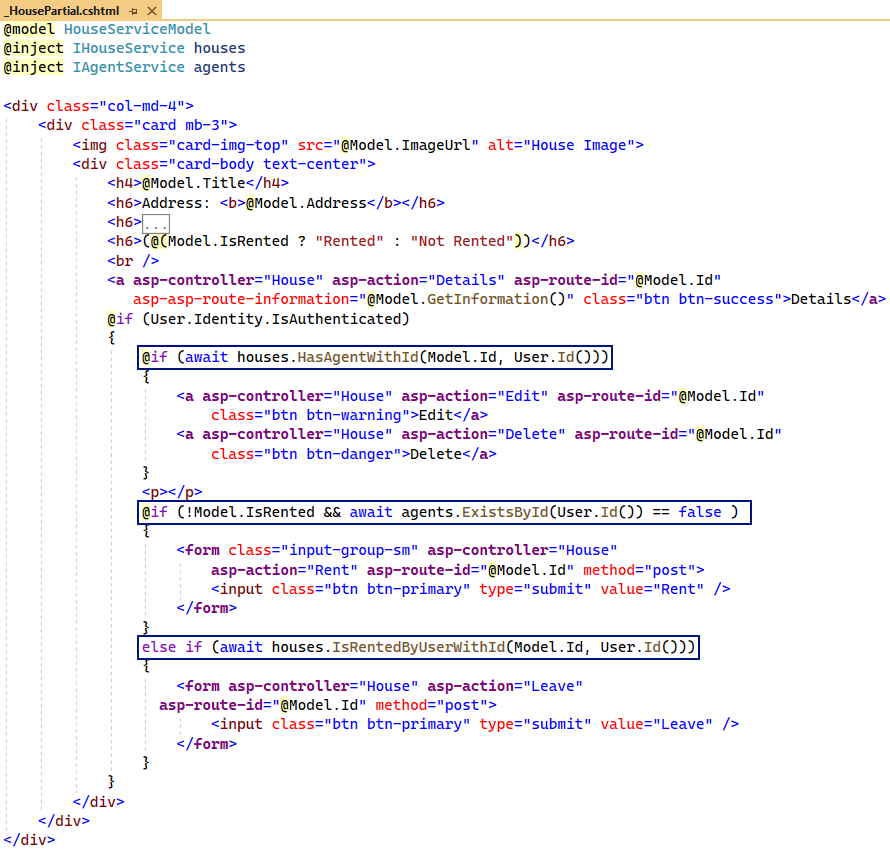


If the **user is an agent**, they should see the [Details], [Edit] and [Delete] **buttons** on **houses they created**. On **other** **agents' houses**, they should see only the [Details] **button**.

In the example page below, the **current user has created only the first house**:



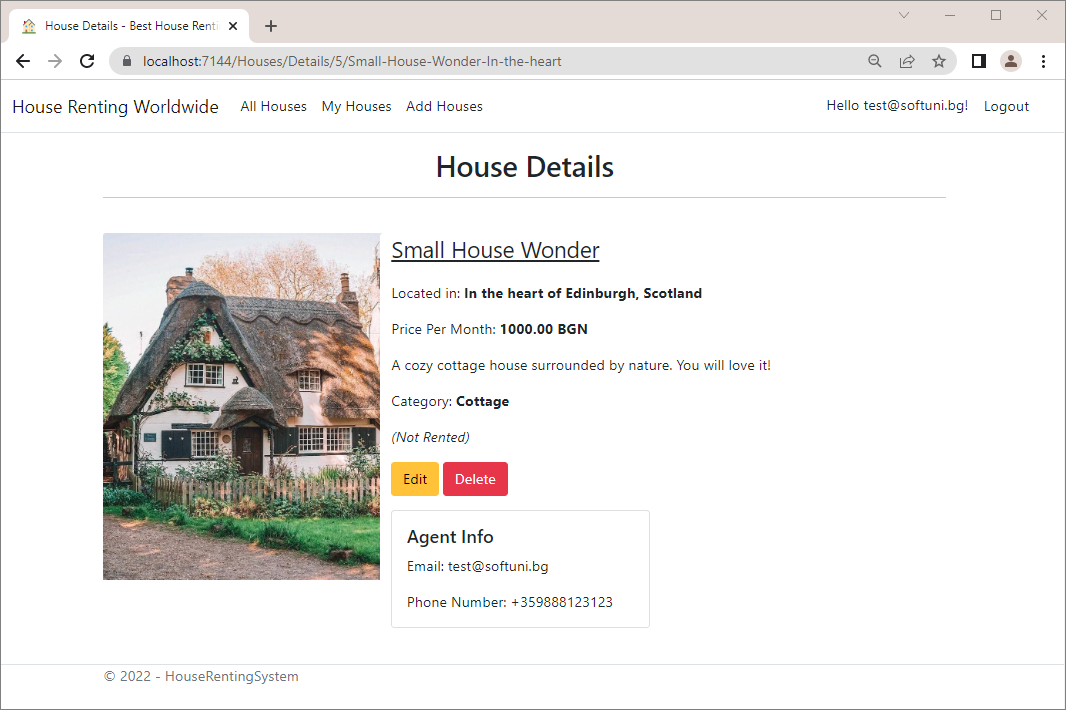
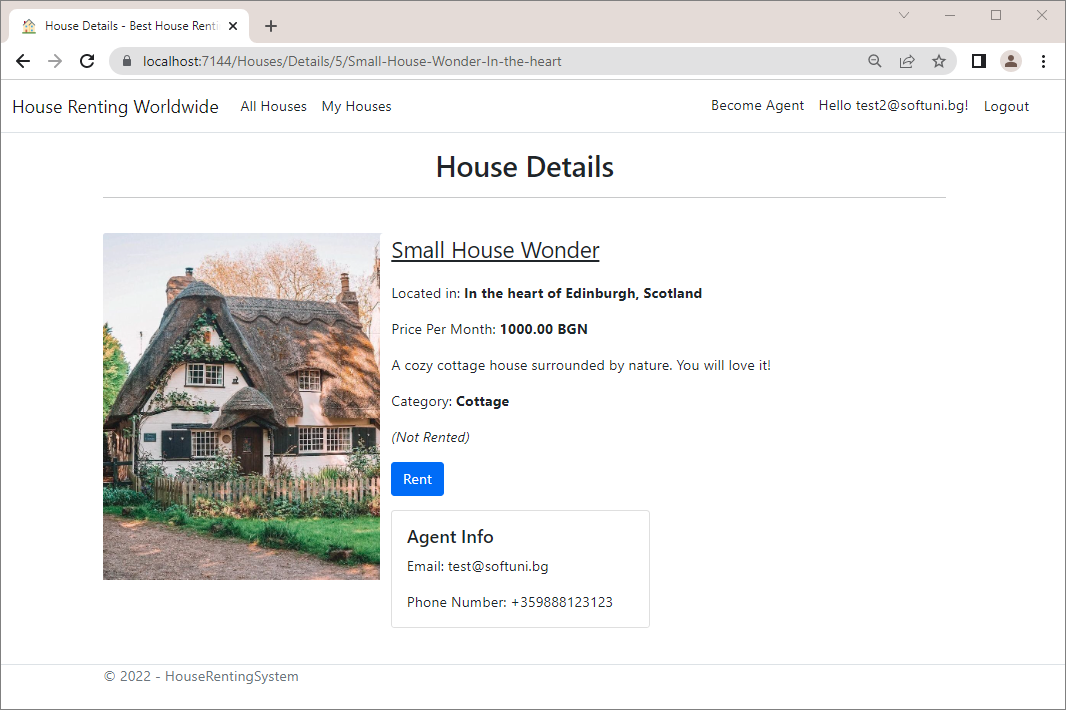
To **change the buttons** on the "All Houses" **page**, go to the "\_HousePartial.cshtml" **view** and **modify** it. **Inject the** IHouseService and the IAgentService, as you will need them. Make the **needed changes to the view**, so that **buttons** are **displayed correctly**. At the end, the **view** should look like this:



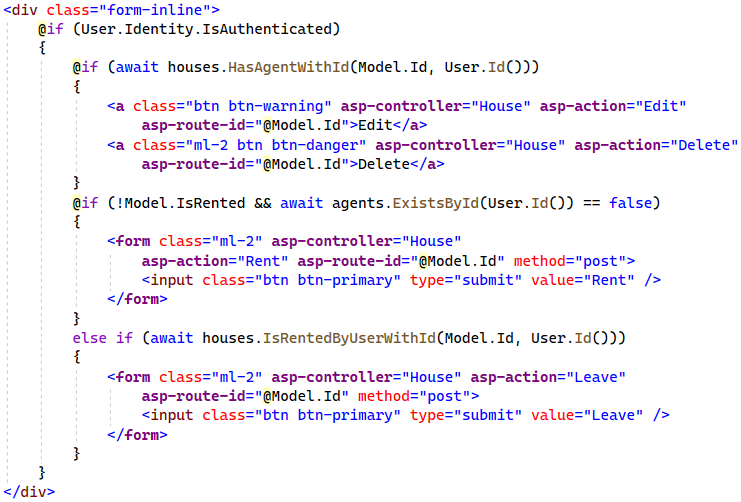
Now look if the **correct buttons are displayed** on the "All Houses" and "My Houses" **pages** in the browser. Make sure the **buttons are displayed as shown on the screenshots above**.

### Step 3: Modify the Details.cshtml View

The Details.cshtml **view** holds the HTML for the "Details" **page**. The page should **display the buttons** for the **current house**, depending on the **user**. It has **the same logic** as of the "All Houses" and "My Houses" **pages**. It should look like this in the **different cases**:



You already know how you should **modify the view**. When done, it should look like this:



### Step 4: Modify the Index.cshtml View

On the last step you should only **restrict** the [Add House] **button** to be **visible only to agents**, when there are **no** **houses** to be displayed on the "Home" **page**.

When the user is **not logged-in** **or not an agent**, the **button should not be visible**:

Graphical user interface, text, application, email

Description automatically generated

When the **user is logged-in and is an agent**, the "Home" **page** should be the following:

Graphical user interface, text, application

Description automatically generated

Do this by **injecting and using a service method**:



**Run the app** and make sure that **all buttons** are **displayed correctly on all pages**.