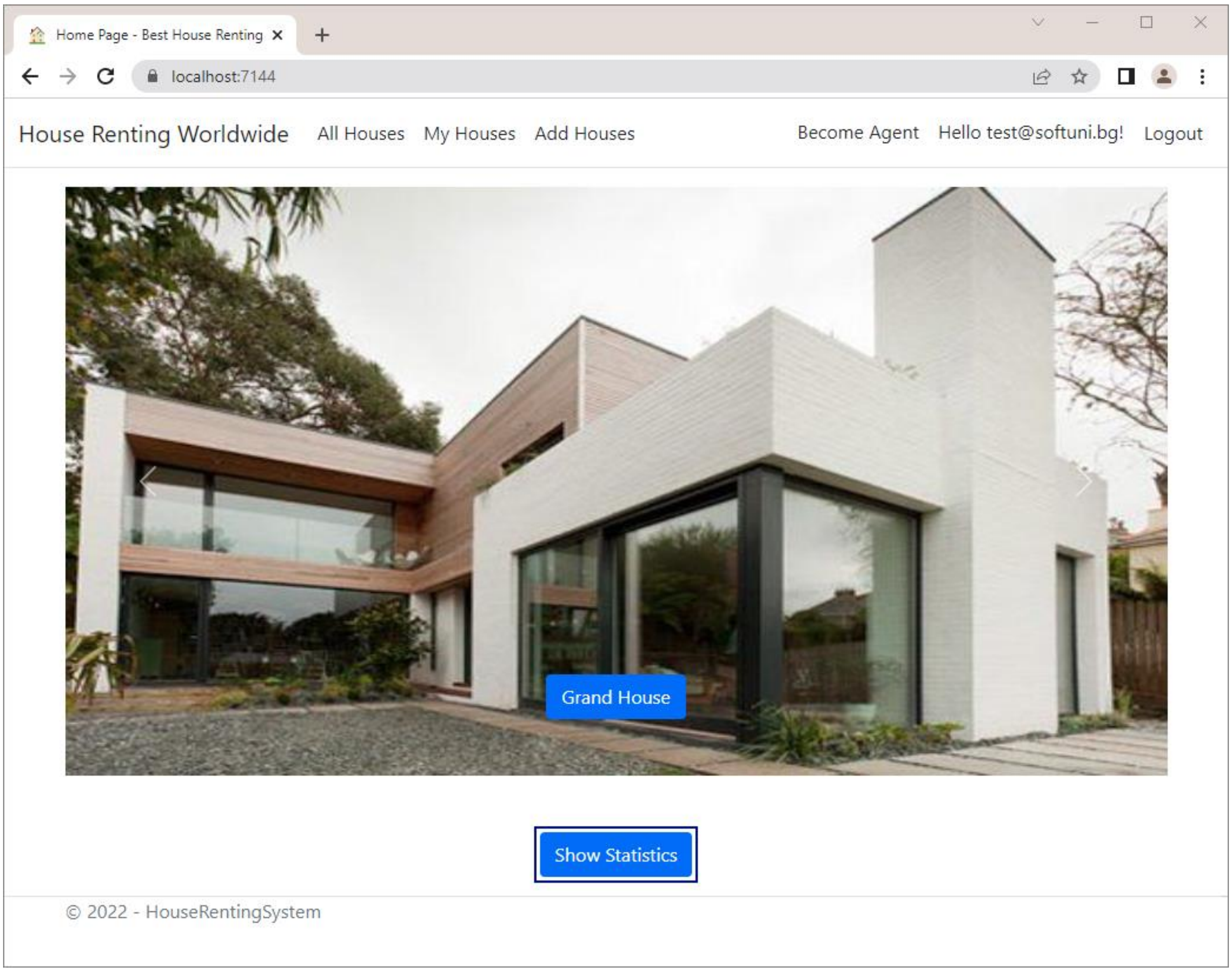
Workshop: Security and Web API

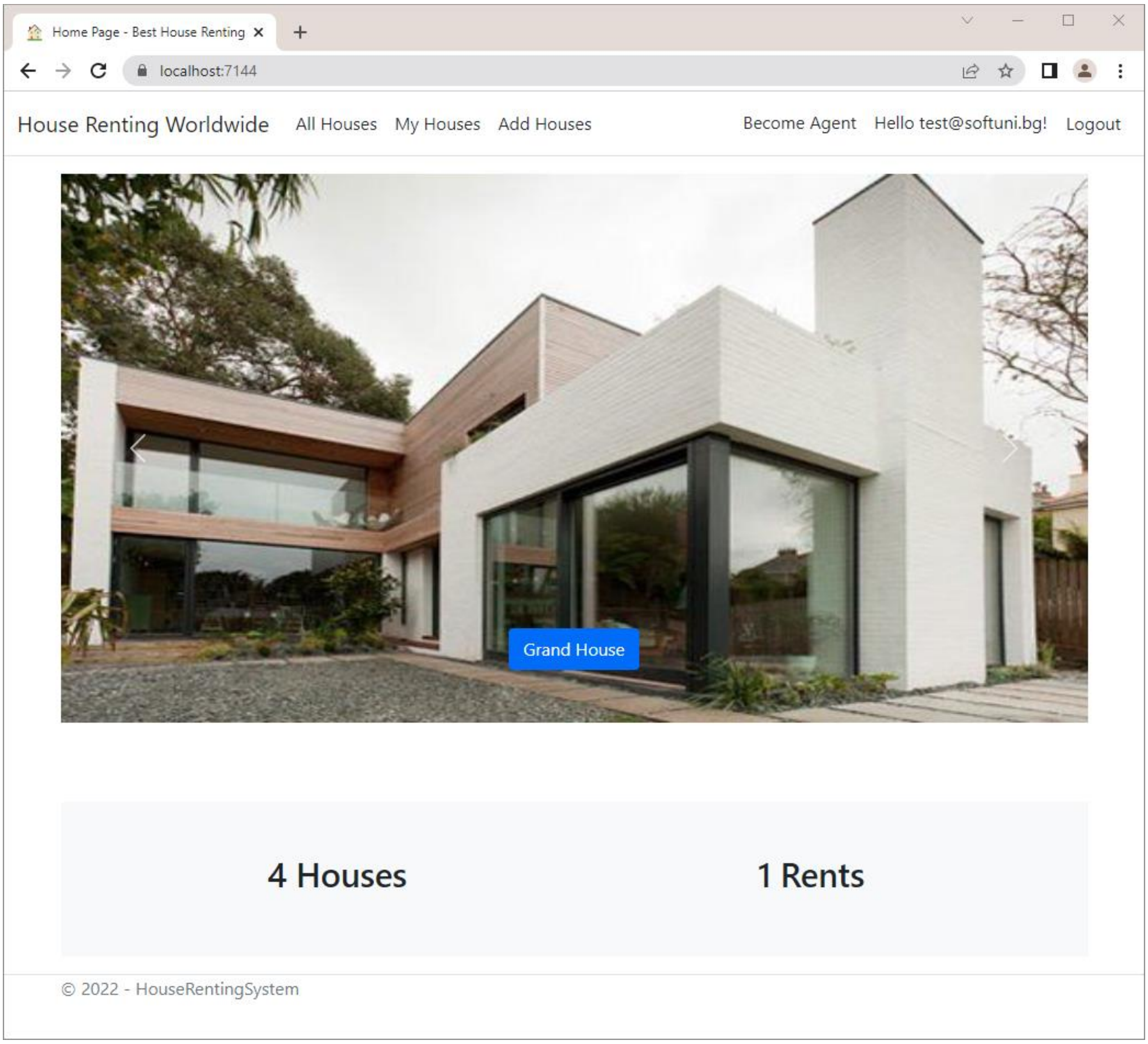
Workshop for the ["ASP.NET Advanced" course @ SoftUni](https://softuni.bg)

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**.

1. 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`**".

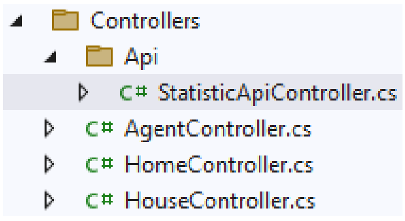
They will have a **single method** to **return a model with total counts** from the database. Create the **`StatisticServiceModel`** in folder "**`/Models/Statistic`**" and **add properties** to it:

```
public class StatisticServiceModel
{
    1 reference
    public int TotalHouses { get; set; }

    1 reference
    public int TotalRents { get; set; }
}
```

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

```
public interface IStatisticService
{
    0 references
    StatisticServiceModel Total();
}

public class StatisticService : IStatisticService
{
    private readonly HouseRentingDbContext _data;

    0 references
    public StatisticService(HouseRentingDbContext data)
    {
        _data = data;
    }

    1 reference
    public StatisticServiceModel Total()
    {
        var totalHouses = _data.Houses.Count();
        var totalRents = _data
            .Houses
            .Where(h => h.RenterId != null)
            .Count();

        return new StatisticServiceModel
        {
            TotalHouses = totalHouses,
            TotalRents = totalRents
        };
    }
}
```

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**:



```
Controllers
  Api
    StatisticApiController.cs
  AgentController.cs
  HomeController.cs
  HouseController.cs
```

```
public class StatisticApiController : Controller
{
}
```

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

```
[ApiController]
[Route("api/statistic")]
0 references
public class StatisticApiController : ControllerBase
{
}
```

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:

```
[ApiController]
[Route("api/statistic")]
1 reference
public class StatisticApiController : ControllerBase
{
    private readonly IStatisticService _statistics;

    0 references
    public StatisticApiController(IStatisticService statistics)
    {
        _statistics = statistics;
    }

    [HttpGet]
    0 references
    public StatisticServiceModel GetStatistic()
    {
        return _statistics.Total();
    }
}
```

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

```
builder.Services.AddTransient<IStatisticService, StatisticService>();
```

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:

```
Index.cshtml
<div class="mb-5"></div>

<div class="row">
    <div class="col-12 text-center">
        <button class="btn btn-primary" id="statistics-button">Show Statistics</button>
    </div>
</div>

<div class="mt-4 p-5 bg-light d-none" id="statistics">
    <div class="row">
        <h2 class="col-md-6 text-center" id="total-houses"></h2>
        <h2 class="col-md-6 text-center" id="total-rents"></h2>
    </div>
</div>

@section Scripts {
    <script>
        $('#statistics-button').on('click', ev => {
            $.get('/api/statistics', (data) => {
                $('#total-houses').text(data.totalHouses + " Houses");
                $('#total-rents').text(data.totalRents + " Rents");
                $('#statistics').removeClass('d-none');
                $('#statistics-button').hide();
            });
        });
    </script>
}
```

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.

2. 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**.

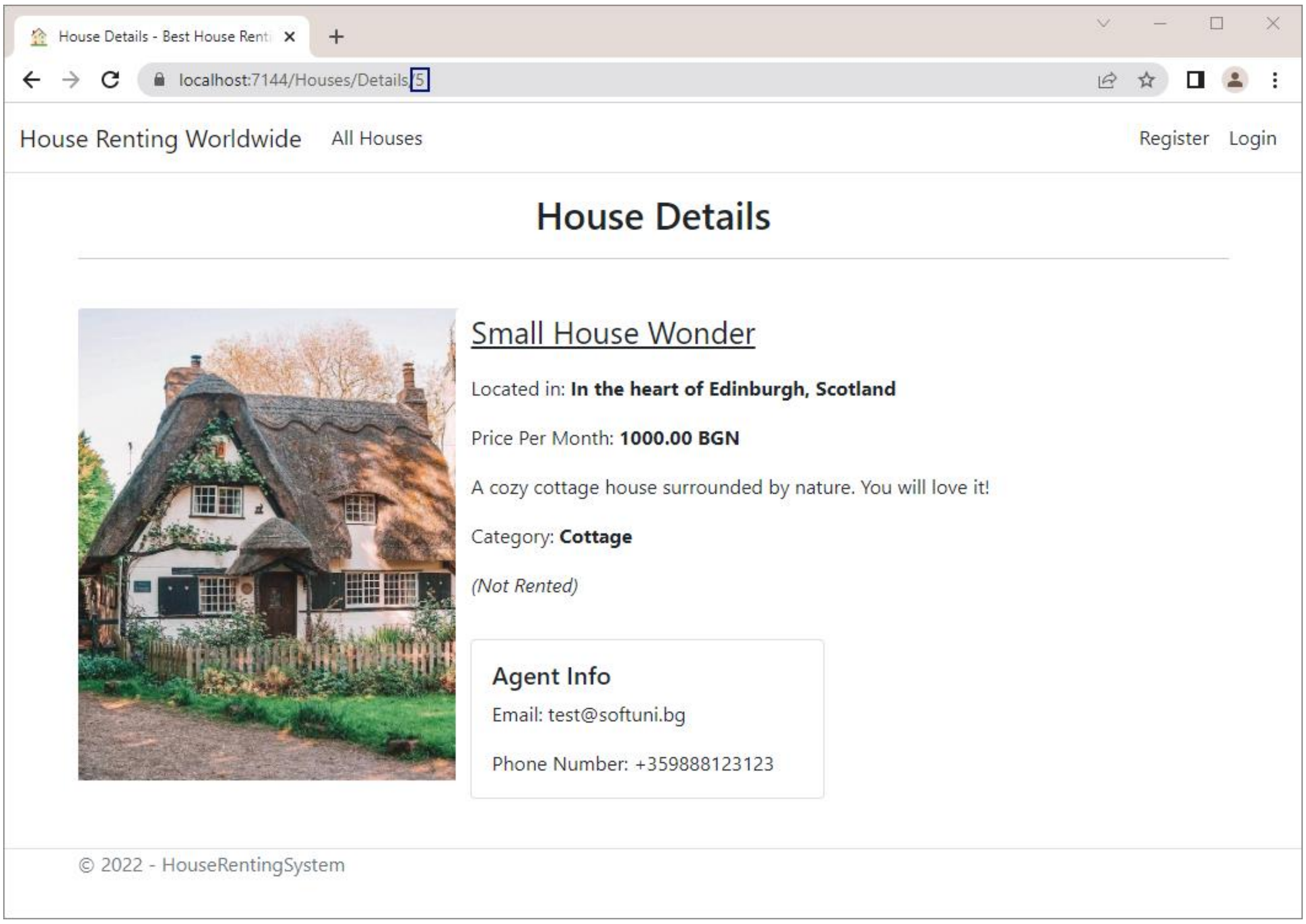
Go to the **`Program.cs`** of your app and **add the filter** to the **`AddControllersWithViews(…)` method** like this:

```
builder.Services.AddControllersWithViews(options =>
{
    options.Filters.Add<AutoValidateAntiforgeryTokenAttribute>();
});
```

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

3. 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**:


House Details - Best House Renti
localhost:7144/Houses/Details/5
House Renting Worldwide
All Houses
Register
Login

House Details

Small House Wonder

Located in: **In the heart of Edinburgh, Scotland**

Price Per Month: **1000.00 BGN**

A cozy cottage house surrounded by nature. You will love it!

Category: **Cottage**

(Not Rented)

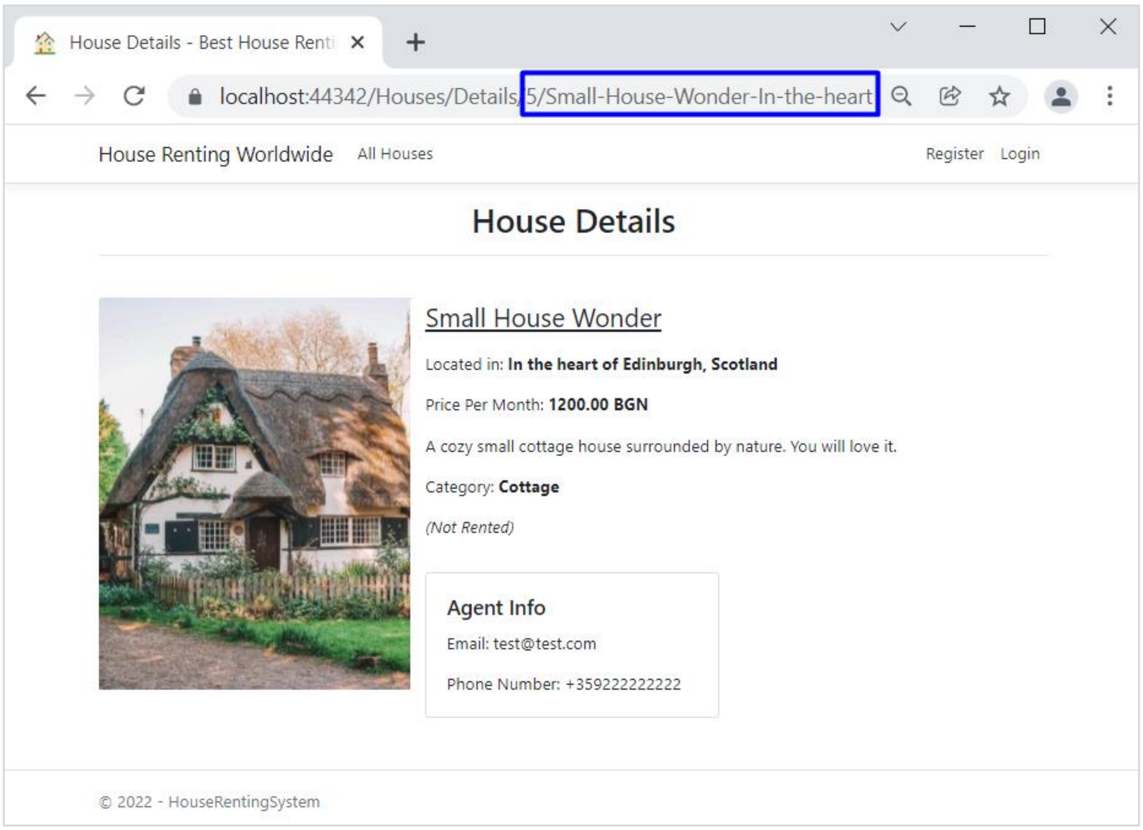
Agent Info

Email: test@softuni.bg

Phone Number: +359888123123

© 2022 - HouseRentingSystem

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:


House Details - Best House Renti
localhost:44342/Houses/Details/5/Small-House-Wonder-In-the-heart
House Renting Worldwide
All Houses
Register
Login

House Details

Small House Wonder

Located in: **In the heart of Edinburgh, Scotland**

Price Per Month: **1200.00 BGN**

A cozy small cottage house surrounded by nature. You will love it.

Category: **Cottage**

(Not Rented)

Agent Info

Email: test@test.com

Phone Number: +359222222222

© 2022 - HouseRentingSystem

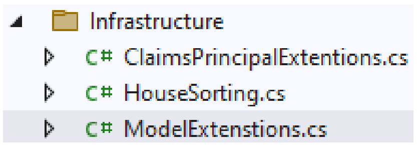
To do this, we should **modify our routes** to **include the information in the URL** and the "**`Details`**" **page** for a house to be accessed on "**`/House/Details/{id}/{information}`**". When we click on a **`[Details]` button**, we will **include this information**. Then, when we access the "**`Details`**" **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:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "House Details",
        pattern: "/House/Details/{id}/{information}",
        defaults: new { Controller = "House", Action = "Details" });

    endpoints.MapDefaultControllerRoute();
    endpoints.MapRazorPages();
});
```

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`**":



```
public static class ModelExtenstions
{
}
```

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**:

```
public interface IHouseModel
{
    0 references
    public string Title { get; }

    0 references
    public string Address { get; }
}
```

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:

```
public static class ModelExtenstions
{
    0 references
    public static string GetInformation(this IHouseModel house)
    {
        return house.Title.Replace(" ", "-") + "-" + GetAddress(house.Address);
    }

    1 reference
    private static string GetAddress(string address)
    {
        address = String.Join("-", address.Split(" ").Take(3));
        return Regex.Replace(address, @"[^a-zA-Z0-9\-]", string.Empty);
    }
}
```

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**:

```
public class HouseServiceModel : IHouseModel

public class HouseIndexServiceModel : IHouseModel
{
```

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**:

```
public async Task<IEnumerable<HouseIndexServiceModel>> LastThreeHouses()
{
    return _data
            .Houses
            .OrderByDescending(c => c.Id)
            .Select(c => new HouseIndexServiceModel
            {
                Id = c.Id,
                Title = c.Title,
                ImageUrl = c.ImageUrl,
                Address = c.Address
            })
            .Take(3);
}
```

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:

```
@using HouseRentingSystem.Infrastructure;
```

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

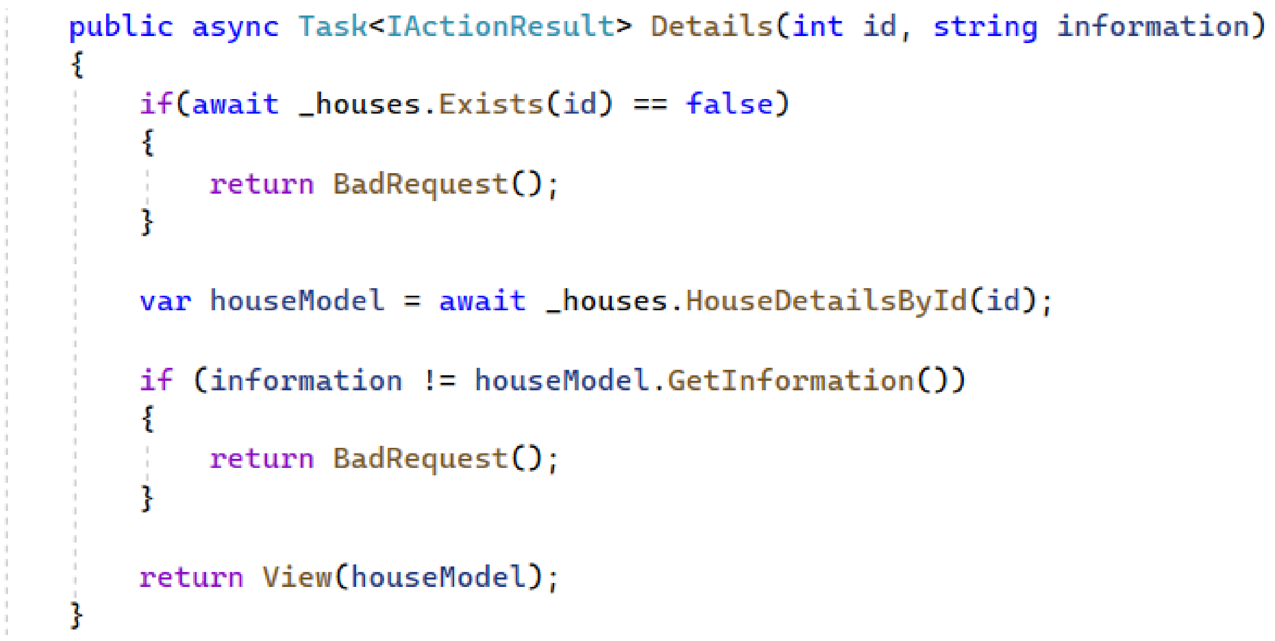
```
<div class="col-md-4">
    <div class="card mb-3">
        <img class="card-img-top" src="@Model.ImageUrl" alt="House Image">
        <div class="card-body text-center">
            <h4>@Model.Title</h4>
            <h6>Address: <b>@Model.Address</b></h6>
            <h6>
                Price Per Month:
                <b>@String.Format("{0:f2}", Model.PricePerMonth) BGN</b>
            </h6>
            <h6>(@(Model.IsRented ? "Rented" : "Not Rented"))</h6>
            <br />
            <a asp-controller="House" asp-action="Details" asp-route-id="@Model.Id"
               asp-asp-route-information="@Model.GetInformation()" class="btn btn-success">Details</a>
```

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

```
<div id="carouselExampleControls" class="carousel slide" data-bs-ride="carousel">
    <div class="carousel-inner">
        @for (int i = 0; i < houses.Count(); i++)
        {
            var house = houses[i];
            <div class="carousel-item @(i == 0 ? "active" : string.Empty)">
                <img class="d-block w-100" style="height:500px"
                 src="@house.ImageUrl" alt="@house.Title">
                <div class="carousel-caption d-none d-md-block">
                    <h5>
                        <a class="btn btn-primary" asp-controller="House" asp-action="Details"
                       asp-route-id="@house.Id"
                       asp-route-information="@house.GetInformation()"> @house.Title</a>
                    </h5>
                </div>
            </div>
        }
    </div>
```

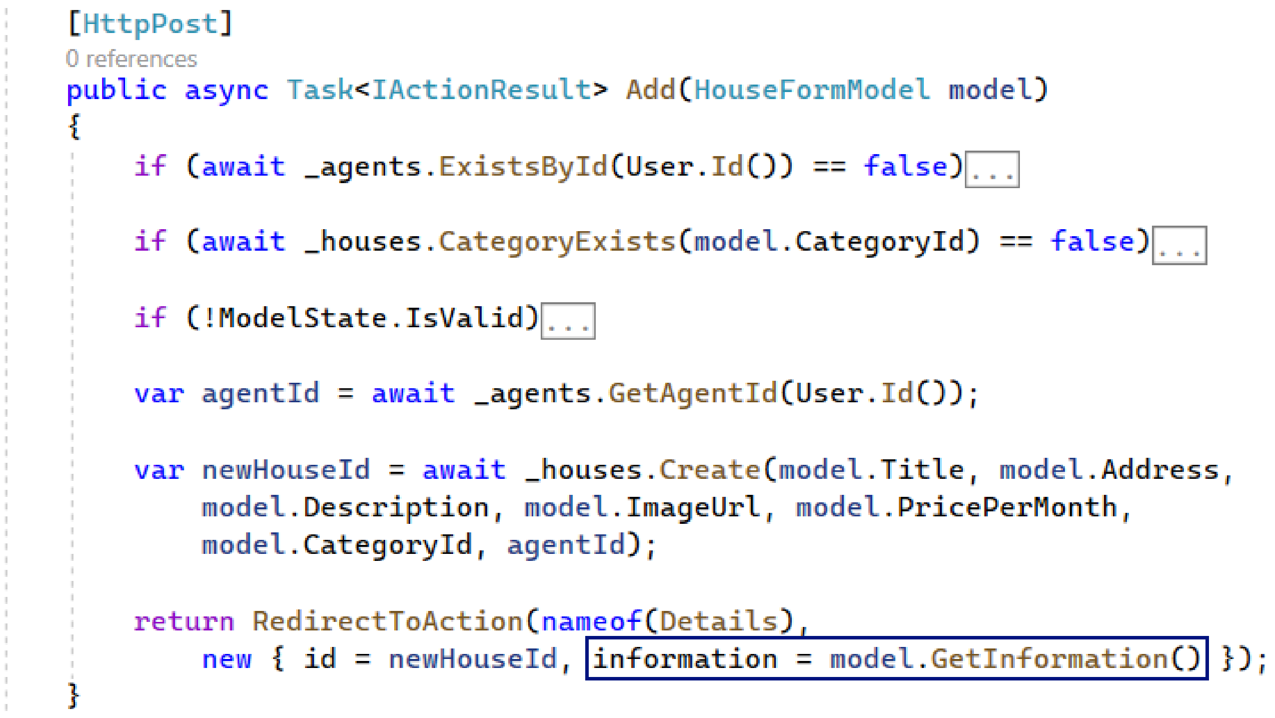
Note that here we are **not protected from unescaped characters** (to do this, we should **decode** and **encode** the **URL**). However, we won't do this here, so you can **do it on your own**, if you want.

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:



```
public async Task<IActionResult> Details(int id, string information)
{
    if(await _houses.Exists(id) == false)
    {
        return BadRequest();
    }

    var houseModel = await _houses.HouseDetailsById(id);

    if (information != houseModel.GetInformation())
    {
        return BadRequest();
    }

    return View(houseModel);
}
```

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**:

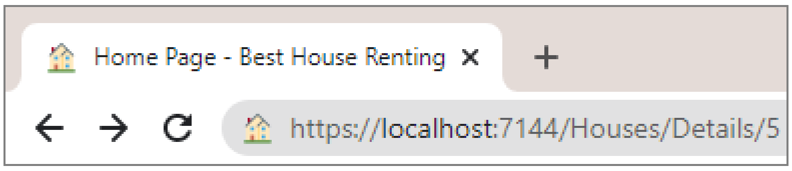


```
[HttpPost]
0 references
public async Task<IActionResult> Add(HouseFormModel model)
{
    if (await _agents.ExistsById(User.Id()) == false)...

    if (await _houses.CategoryExists(model.CategoryId) == false)...

    if (!ModelState.IsValid)...

    var agentId = await _agents.GetAgentId(User.Id());

    var newHouseId = await _houses.Create(model.Title, model.Address,
        model.Description, model.ImageUrl, model.PricePerMonth,
        model.CategoryId, agentId);

    return RedirectToAction(nameof(Details),
        new { id = newHouseId, information = model.GetInformation() });
}
```

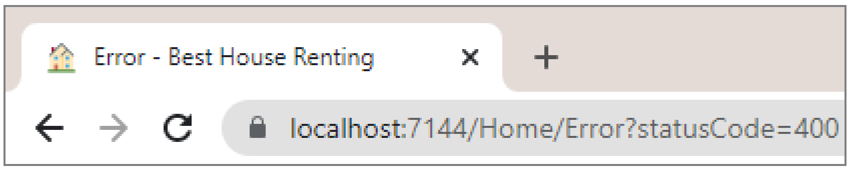
```
[HttpPost]
0 references
public async Task<IActionResult> Edit(int id, HouseFormModel house)
{
    if (await _houses.Exists(id) == false)...

    if (await _houses.HasAgentWithId(id, User.Id()) == false)...

    if (await _houses.CategoryExists(house.CategoryId) == false)...

    if (!ModelState.IsValid)...

    _houses.Edit(id, house.Title, house.Address, house.Description,
        house.ImageUrl, house.PricePerMonth, house.CategoryId);

    return RedirectToAction(nameof(Details), new { id = id,
        information = house.GetInformation() });
}
```

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

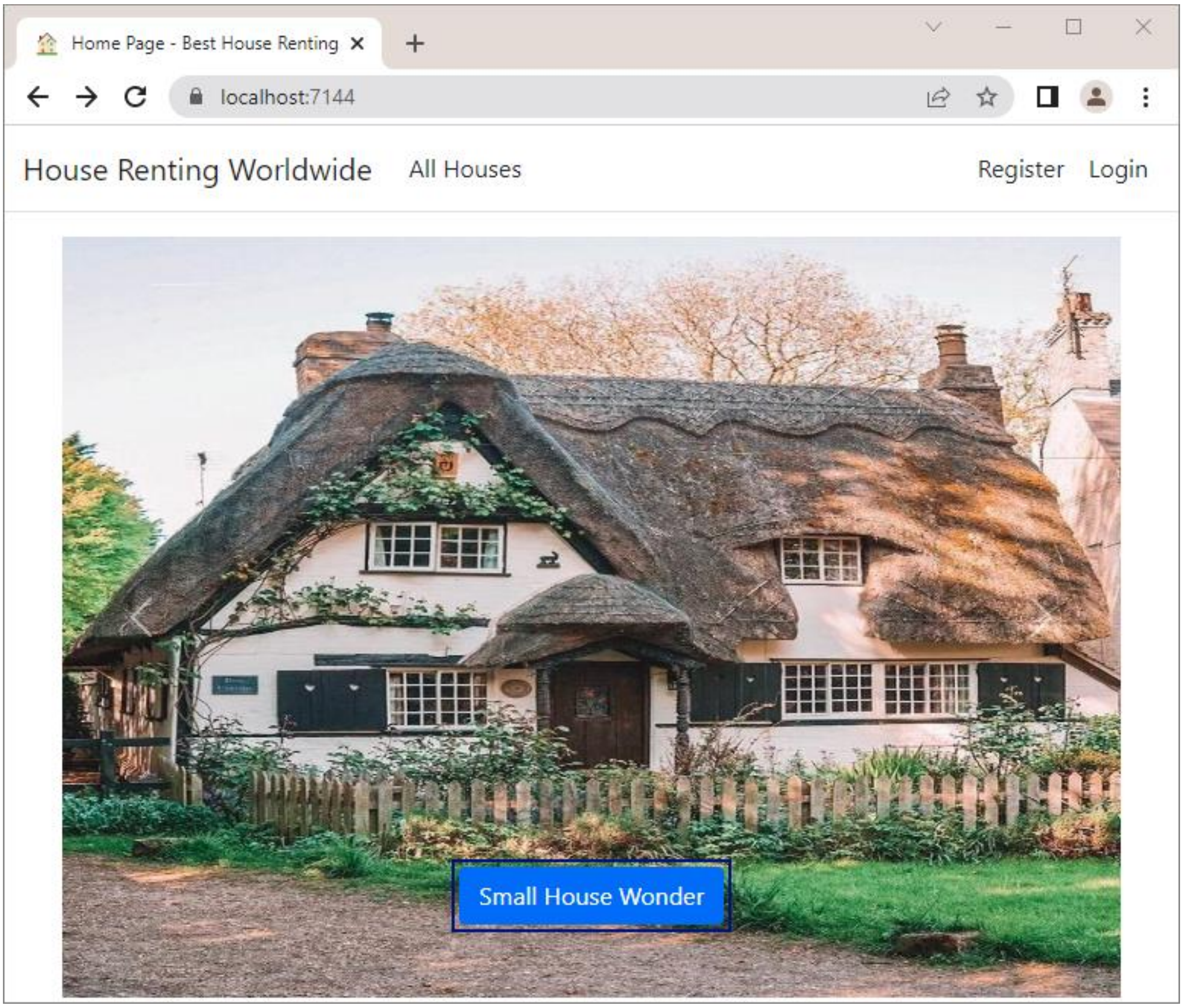
At last, try out the **URLs in the browser**. If you try to **access** the "**`Details`**" **page** with only an **id**, you should see the "**`400 Bad Request`**" **error page**:


Home Page - Best House Renting
https://localhost:7144/Houses/Details/5

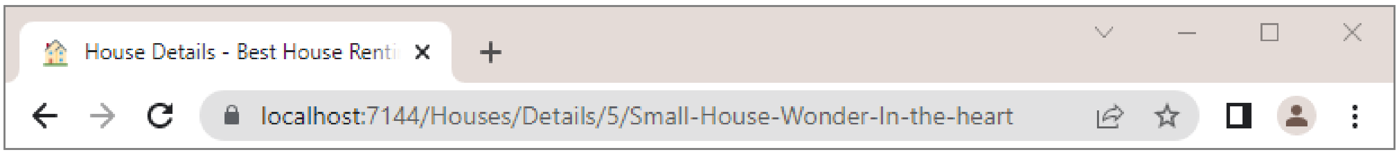
→


Error - Best House Renting
localhost:7144/Home/Error?statusCode=400

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**:



→


House Details - Best House Renti
localhost:7144/Houses/Details/5/Small-House-Wonder-In-the-heart

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

4. Inject Services in Views

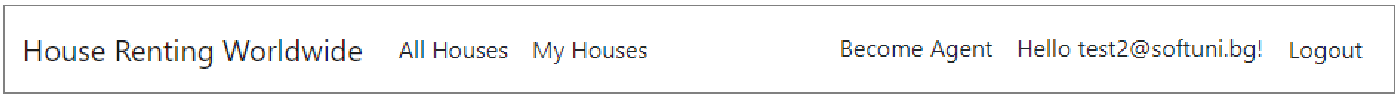
In this task, we will see how to **inject services in views**, so that we can **use service methods**. We want to do this, so that we can **show different buttons**, depending on whether the **user is an agent or not**, whether the user is **the agent of the current house**, etc.

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

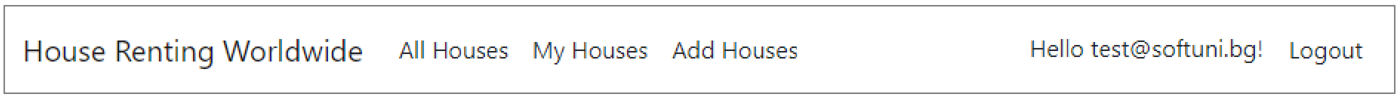
```
@using HouseRentingSystem.Services.Agent;
@using HouseRentingSystem.Contracts.Agent;
```

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

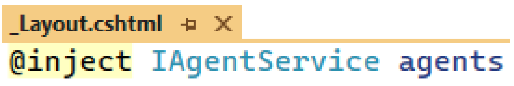
Let's start by modifying the **navigation bar views**. Until now, we showed all buttons, no matter if the current user is an agent or not. We will change that now – when the **user is not an agent**, they should see **all buttons for logged-in users**, except for the **[Add House]** one:



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

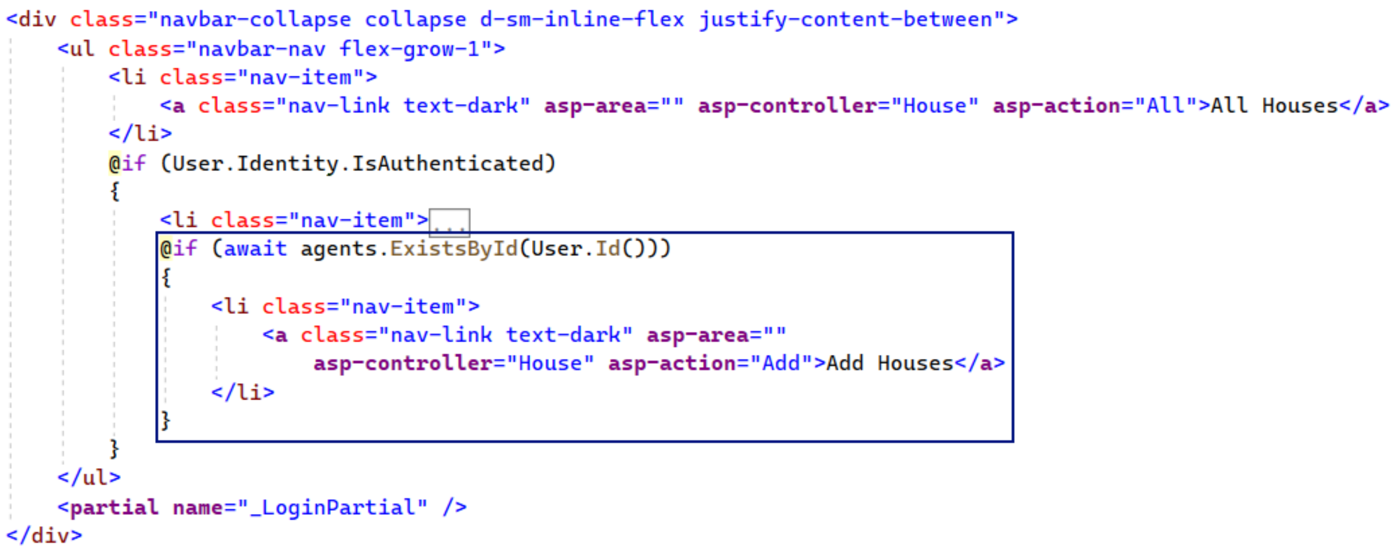


To **modify** when the **[Add House] button is displayed or not**, we should go to the "**_Layout.cshtml" view** and **inject the IAgentService**, as we want to use its **ExistsById(string userId) method**. Do it like this:



```
@inject IAgentService agents
```

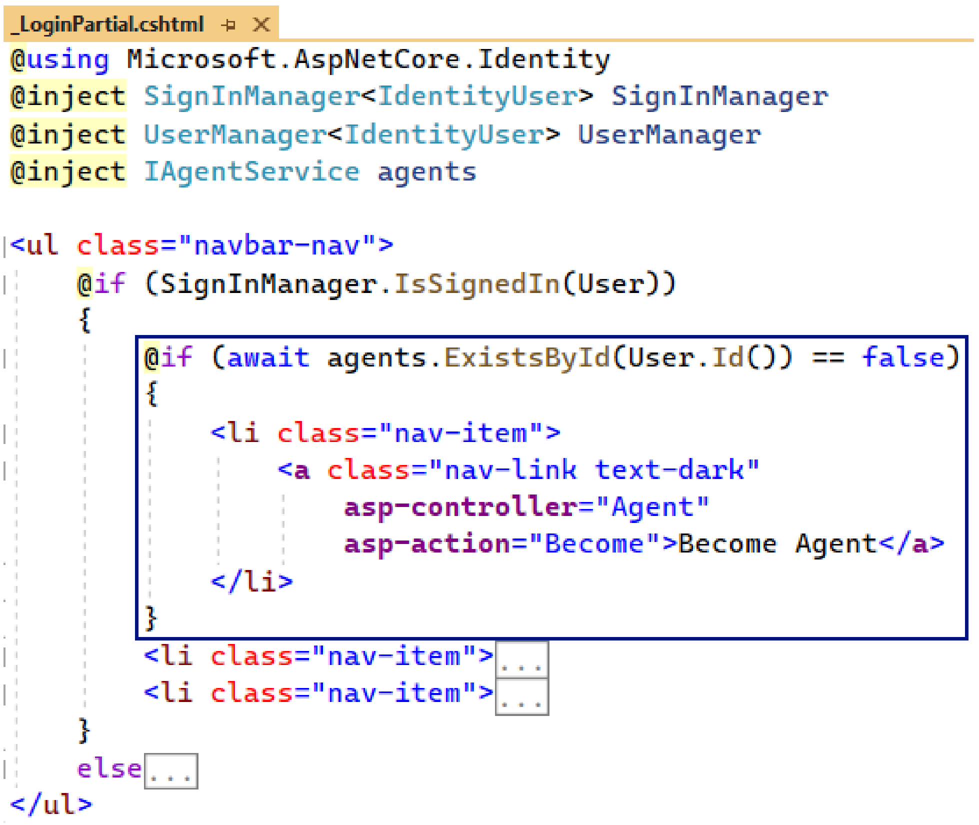
Then, **use the service methods** to check whether the **current user is an agent**. If **they are**, display the **[Add House] button**:



```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="House" asp-action="All">All Houses</a>
        </li>
        @if (User.Identity.IsAuthenticated)
        {
            <li class="nav-item">...
            @if (await agents.ExistsById(User.Id()))
            {
                <li class="nav-item">
                    <a class="nav-link text-dark" asp-area=""
                        asp-controller="House" asp-action="Add">Add Houses</a>
                </li>
            }
        }
    </ul>
    <partial name="_LoginPartial" />
</div>
```

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:



```
_LoginPartial.cshtml
@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
@inject IAgentService agents

<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        @if (await agents.ExistsById(User.Id()) == false)
        {
            <li class="nav-item">
                <a class="nav-link text-dark"
                    asp-controller="Agent"
                    asp-action="Become">Become Agent</a>
            </li>
        }
        <li class="nav-item">...
        <li class="nav-item">...
    }
    else...
</ul>
```

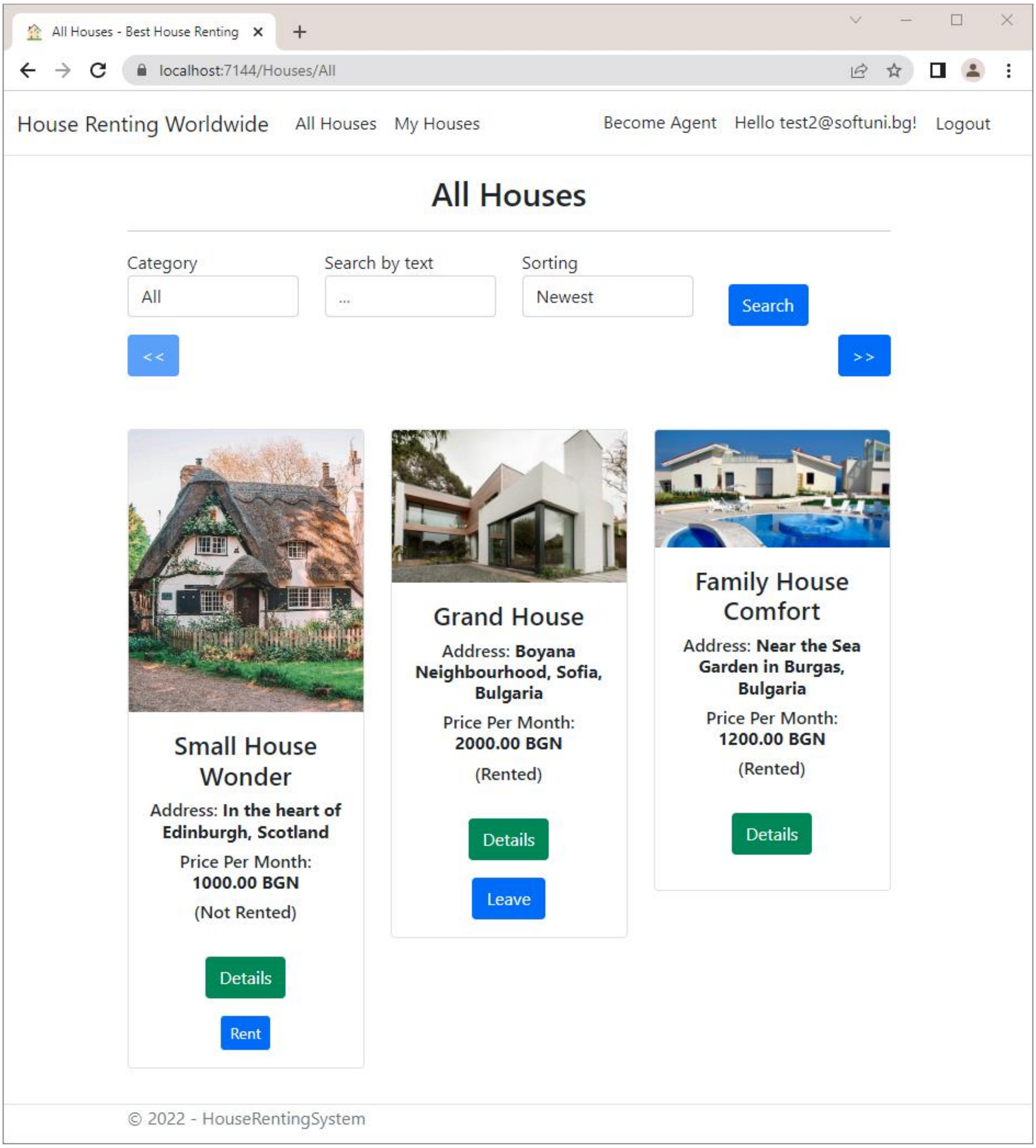
Try out the **[Become Agent] button** in the **browser**, too.

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**:


All Houses - Best House Renting

localhost:7144/Houses/All

House Renting Worldwide All Houses My Houses Become Agent Hello test2@softuni.bg! Logout

All Houses

Category: All

Search by text: ...

Sorting: Newest

Search

<< >>

Small House Wonder
Address: **In the heart of Edinburgh, Scotland**
Price Per Month: **1000.00 BGN**
(Not Rented)
Details
Rent

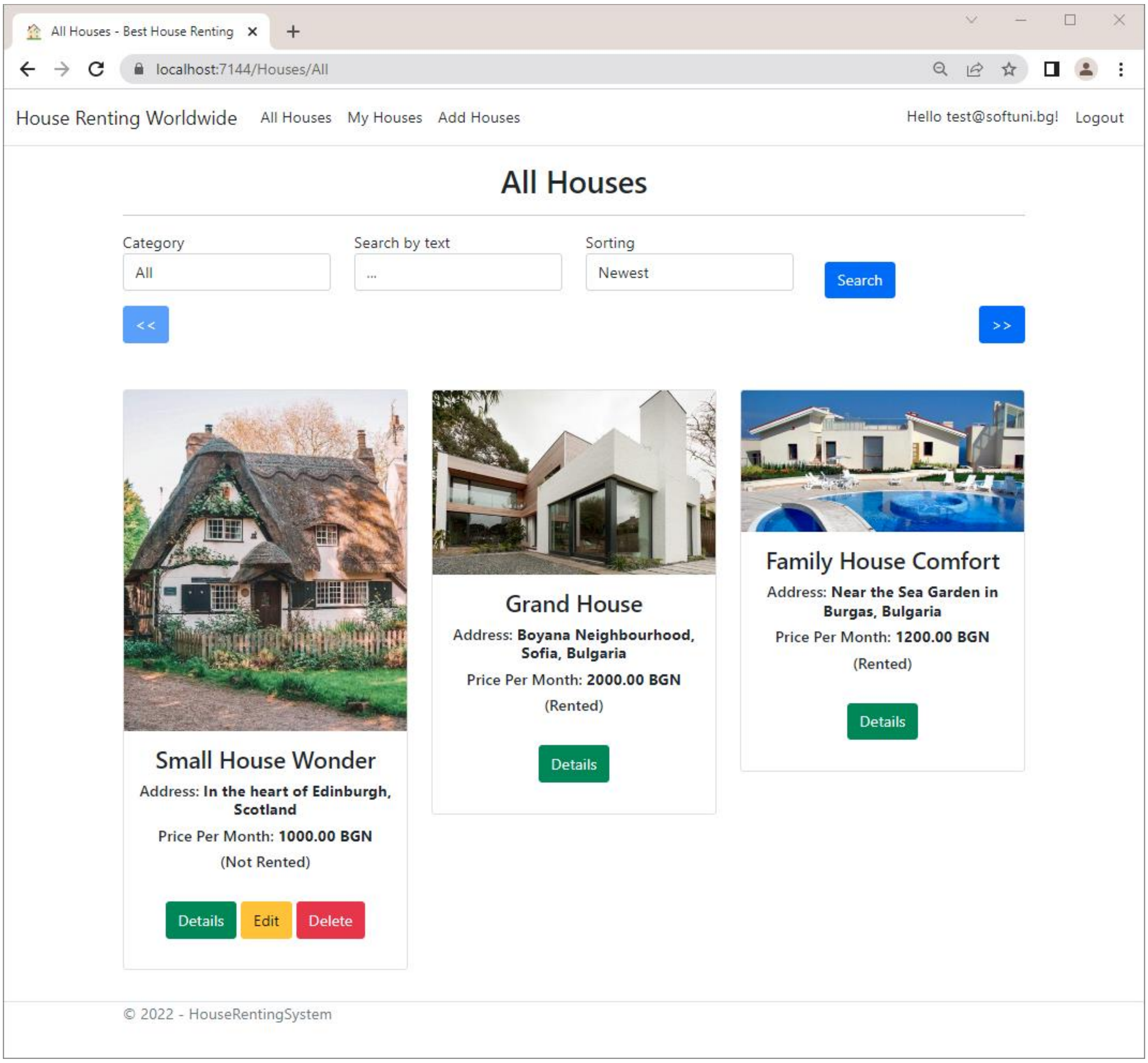
Grand House
Address: **Boyana Neighbourhood, Sofia, Bulgaria**
Price Per Month: **2000.00 BGN**
(Rented)
Details
Leave

Family House Comfort
Address: **Near the Sea Garden in Burgas, Bulgaria**
Price Per Month: **1200.00 BGN**
(Rented)
Details

© 2022 - HouseRentingSystem

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**:



All Houses

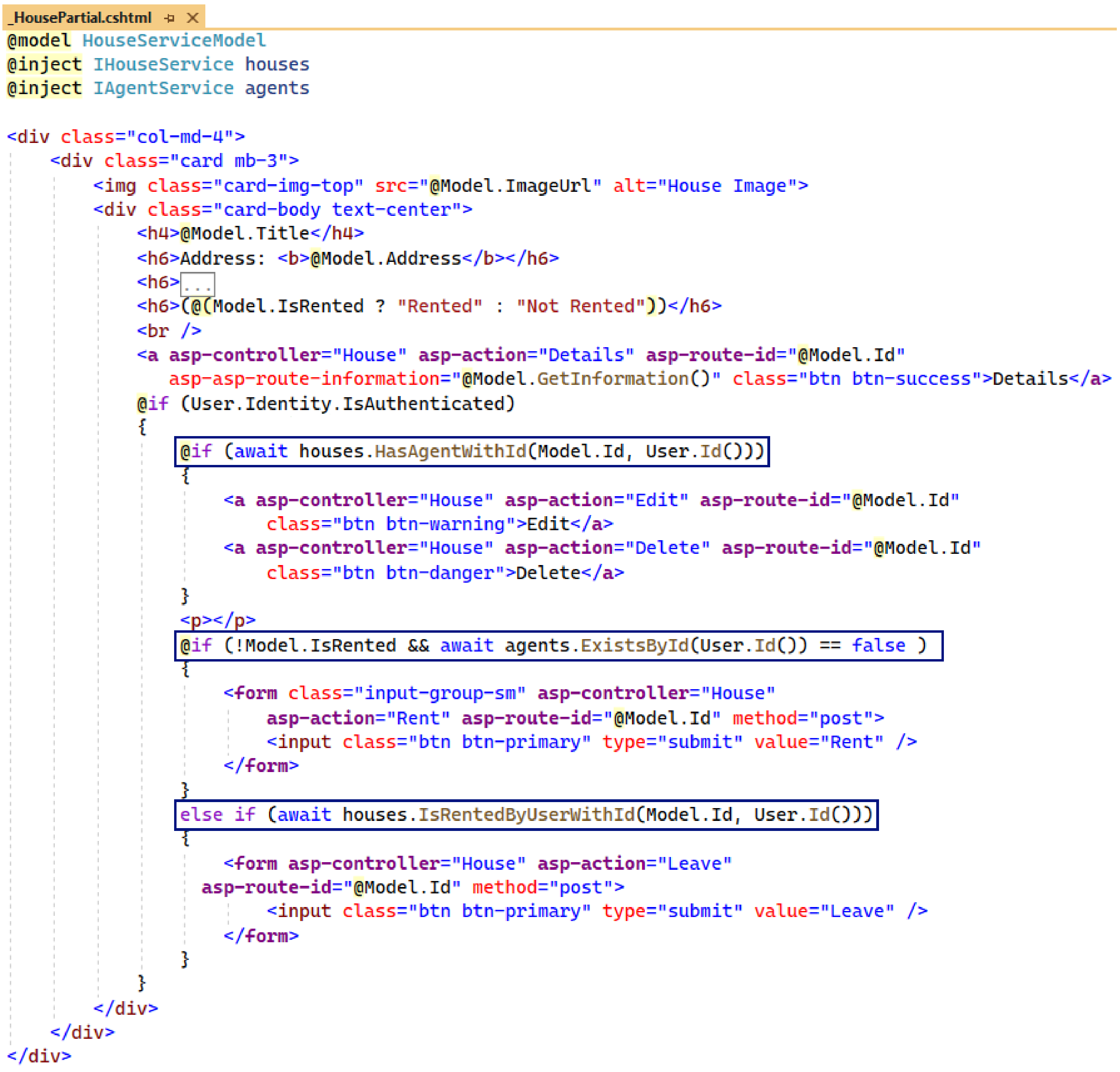
Small House Wonder
Address: **In the heart of Edinburgh, Scotland**
Price Per Month: **1000.00 BGN**
(Not Rented)

Grand House
Address: **Boyana Neighbourhood, Sofia, Bulgaria**
Price Per Month: **2000.00 BGN**
(Rented)

Family House Comfort
Address: **Near the Sea Garden in Burgas, Bulgaria**
Price Per Month: **1200.00 BGN**
(Rented)

© 2022 - HouseRentingSystem

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:

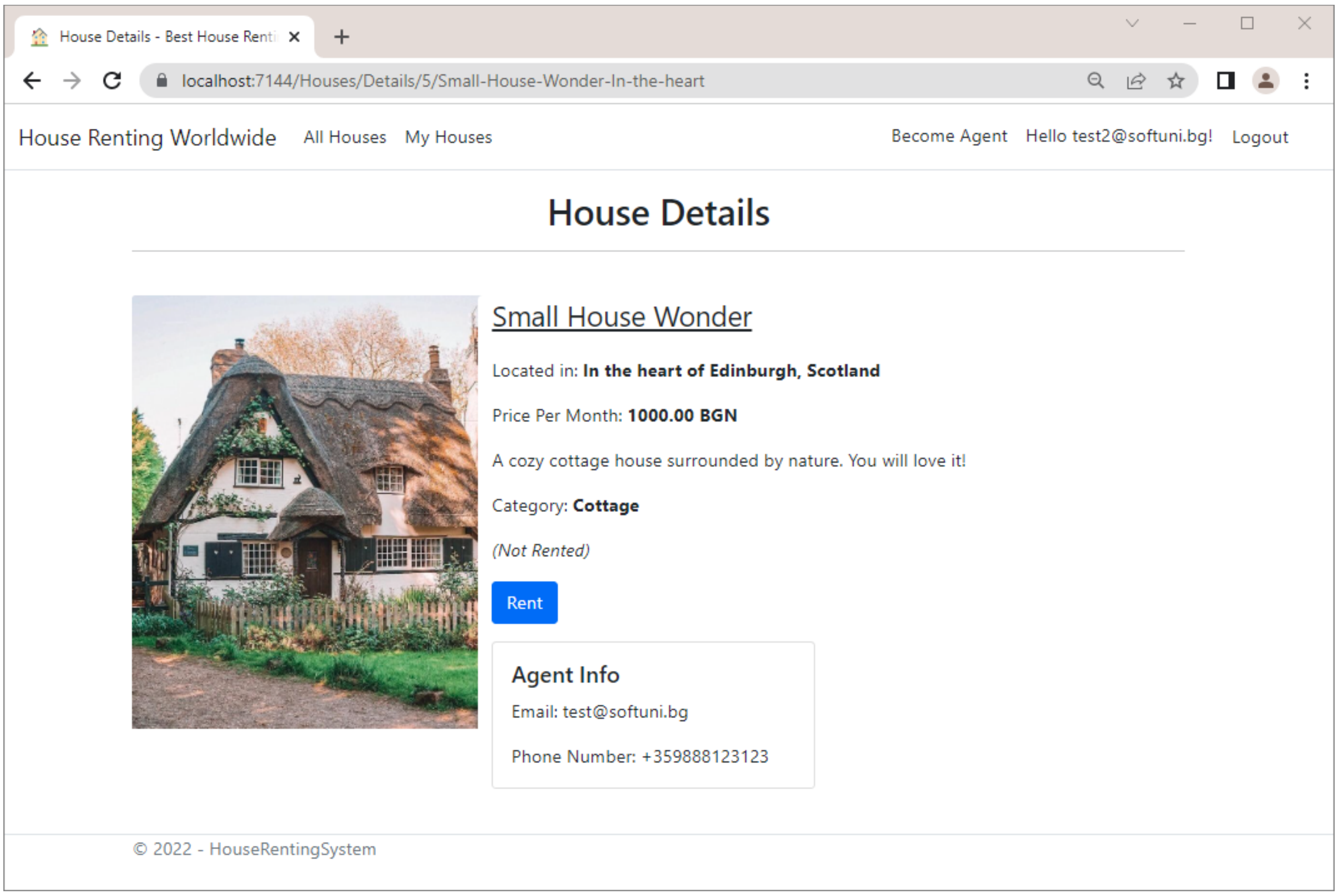


```
@model HouseServiceModel
@inject IHouseService houses
@inject IAgentService agents

<div class="col-md-4">
    <div class="card mb-3">
        <img class="card-img-top" src="@Model.ImageUrl" alt="House Image">
        <div class="card-body text-center">
            <h4>@Model.Title</h4>
            <h6>Address: <b>@Model.Address</b></h6>
            <h6>...
            <h6>(@(Model.IsRented ? "Rented" : "Not Rented"))</h6>
            <br />
            <a asp-controller="House" asp-action="Details" asp-route-id="@Model.Id"
               asp-asp-route-information="@Model.GetInformation()" class="btn btn-success">Details</a>
            @if (User.Identity.IsAuthenticated)
            {
                @if (await houses.HasAgentWithId(Model.Id, User.Id()))
                {
                    <a asp-controller="House" asp-action="Edit" asp-route-id="@Model.Id"
                        class="btn btn-warning">Edit</a>
                    <a asp-controller="House" asp-action="Delete" asp-route-id="@Model.Id"
                        class="btn btn-danger">Delete</a>
                }
                <p></p>
                @if (!Model.IsRented && await agents.ExistsById(User.Id()) == false )
                {
                    <form class="input-group-sm" asp-controller="House"
                        asp-action="Rent" asp-route-id="@Model.Id" method="post">
                        <input class="btn btn-primary" type="submit" value="Rent" />
                    </form>
                }
                else if (await houses.IsRentedByUserWithId(Model.Id, User.Id()))
                {
                    <form asp-controller="House" asp-action="Leave"
                  asp-route-id="@Model.Id" method="post">
                        <input class="btn btn-primary" type="submit" value="Leave" />
                    </form>
                }
            }
        </div>
    </div>
</div>
```

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**:


House Details - Best House Renti

localhost:7144/Houses/Details/5/Small-House-Wonder-In-the-heart

House Renting Worldwide All Houses My Houses Become Agent Hello test2@softuni.bg! Logout

House Details

Small House Wonder

Located in: **In the heart of Edinburgh, Scotland**

Price Per Month: **1000.00 BGN**

A cozy cottage house surrounded by nature. You will love it!

Category: **Cottage**

(Not Rented)

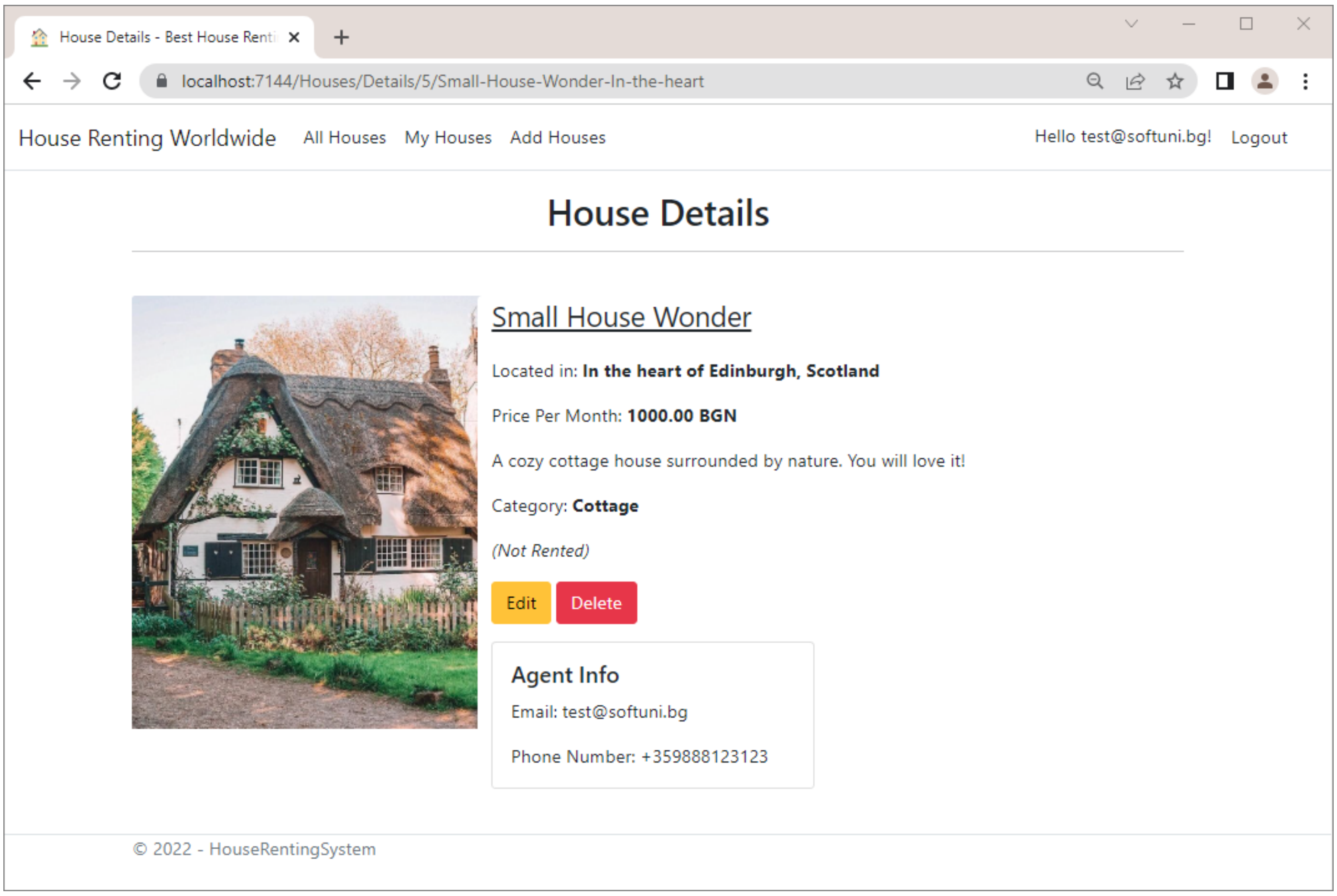
Rent

Agent Info

Email: test@softuni.bg

Phone Number: +359888123123

© 2022 - HouseRentingSystem


House Details - Best House Renti

localhost:7144/Houses/Details/5/Small-House-Wonder-In-the-heart

House Renting Worldwide All Houses My Houses Add Houses Hello test@softuni.bg! Logout

House Details

Small House Wonder

Located in: **In the heart of Edinburgh, Scotland**

Price Per Month: **1000.00 BGN**

A cozy cottage house surrounded by nature. You will love it!

Category: **Cottage**

(Not Rented)

Edit Delete

Agent Info

Email: test@softuni.bg

Phone Number: +359888123123

© 2022 - HouseRentingSystem

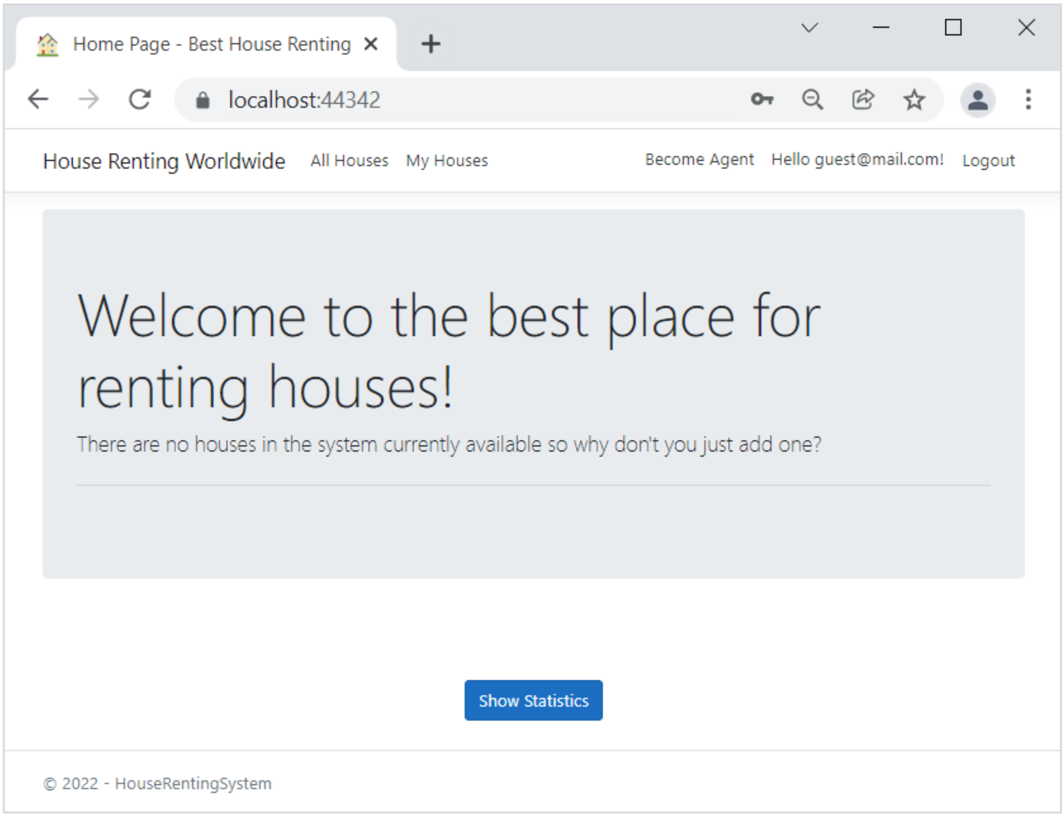
You already know how you should **modify the view**. When done, it should look like this:

```
<div class="form-inline">
    @if (User.Identity.IsAuthenticated)
    {
        @if (await houses.HasAgentWithId(Model.Id, User.Id()))
        {
            <a class="btn btn-warning" asp-controller="House" asp-action="Edit"
                asp-route-id="@Model.Id">Edit</a>
            <a class="ml-2 btn btn-danger" asp-controller="House" asp-action="Delete"
                asp-route-id="@Model.Id">Delete</a>
        }
        @if (!Model.IsRented && await agents.ExistsById(User.Id()) == false)
        {
            <form class="ml-2" asp-controller="House"
                asp-action="Rent" asp-route-id="@Model.Id" method="post">
                <input class="btn btn-primary" type="submit" value="Rent" />
            </form>
        }
        else if (await houses.IsRentedByUserWithId(Model.Id, User.Id()))
        {
            <form class="ml-2" asp-controller="House" asp-action="Leave"
                asp-route-id="@Model.Id" method="post">
                <input class="btn btn-primary" type="submit" value="Leave" />
            </form>
        }
    }
</div>
```

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**:

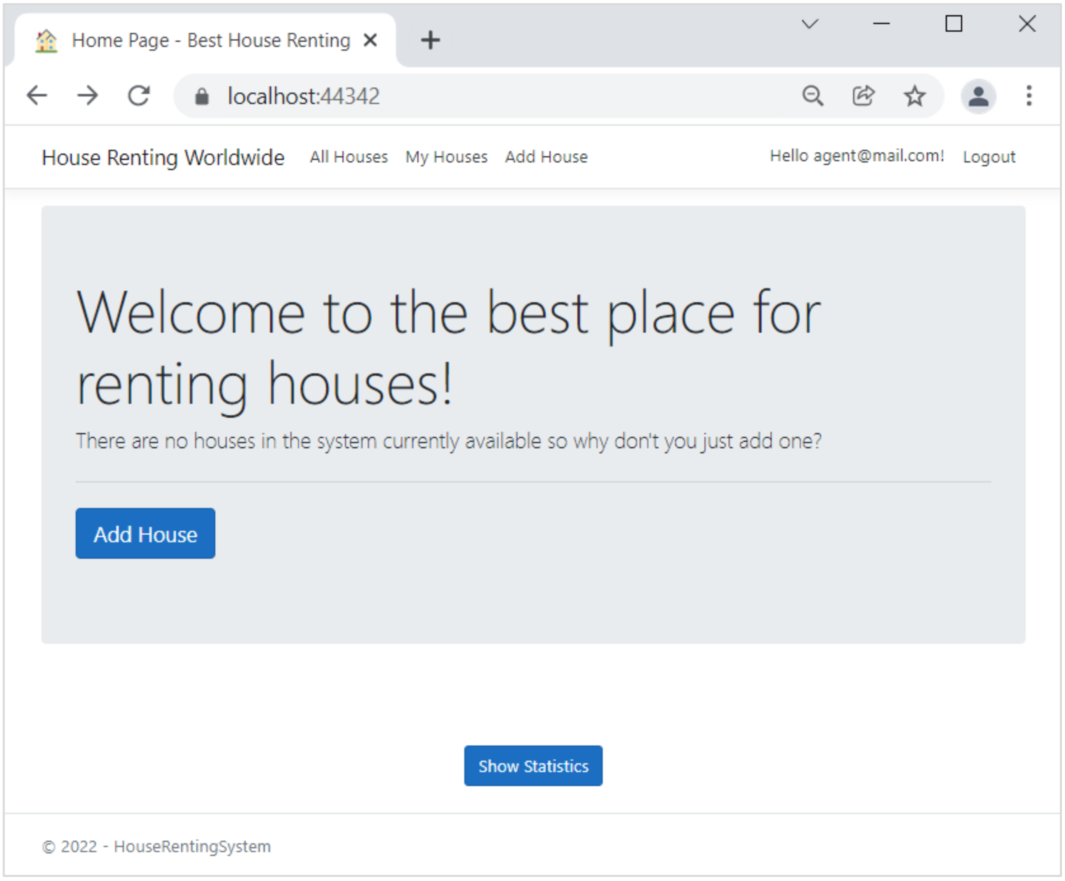


Welcome to the best place for renting houses!

There are no houses in the system currently available so why don't you just add one?

© 2022 - HouseRentingSystem

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



Welcome to the best place for renting houses!

There are no houses in the system currently available so why don't you just add one?

© 2022 - HouseRentingSystem

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

Index.cshtml

```
@model IEnumerable<HouseIndexServiceModel>
@inject IAgentService agents

@{
    ViewData["Title"] = "Home Page";
    var houses = Model.ToList();
}

@if (!houses.Any())
{
    <div class="mt-4 p-5 bg-light">
        <h1 class="display-4">Welcome to the best place for renting houses!</h1>
        <p class="lead">
            There are no houses in the system currently available
            so why don't you just add one?
        </p>
        <hr class="my-4">
        <p class="lead">
            @if (User.Identity.IsAuthenticated && await agents.ExistsById(User.Id()))
            {
                <a asp-controller="House" asp-action="Add" class="btn btn-primary btn-lg"
           role="button">Add House</a>
            }
        </p>
    </div>
}
```

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