Final Project: ASP.NET Core MVC App "Eventures"

In this workshop, we shall create a fully functional ASP.NET MVC App **"Eventures"** with SQL Server database using Entity Framework and MVC.

Eventures

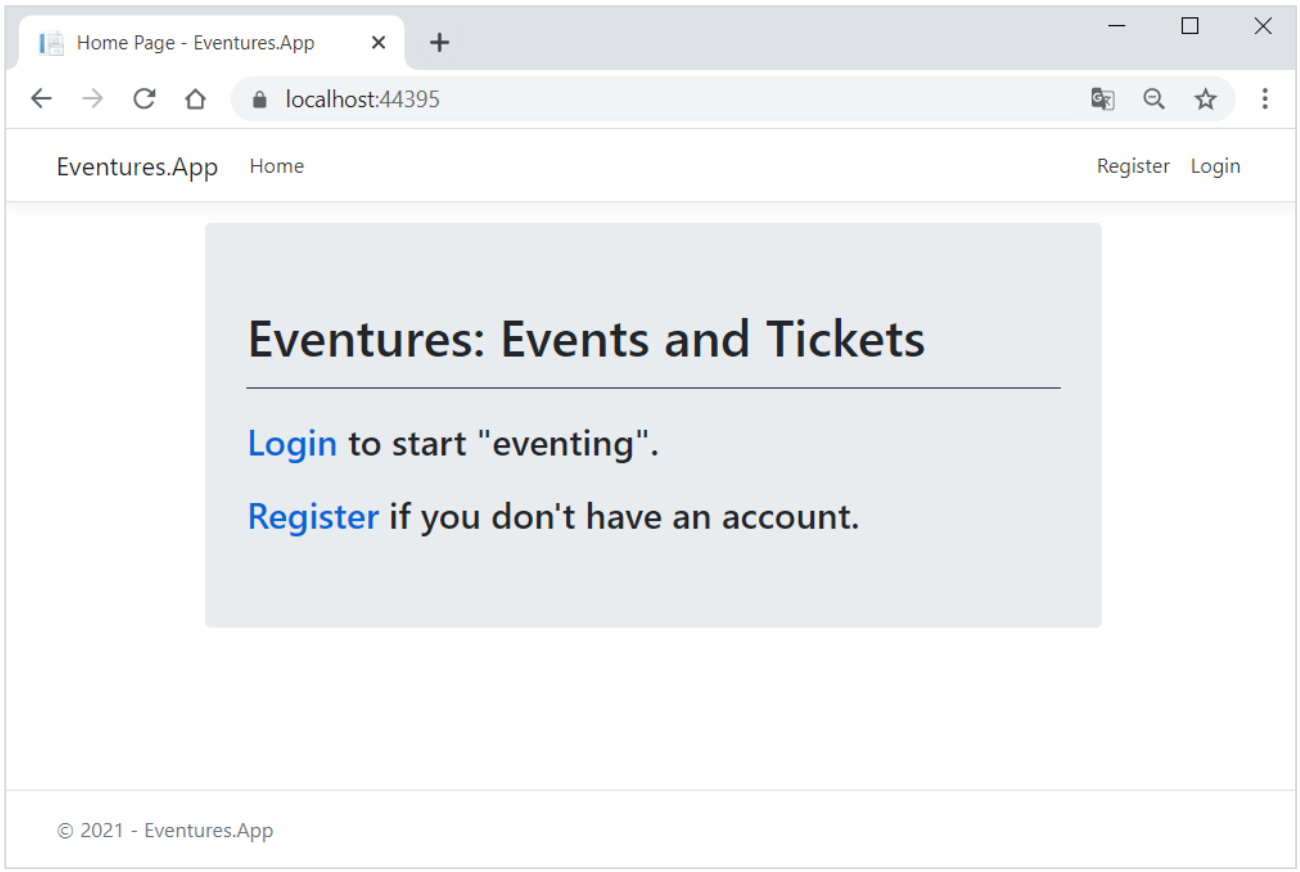
Eventures Inc. is a fast-rising newly made Start-Up Company, which specializes in **Event Tickets Sales**. It is said to be the killer of systems like Eventim, Eventbride, etc.

You have been appointed as the developer of the **main web application**. This is a great responsibility, so do your best and do not dissapoint your employers. The application functionality is not that complex, and it will be **separated** into **several parts**, each part consisting of **several tasks**.

Your current task is to create the **architecture** and **core logic** of the **application**, so get started.

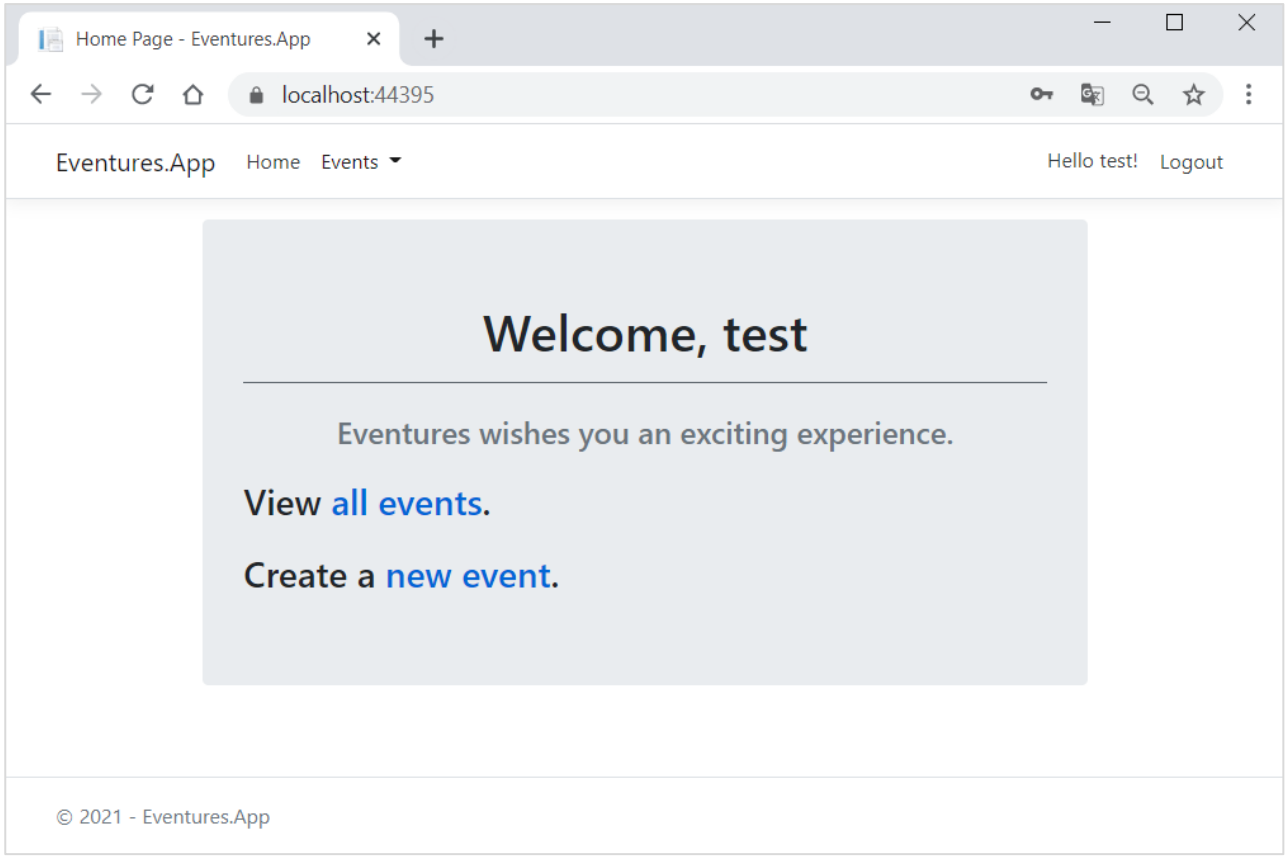
1. Pages

Home (Guest Users)

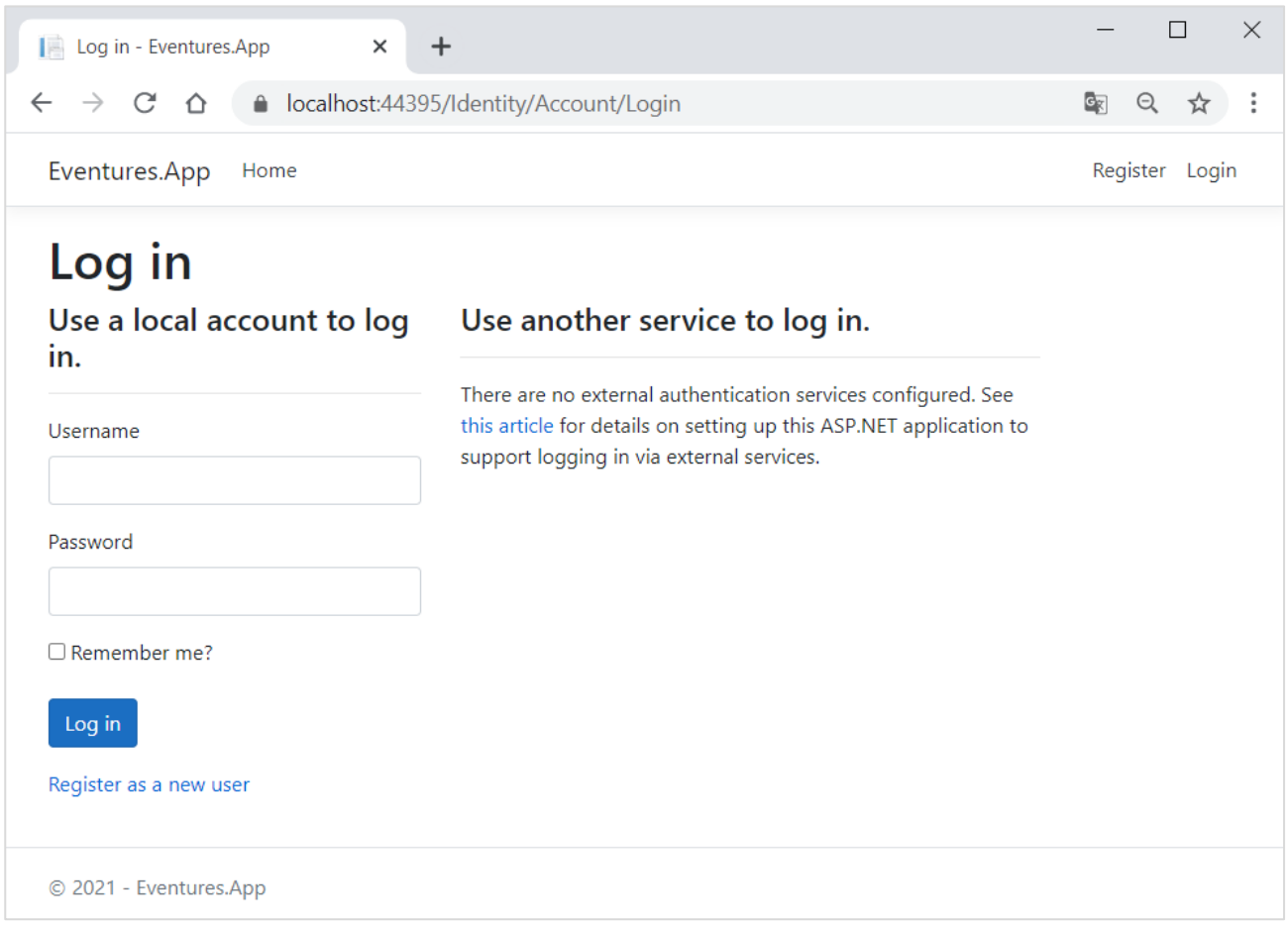


Home (Logged-in Users)

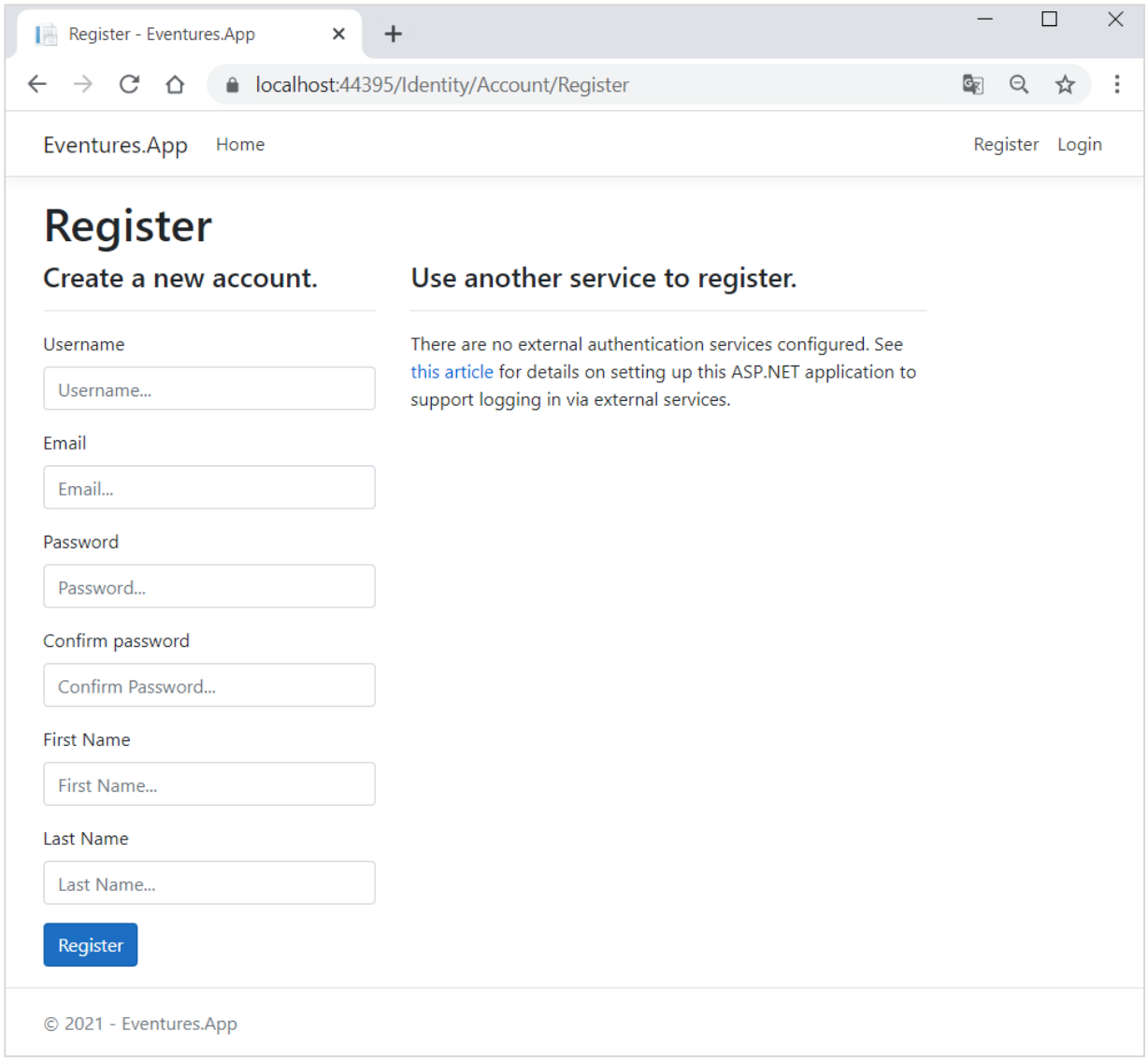
We are going to use a **user** with **username** `test` for testing our app.



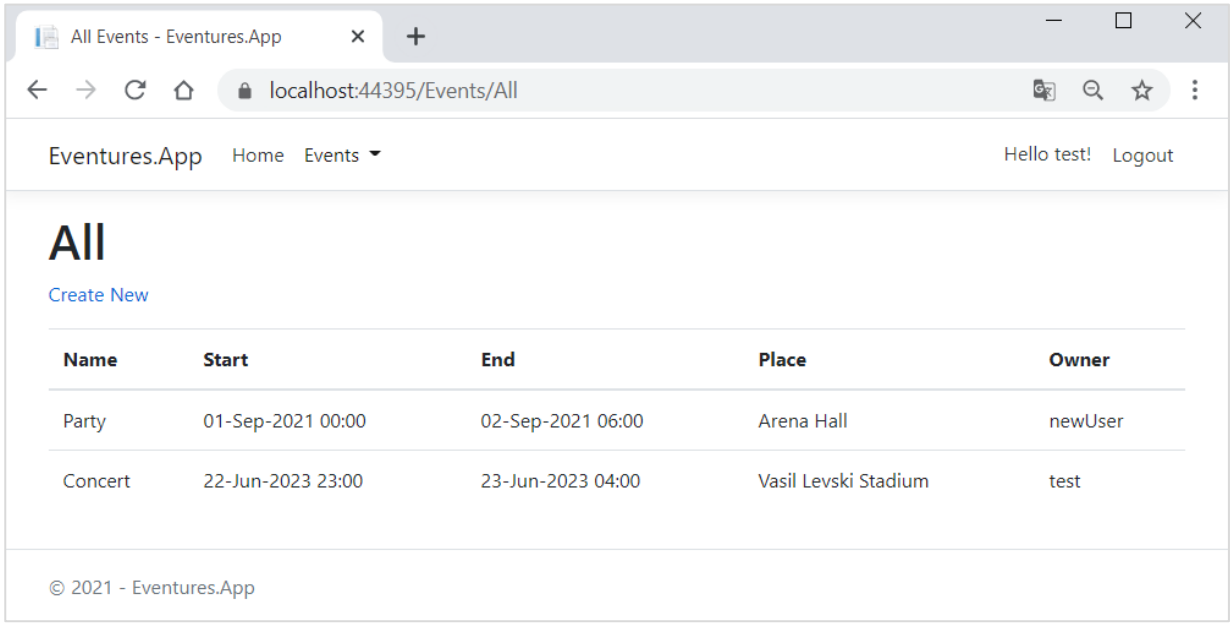
Login Page



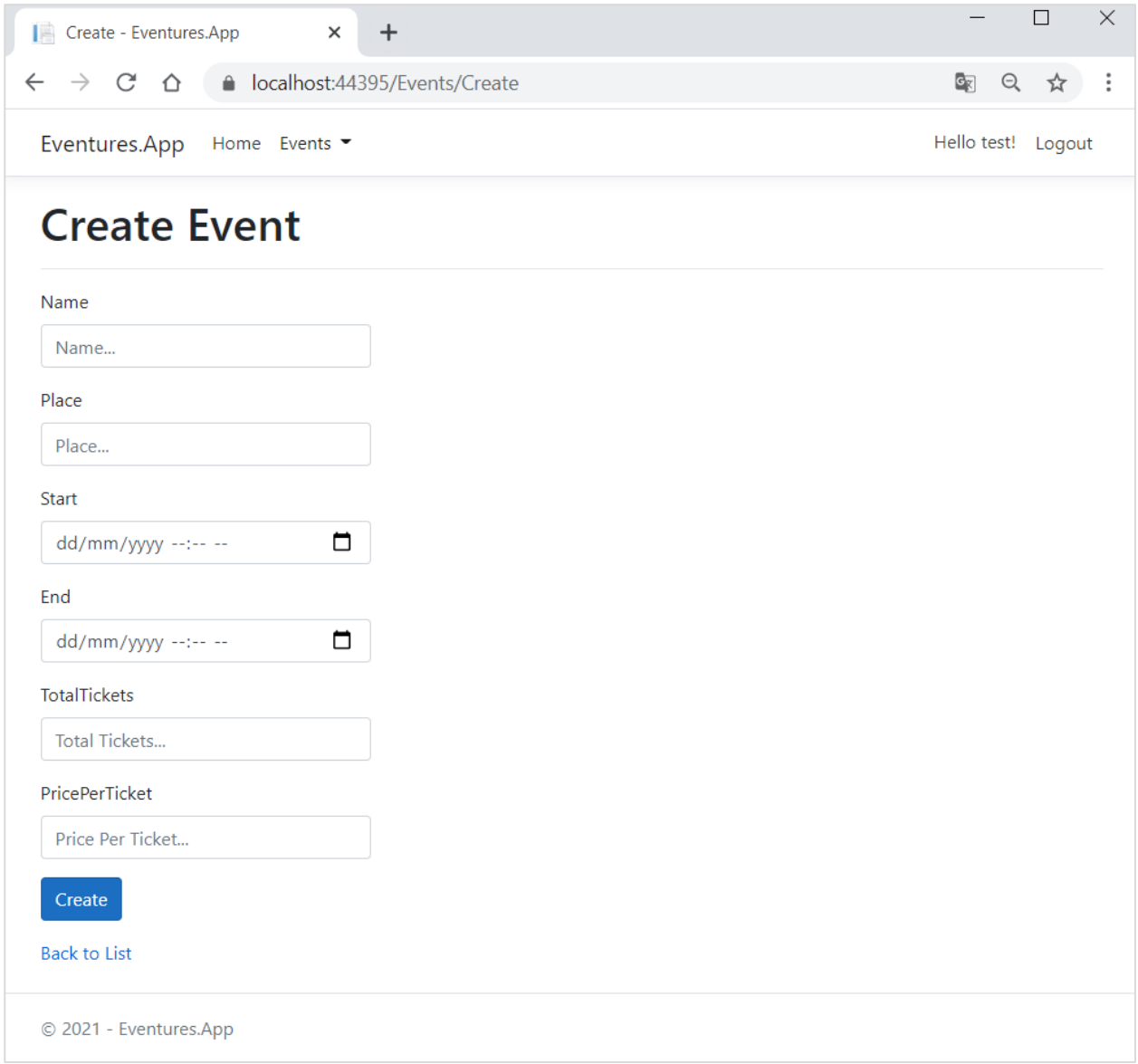
Register Page



All Events Page (Logged-in)



Create Event Page (Logged-in)



2. Data Storage

The core application logic requires **2 data models** to be implemented:

User

Has the following properties:

- **Username** – a **string** (from **IdentityUser**).
- **Password** – a **string** (from **IdentityUser**).
- **Email** – a **string** (from **IdentityUser**).
- **First Name** – a **string**.
- **Last Name** – a **string**.

Event

Has the following properties:

- **Id** – a **UUID**.
- **Name** – a **string**.
- **Place** – a **string**.

- **`Start`** – a **`DateTime`** object.
- **`End`** – a **`DateTime`** object.
- **`Total Tickets`** – an **`integer`**.
- **`Price Per Ticket`** – a **`double`** value.
- **`Owner`** – an **`EventuresUser`** object.
- **`OwnerId`** – a **`string`**.

3. Business Logic

Technical Requirements

The application should be an **ASP.NET Core Web** app. As such it should use **the most** of the **ASP.NET Core MVC Framework**.

Use **ASP.NET Core Identity** for authentication.

Functionality

The application should provide its **`Guest`** users (**not logged-in**) the functionality to **`register`** and **`login`**.

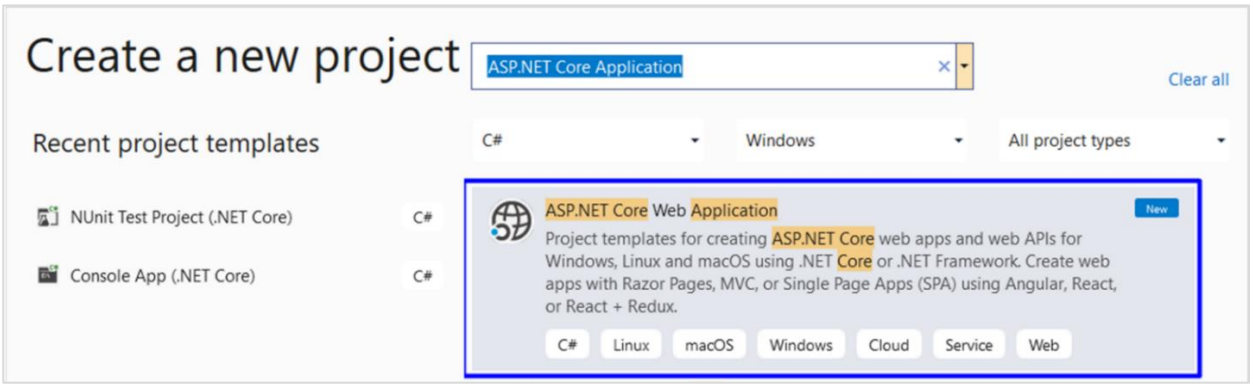
The application should provide its **`Regular`** users (**logged-in `Users`** with **`Role`** – **`User`**) the functionality to **create new `Events`**, **view all `Events`**.

Initial Setup

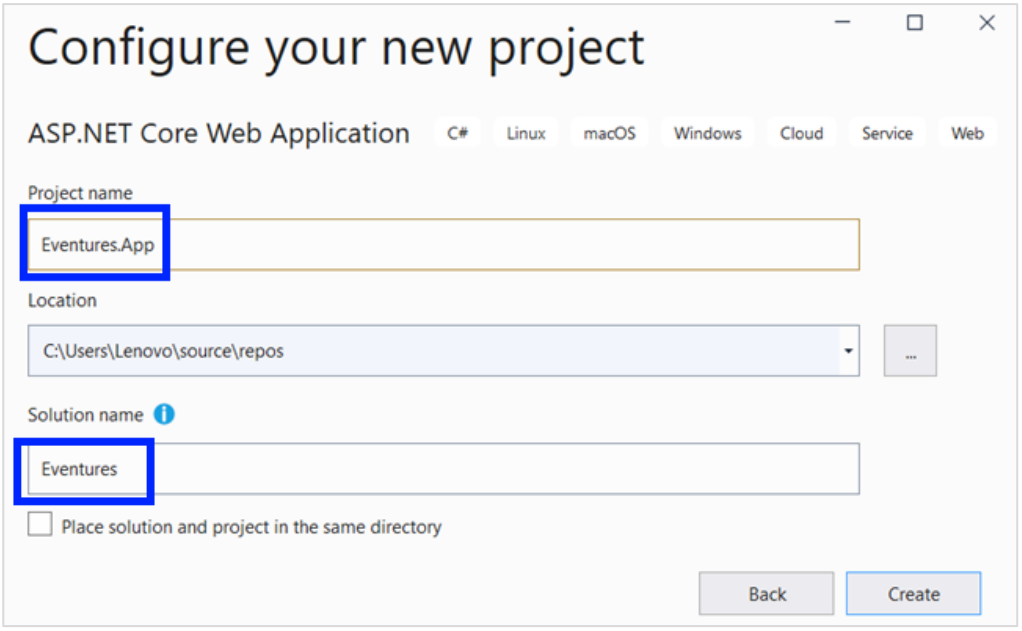
In this section we will setup our project and lay the foundations.

1. Create a New ASP.NET Core MVC Application

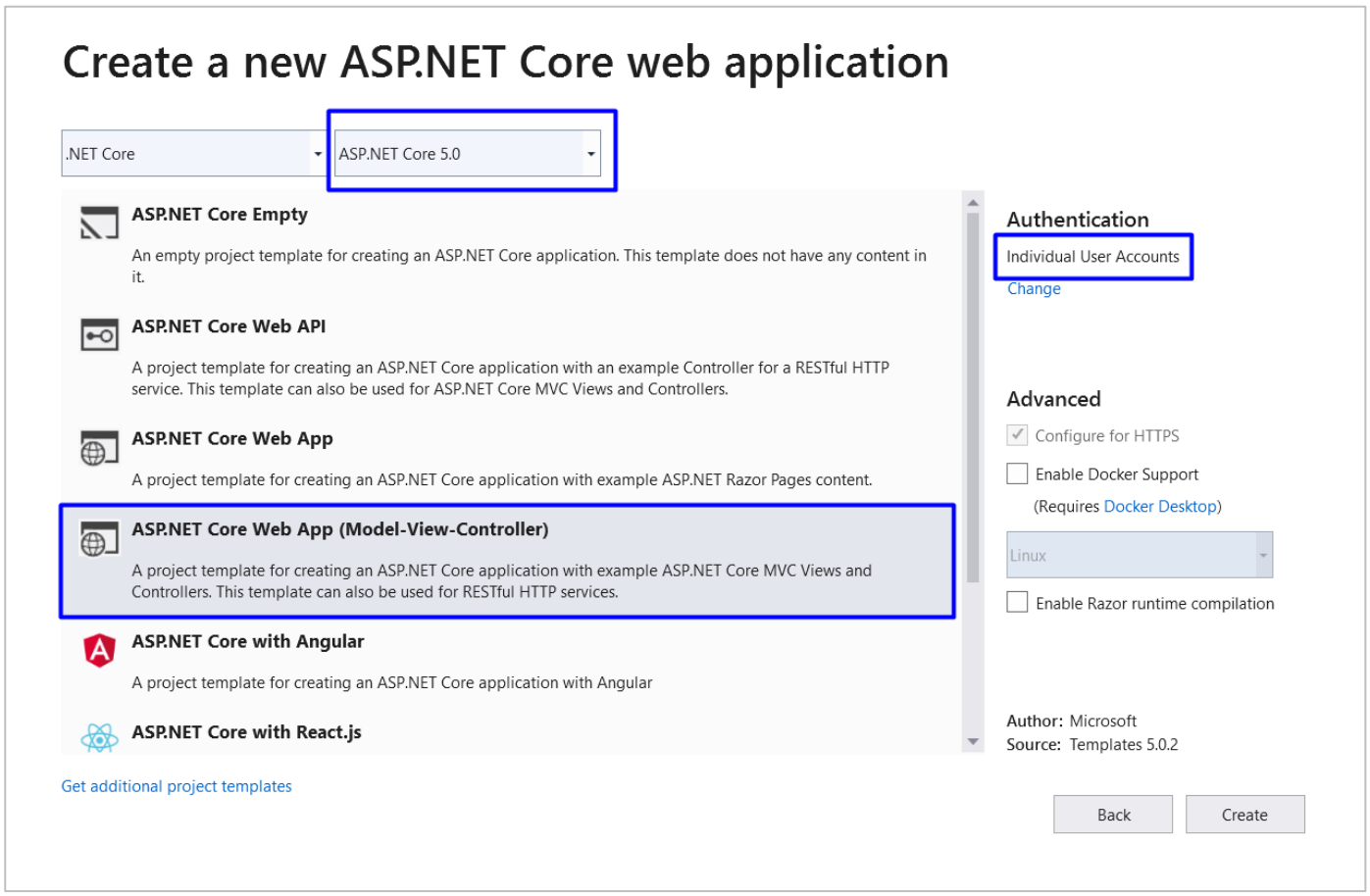
First, let's start by creating an **ASP.NET Core MVC Application** in Visual Studio as we did in previous exercises.



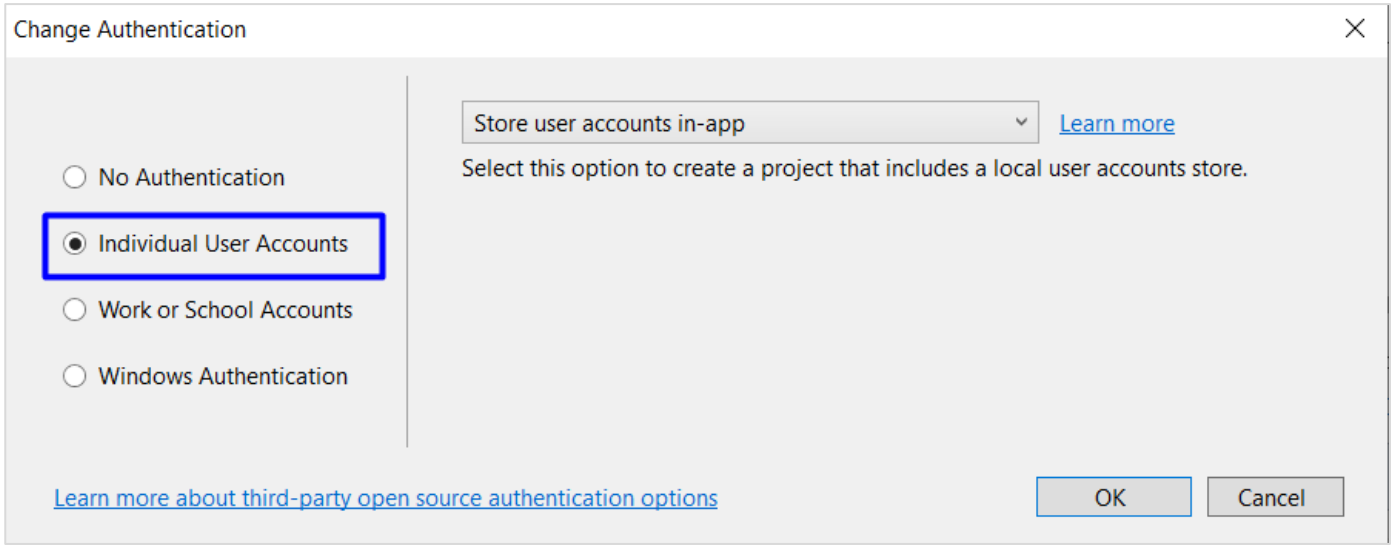
Don't forget to name the project appropriately, as if you leave this for later, you can encounter major problems. All code in the guide is made in a project with the name **`Eventures.App`** and solution name **`Eventures`**:



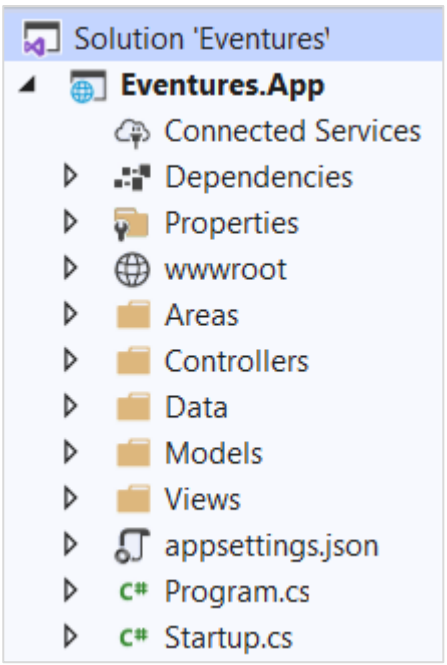
In the next window, change ASP.NET version to **`ASP.NET Core 5.0`**. Then, choose **MVC** and untick the "**Host in the cloud**" checkbox (if you use Visual Studio 2015). Also, you need to change the **authentication** to **`Individual User Accounts`**, as shown below:



To change authentication, press **`[Change]`** under **`Authentication`** on the right and choose **`Individual User Accounts`**. Then press **`[OK]`**:



Press **[OK]** and you should see the following project structure:



2. Change Database

First, go to **`appsetings.json`** and change the **default connection string** so that the newly created database has a suitable name:

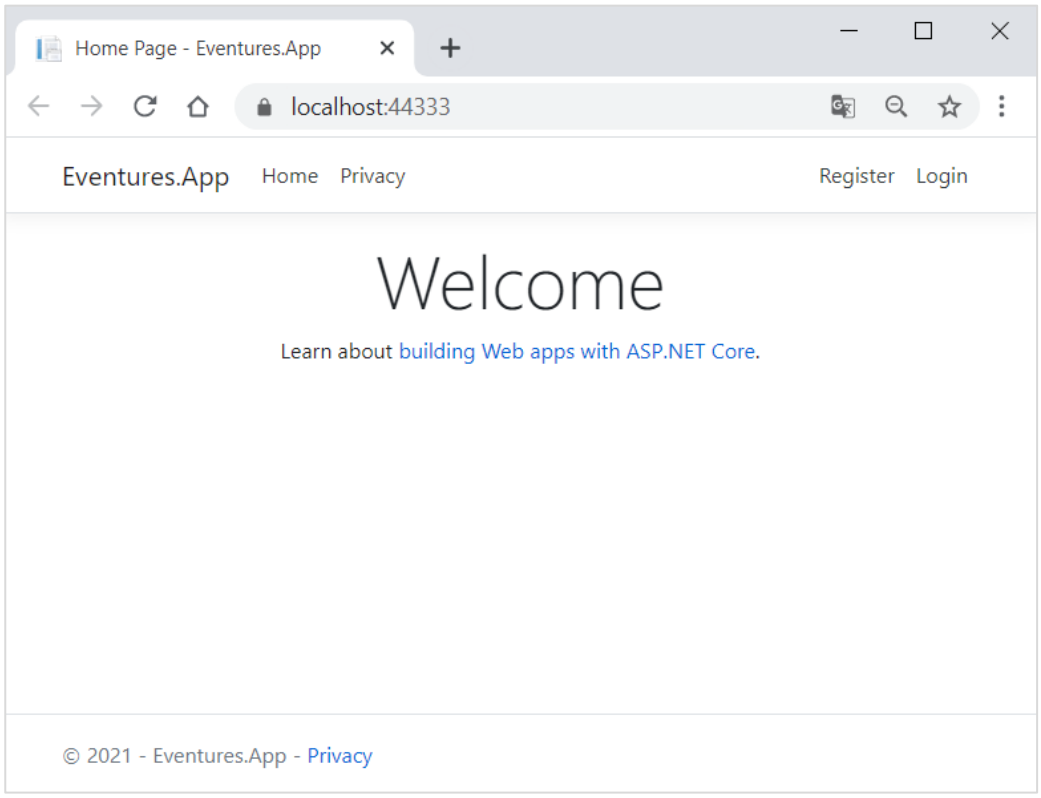
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=Eventures;T
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Now find **`ApplicationDbContext.cs`** in **`Data`** folder and ensure that database is created:

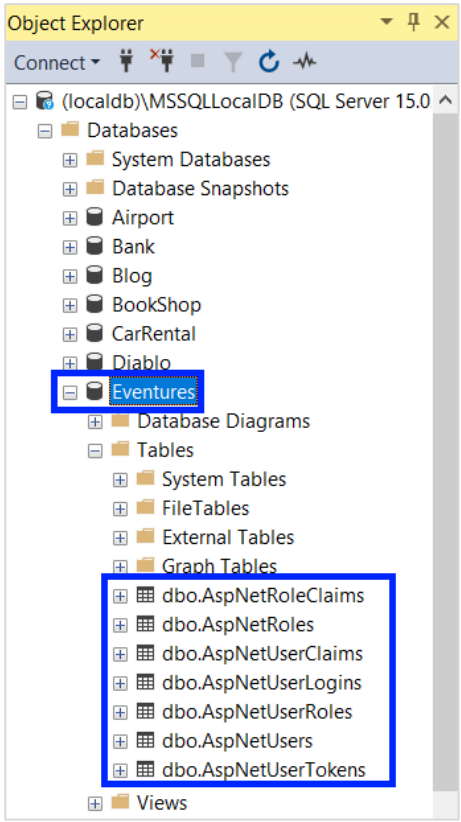
```
namespace Eventures.App.Data
{
    6 references
    public class ApplicationDbContext : IdentityDbContext
    {
        0 references
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
            this.Database.EnsureCreated();
        }
    }
}
```

3. Run the Application

Run the application to see what was generated by the Visual Studio MVC application template. Press **[Ctrl+F5]**.



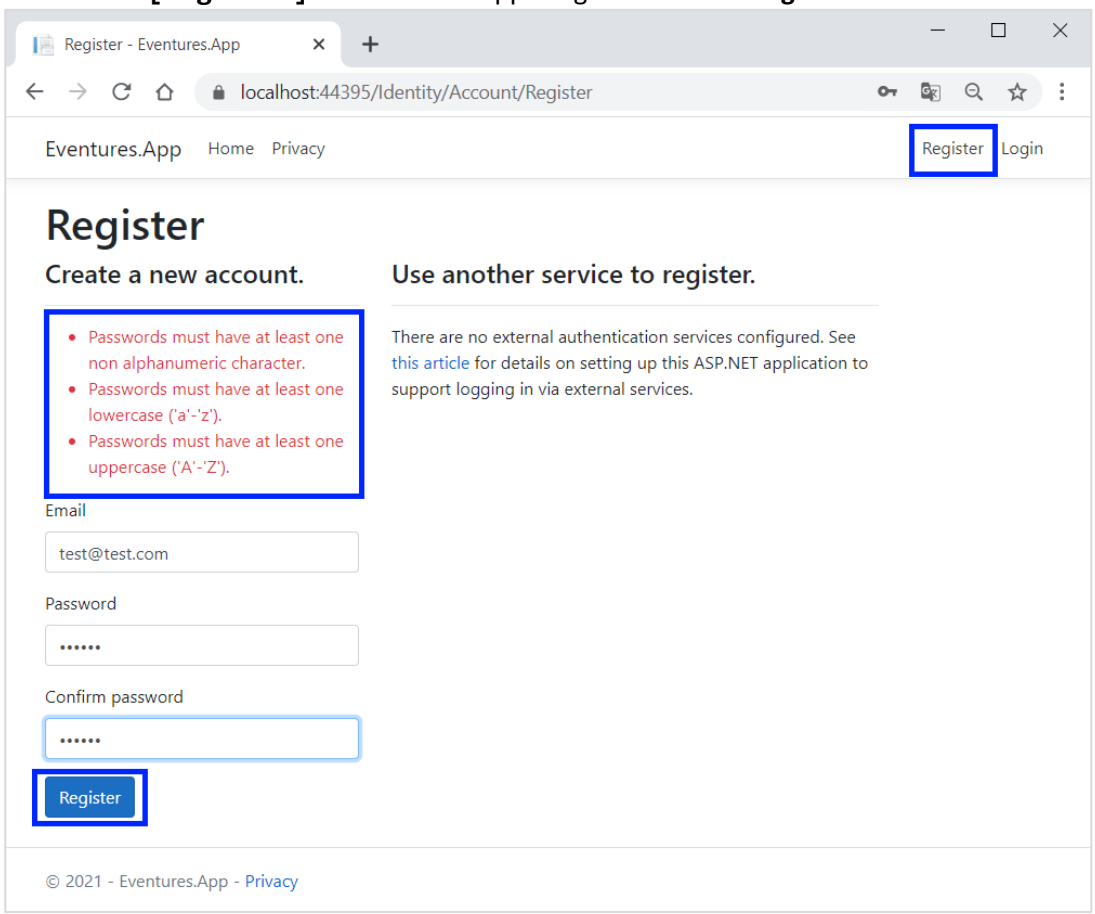
Go to **SSMS** and **refresh** it. Check if the new **`Eventures`** database with its tables is created.



User Setup

1. Register a User

Click on the **`[Register]`** button in the upper right corner and **register a user**.

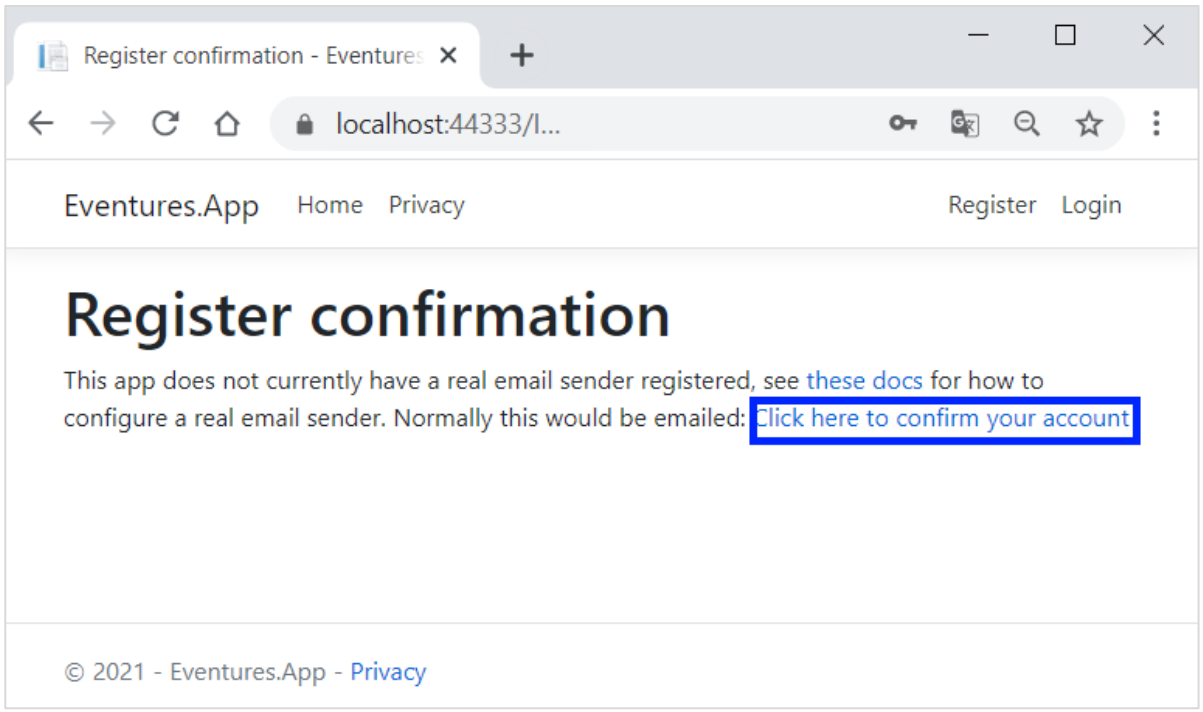


However, as you can see it is hard to think of a **password** with so many **requirements**. That is why we will **remove and change** some of them. To do so, go to the **`Startup.cs`** file and add the following code to the **`ConfigureServices(IServiceCollection services)`** method as shown below:

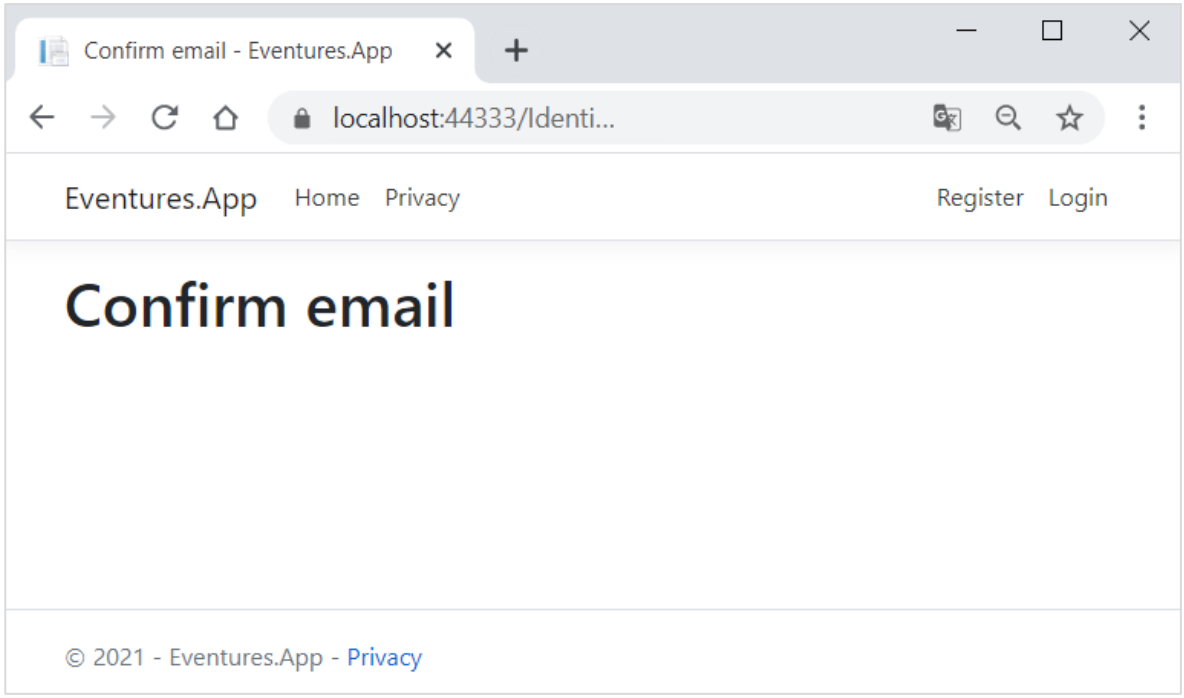
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
    services.AddRazorPages();

    services.Configure<IdentityOptions>(options =>
    {
        options.Password.RequireDigit = false;
        options.Password.RequiredLength = 5;
        options.Password.RequireLowercase = false;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = false;
        options.Password.RequiredUniqueChars = 0;
    });
}
```

Now, try again to register a user with a **simple password**. After pressing **`[Register]`**, you should see the page below.

We **have not configured register confirmation**, so the only thing you should do is press **`[Click here to confirm your account]`**.

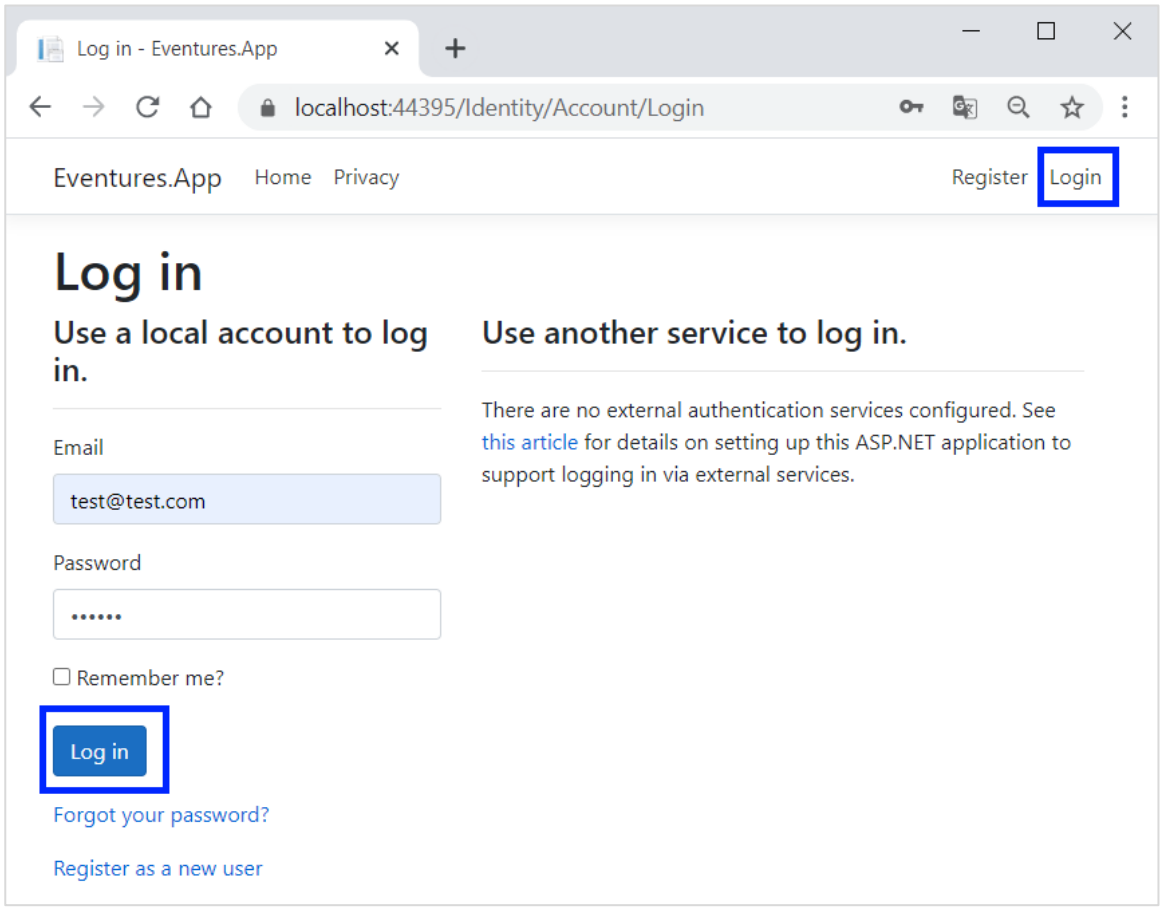


At the end, you should see the following page:

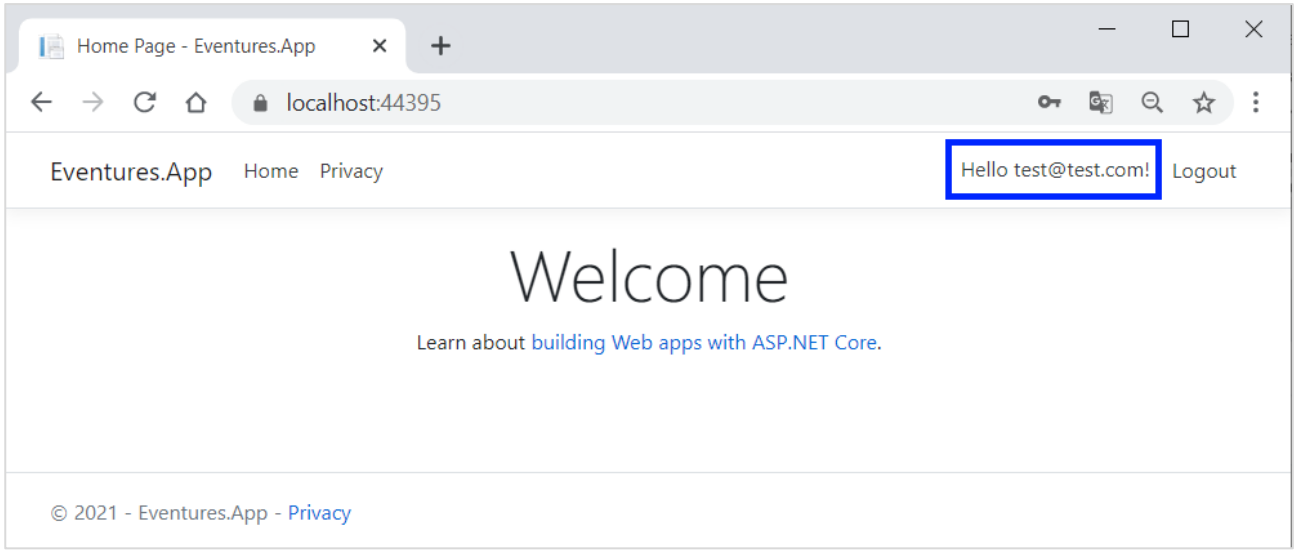


4. Log in with Registered User

You should already have a registration, so try **logging-in** by clicking on **`[Login]`** and entering your **credentials**. Then press **[Log in]**:



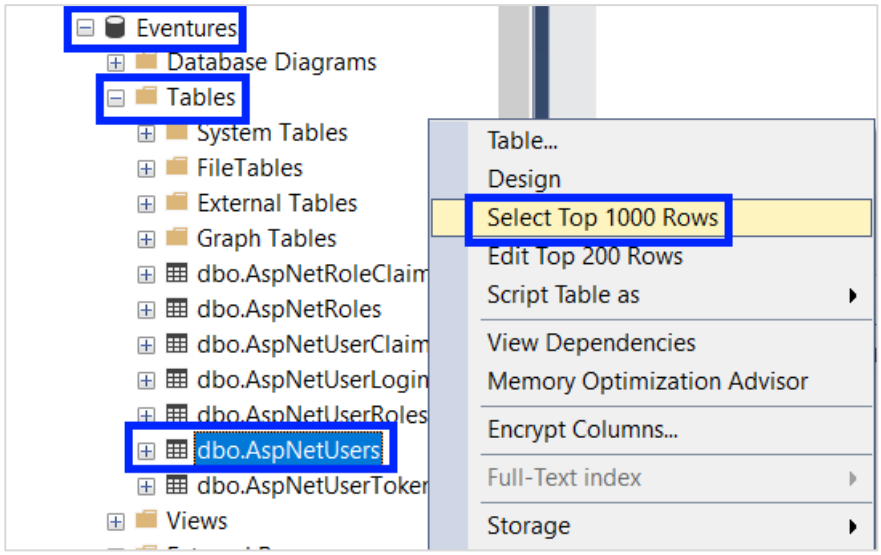
After **successful** login, you should be redirected to the **`Home page`**. Note that you are **logged-in** the system:



5. Check Database

After a bit of waiting, Entity Framework will **create the database schema** in SQL Server and the MVC application will **register the user** in the database.

Open the **database** to ensure it works as expected. You should have a database "**`Eventures`** " in the MS SQL Server Local DB, holding the **`AspNetUsers`** table, which should hold your registered **user account**:

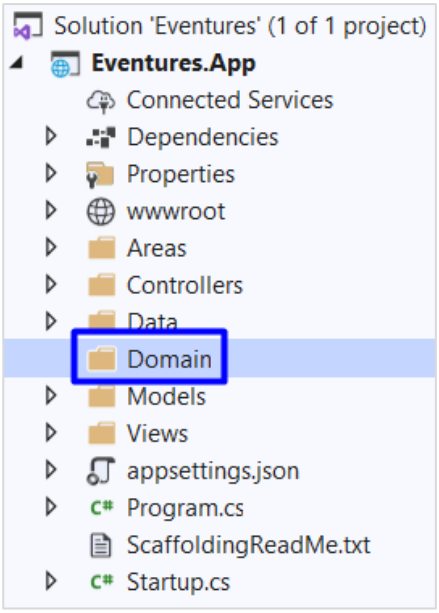


You should see **your user** as a result of the **"`Select Top 1000 Rows`"** command:

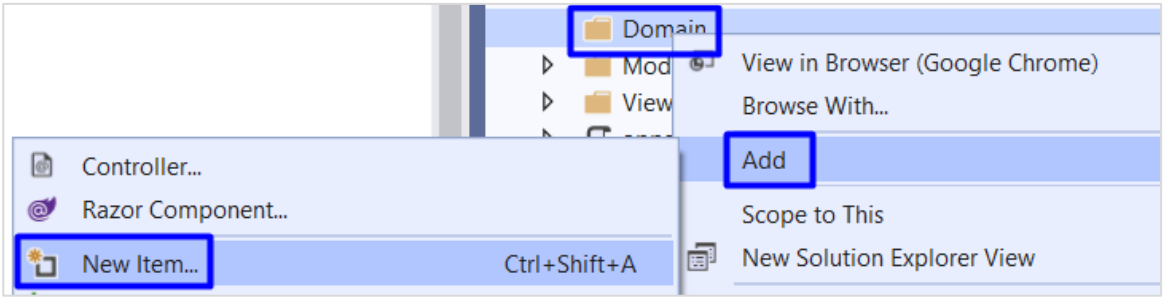
	Id	UserName	NormalizedUserName	Email	NormalizedEmail
1	4cc2c3a9-2c85-405f-a899-dd9c8053d549	test@test.com	TEST@TEST.COM	test@test.com	TEST@TEST.COM

6. Create User Class

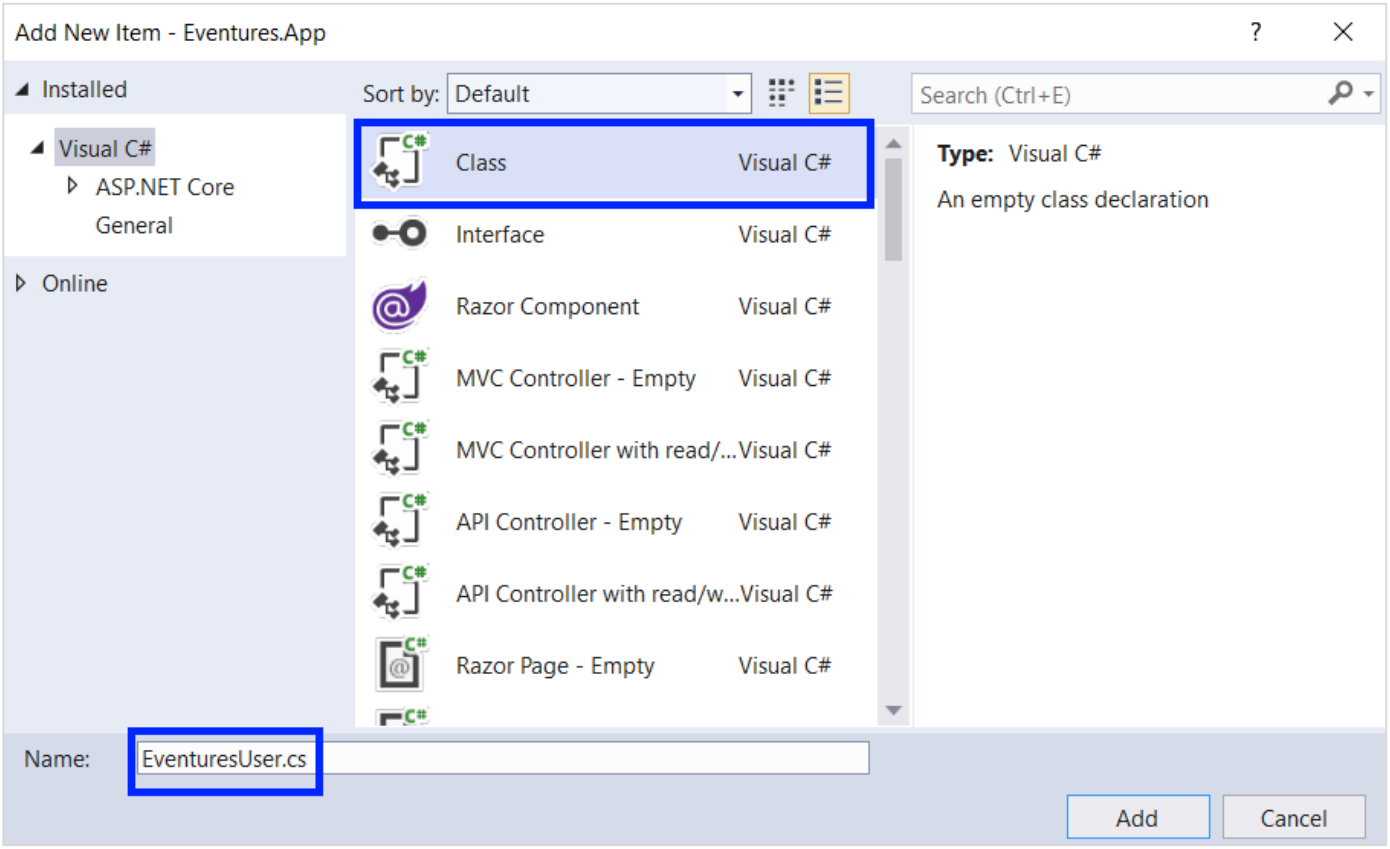
First, create a new **`Domain`** folder by **right-clicking** on **`Eventures.App -> [Add] -> [New folder]`**.



Let's add a **`EventuresUser`** class to hold our user's properties. **Right-click** on the **`Domain folder -> [Add] -> [New Item…]`**:

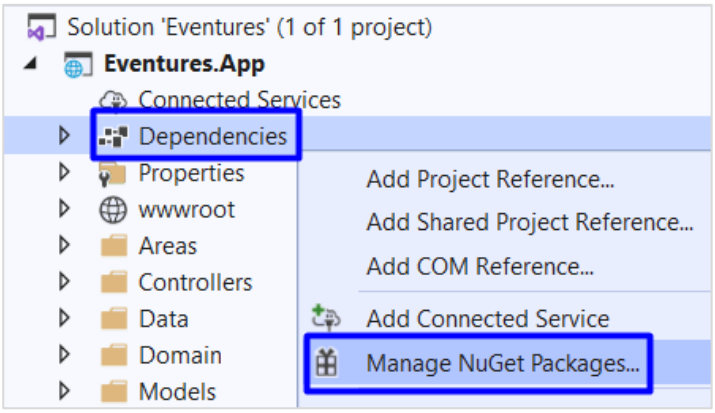


Then, add a class named **`EventuresUser`**:

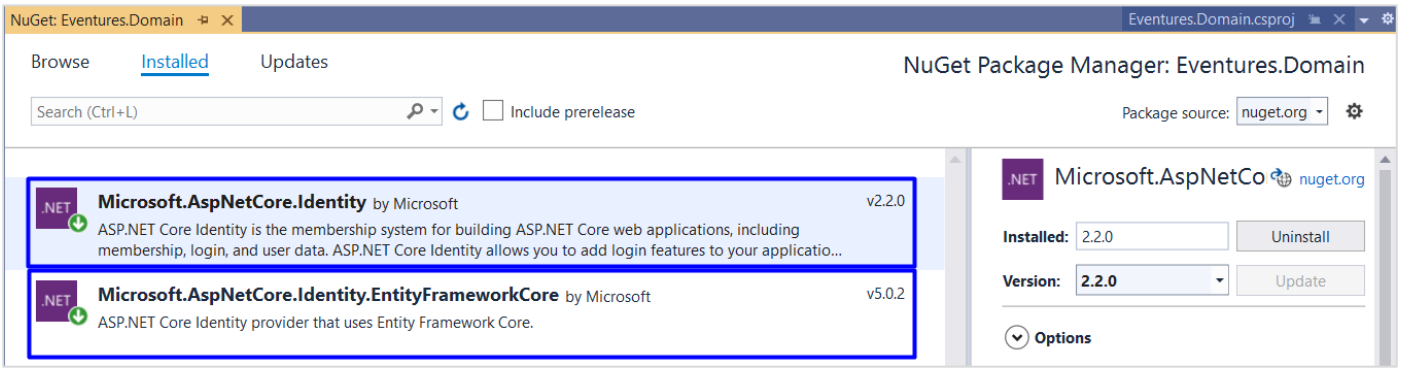


Install Packages

First, we need to **install some packages**, as we will need them. To do so, right-click on project **Dependencies** and choose **[Manage NuGet Packages…]**:



Now you should see the **NuGet Package Manager**. Search for and install **Microsoft.AspNetCore.Identity** and **Microsoft.AspNetCore.Identity.EntityFrameworkCore** packages. When finished, your installed packages should be the following:



Modify Class

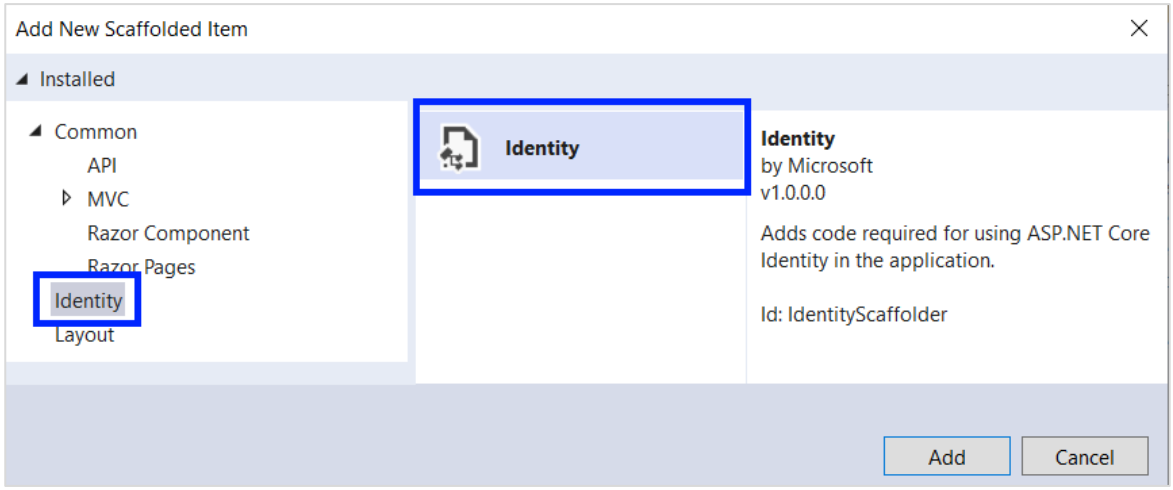
Then, make **EventuresUser** class inherit the **default IdentityUser** class. Modify the class like this:

```
namespace Eventures.App.Domain
{
    0 references
    public class EventuresUser : IdentityUser
    {
    }
}
```

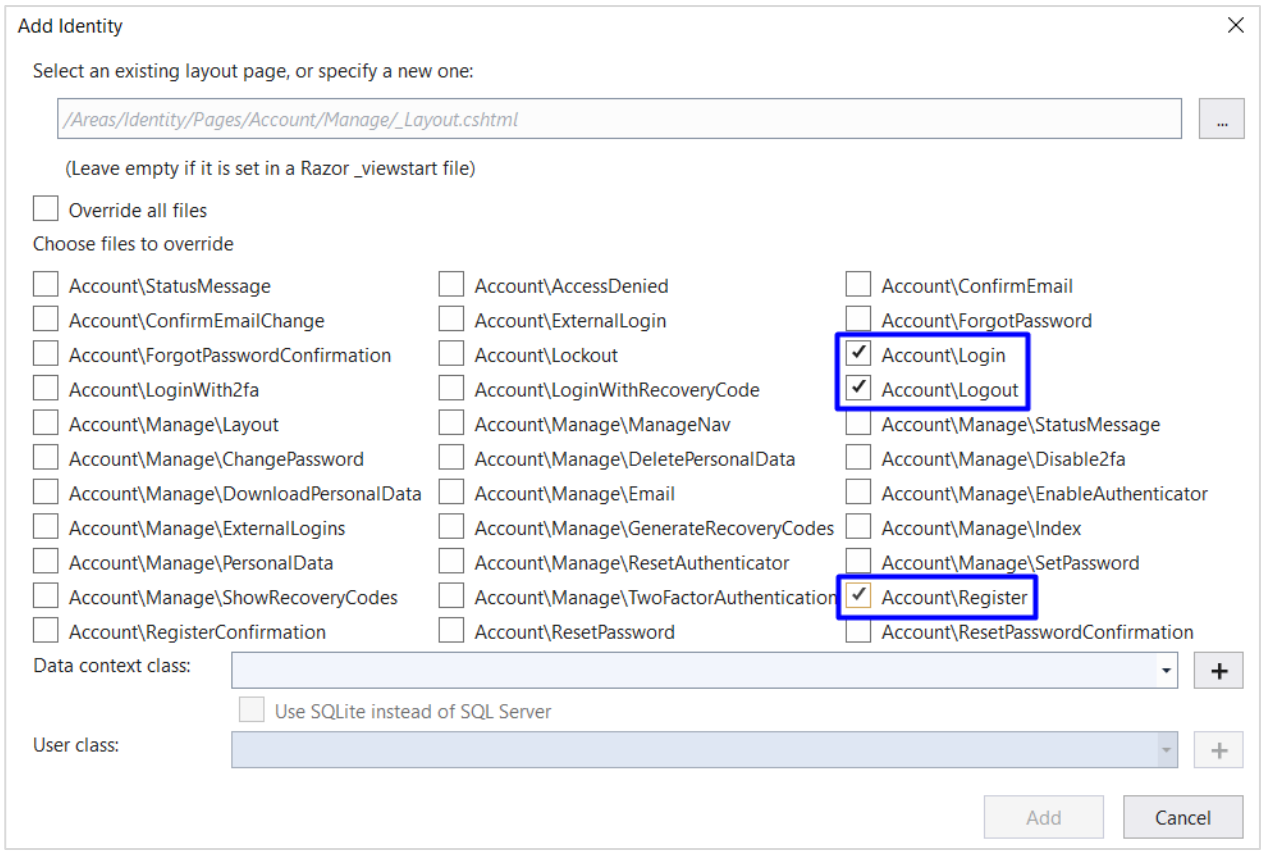
7. Add a New Scaffolded Item

In order to make modifications to **authentication** in our app, we need to add **Identity** to the **Eventures.App**.

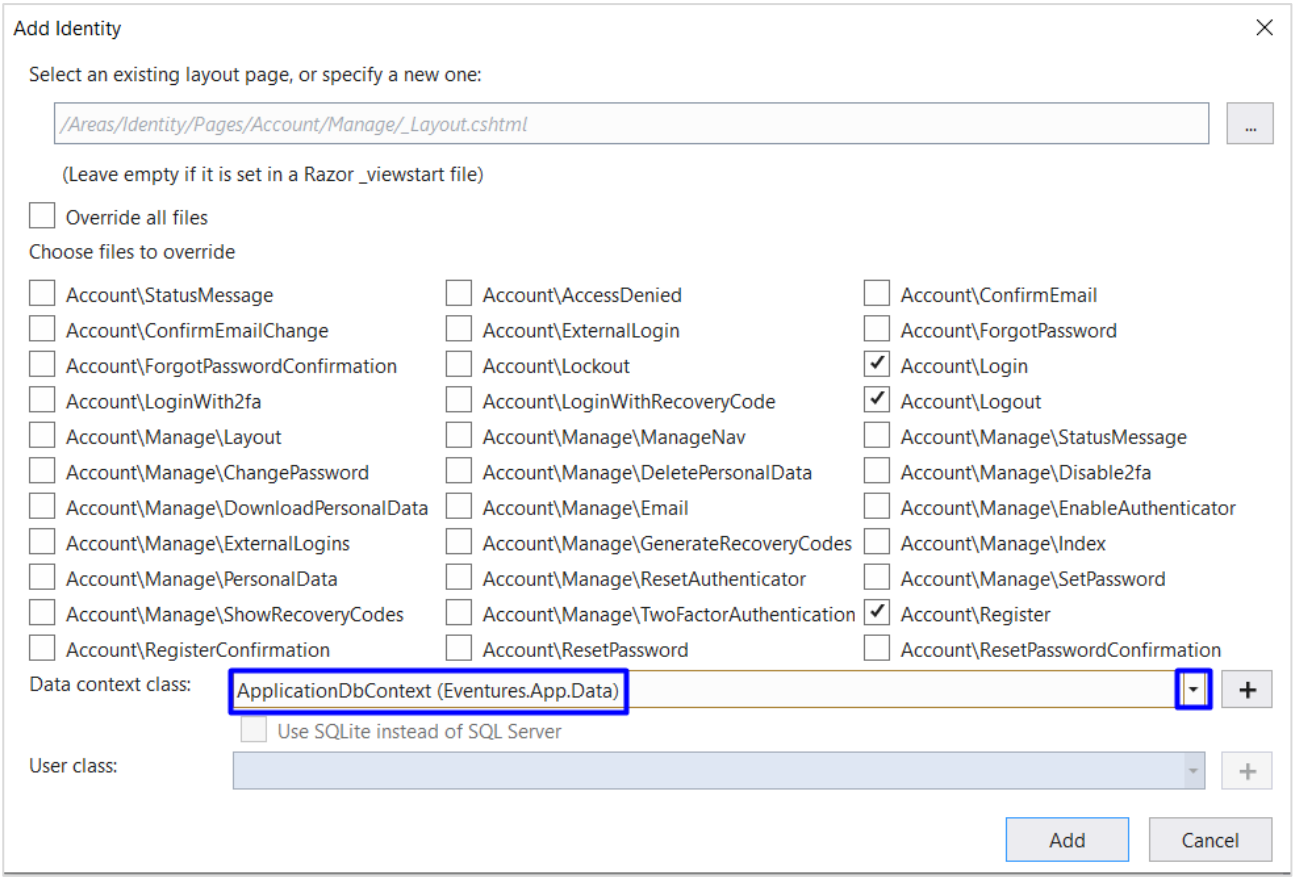
To add Identity, we should create a **new Scaffolded Item** first. **Right-click** on **Eventures.App** and choose **Add -> [New Scaffolded Item…]**. On the new window, go to the menu on the left, choose **Identity** and press **[Add]**.



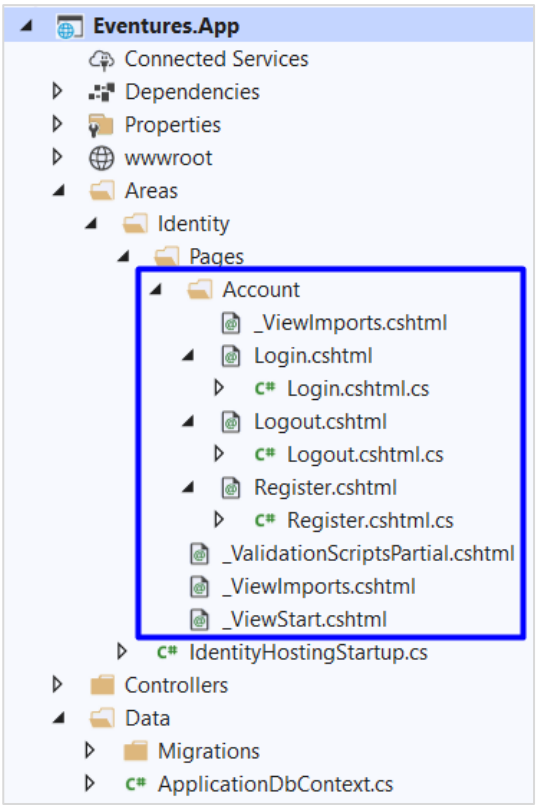
From the new window, put ticks to **`Login`**, **`Logout`** and **`Register`** boxes like shown below:



To add a **data context class** press **`[▾]`** and choose **`ApplicationDbContext`**:



Finally, press **[Add]**. The newly created files and folders should be present:

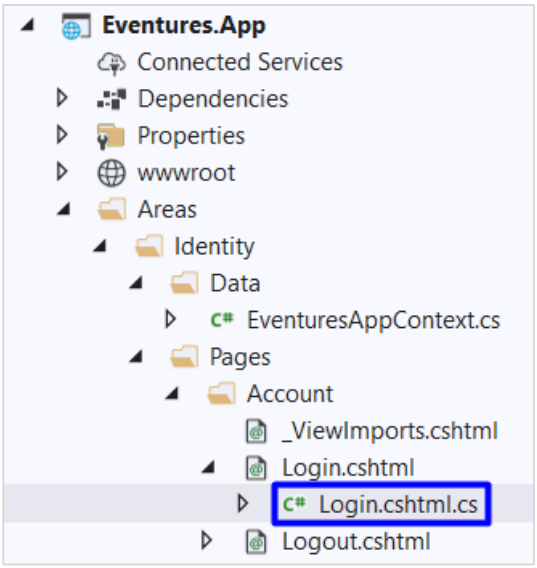


Use Our EventuresUser

In order to use the **EventuresUser** we already created, instead of the default **IdentityUser**, we need to make **modifications** to different files.

Modify Login.cshtml.cs

First, go to the **`Login.cshtml.cs`** file and change **`IdentityUser`** to **`EventuresUser` everywhere**:




```
namespace Eventures.App.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    8 references
    public class LoginModel : PageModel
    {
        private readonly UserManager<EventuresUser> _userManager;
        private readonly SignInManager<EventuresUser> _signInManager;
        private readonly ILogger<LoginModel> _logger;

        0 references
        public LoginModel(SignInManager<EventuresUser> signInManager,
            ILogger<LoginModel> logger,
            UserManager<EventuresUser> userManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
        }
```

Note: you will need to add "**`using Eventures.App.Domain;`**" to all modified files so that **`EventuresUser`** exists.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using Eventures.App.Domain;
```

Modify Logout.cshtml.cs and Register.cshtml.cs

Change **`IdentityUser`** to **`EventuresUser` everywhere** in **`Logout.cshtml.cs`** and **`Register.cshtml.cs`** files as we did in the **`Login.cshtml.cs`**.

The **Logout.cshtml.cs** should be changed like this:

```
namespace Eventures.App.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    8 references
    public class LogoutModel : PageModel
    {
        private readonly SignInManager<EventuresUser> _signInManager;
        private readonly ILogger<LogoutModel> _logger;

        0 references
        public LogoutModel(SignInManager<EventuresUser> signInManager, ILogger<LogoutModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }
```

The **Register.cshtml.cs** should be changed like this:

```
namespace Eventures.App.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    8 references
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<EventuresUser> _signInManager;
        private readonly UserManager<EventuresUser> _userManager;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;

        0 references
        public RegisterModel(
            UserManager<EventuresUser> userManager,
            SignInManager<EventuresUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
        }
```

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = new EventuresUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, Input.Password);
```

Modify _LoginPartial.cshtml

Go to **Views -> Shared** and modify the **_LoginPartial.cshtml** file, as well. It should be changed like this:

```
@using Microsoft.AspNetCore.Identity
@using Eventures.App.Domain
@inject SignInManager<EventuresUser> SignInManager
@inject UserManager<EventuresUser> UserManager
```

Modify Startup.cs

Make a small change to the **`Startup.cs`** file. Change **`services.AddDefaultIdentity()`** to **`services.AddIdentity()`** the following way:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddIdentity<EventuresUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();
    services.AddControllersWithViews();
    services.AddRazorPages();
```

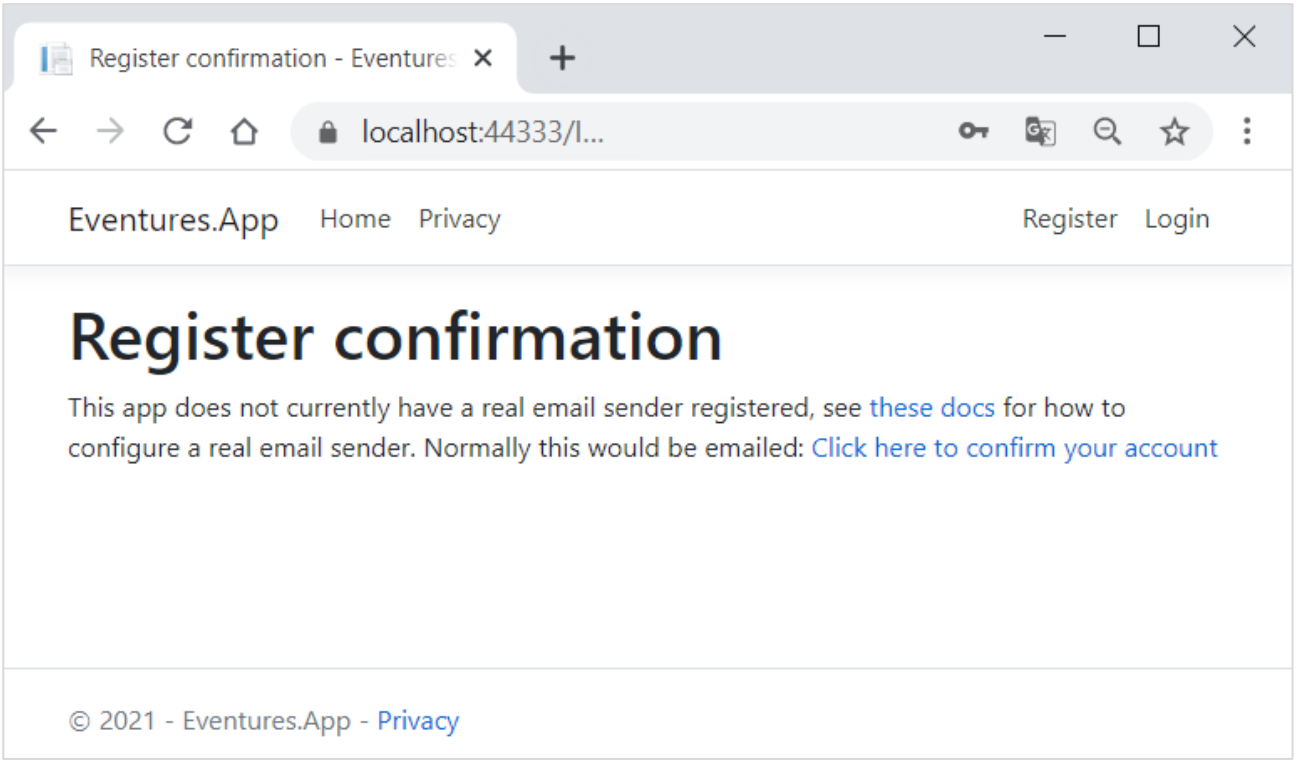
Modify ApplicationDbContext.cs

Finally, make changes to the **`ApplicationDbContext.cs`** file in the **`Data`** folder:

```
public class ApplicationDbContext : IdentityDbContext<EventuresUser>
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
        this.Database.EnsureCreated();
    }
}
```

8. Delete Email Sender

As you know, after **successful registration** this page appears:



However, we **will not** implement **account confirmation**, so we can just remove it because it will create an **`error`** if we try to register now. So, go to the **`Register.cshtml.cs`** file and modify it. **Remove** the **`emailSender`** as shown below:

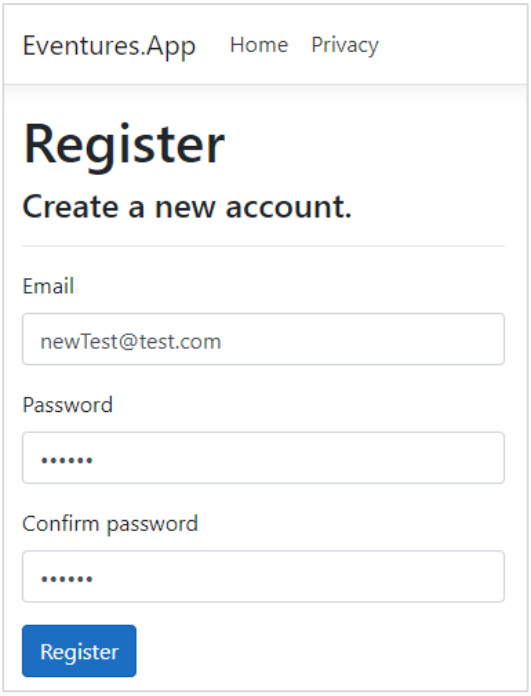
```
private readonly SignInManager<IdentityUser> _signInManager;
private readonly UserManager<IdentityUser> _userManager;
private readonly ILogger<RegisterModel> _logger;
private readonly IEmailSender _emailSender;
```

```
public RegisterModel(
    UserManager<IdentityUser> userManager,
    SignInManager<IdentityUser> signInManager,
    ILogger<RegisterModel> logger,
    IEmailSender emailSender)
{
    _userManager = userManager;
    _signInManager = signInManager;
    _logger = logger;
    _emailSender = emailSender;
}
```

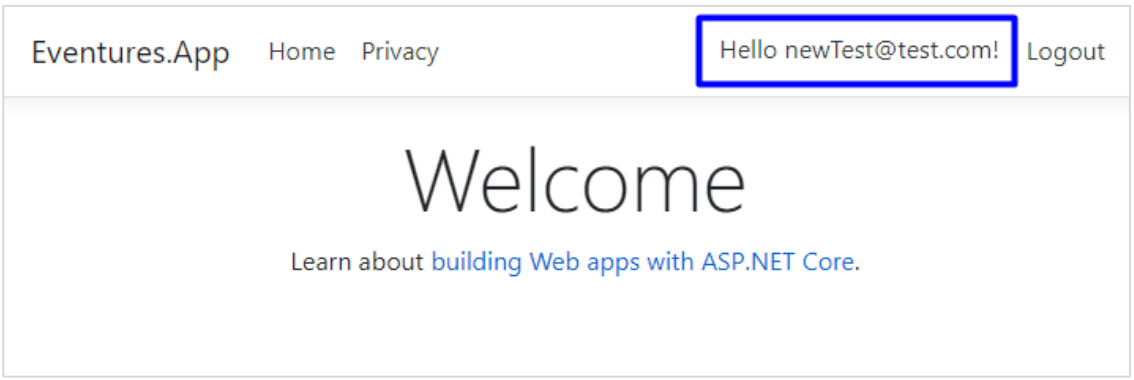
```
await _emailSender.SendEmailAsync
    (Input.Email, "Confirm your email",
    $"Please confirm your account by " +
    $"<a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
```

9. Run the Application

We made a lot of **changes**, so it is a good idea to **check** if the app is **running properly**. Press **`[Ctrl+F5]`** to run the app. **Test** its functionalities to check if they work **correctly**. **Register** a new user. If an **exception** is thrown, go back to previous steps and try finding what you have missed.



After successful registration, you should be **redirected** to the **`Home page`** and you should be **logged-in** with your new user:



Pages Setup

10. Remove Privacy Page

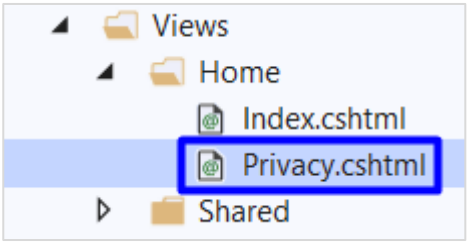
We won't need the **`Privacy page`** in our app, so we can just **delete** it.

First, to **remove** it from the **navigation bar** and the **footer**, go to **`_Layout.cshtml.cs`** in **`Views -> Shared`** and **delete** the **`<li>`** and **`<a>`** tags shown on the pictures below.

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
    <partial name="_LoginPartial" />
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
    </ul>
</div>
```

```
<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2021 - Eventures.App - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
</footer>
```

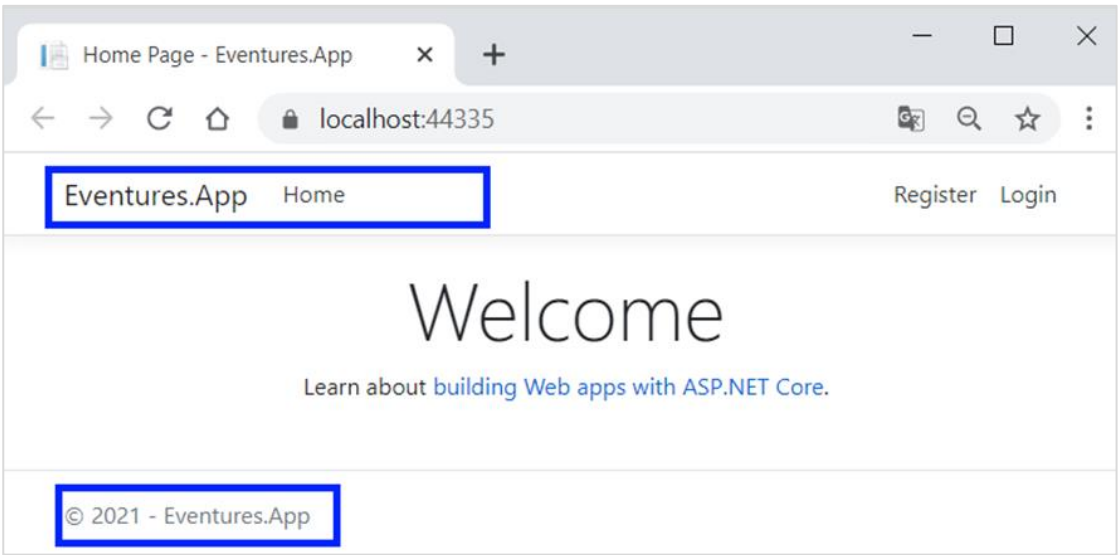
Then, search for the **`Privacy.cshtml`** file in **`Views -> Home`** and delete it.



Finally, let's **delete** the **`Privacy IAction`** from **`HomeController.cs`**:

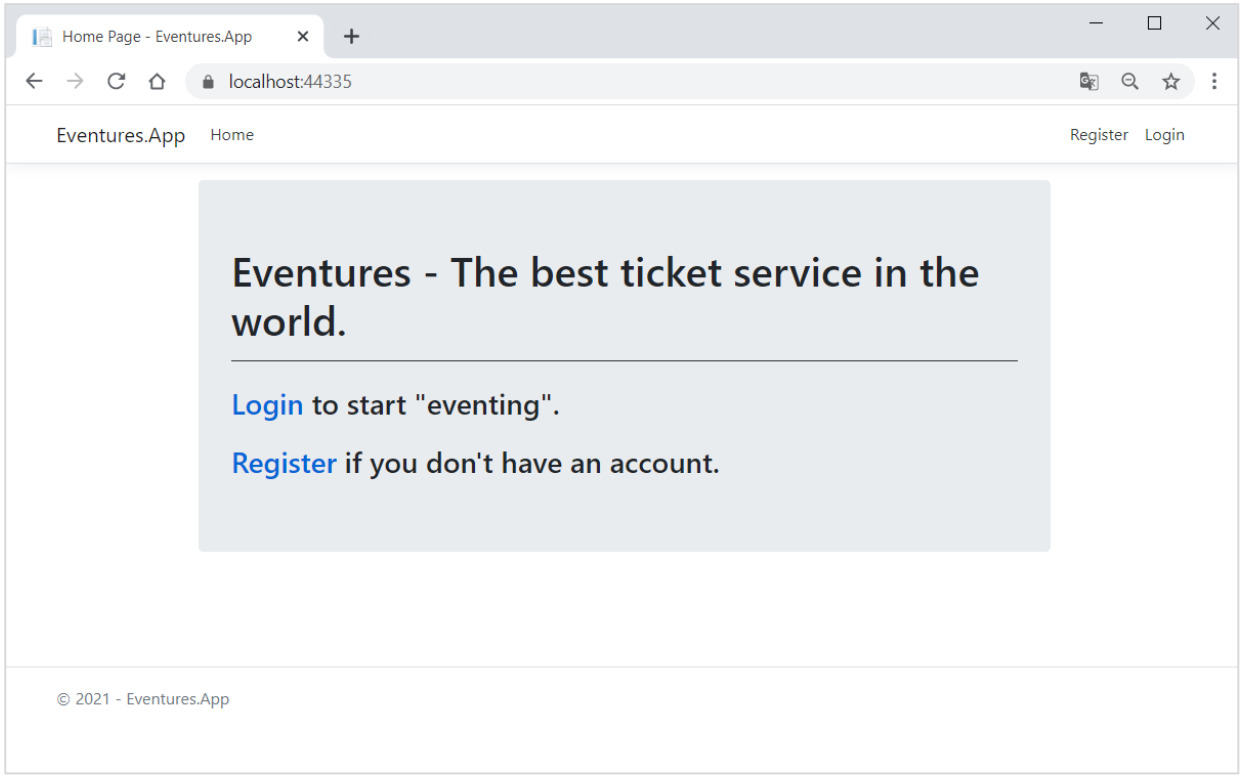
```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    0 references
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    0 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public IActionResult Privacy()
    {
        return View();
    }
```

`Privacy page` doesn't exist anymore. **Run** the app and look at the changed **navigation bar** and **footer**:



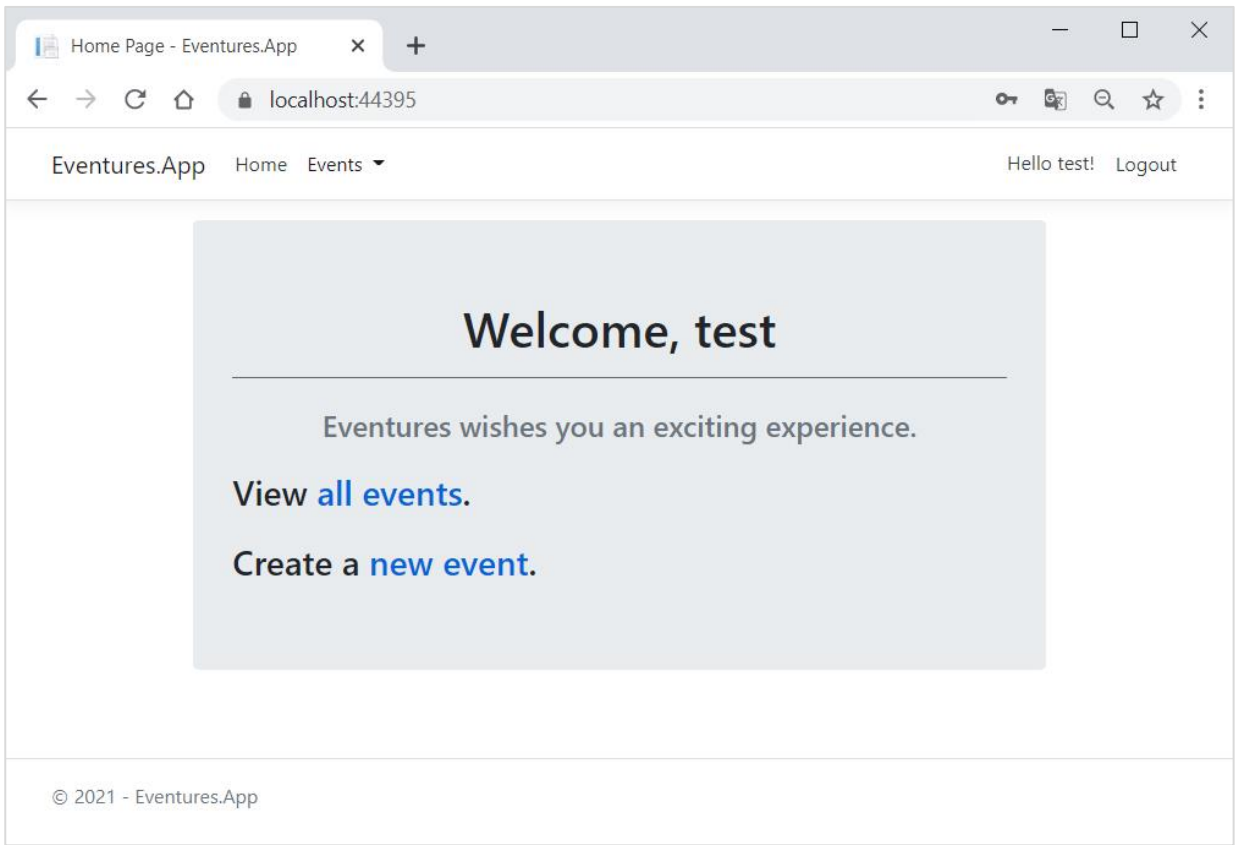
11. Change Home Page

We want to change our **`Home page`** a little bit so that it looks better and its content is suitable to our app. **Home page** for **not logged-in users** should look like this:



Note that **`[Login]`** and **`[Register]`** are **links** to Login and Register pages.

`Home page` for **logged-in** users should dynamically include our user's **name** and look like this:



To change **`Home page`**, go to **`Index.cshtml`** file in **`Views -> Home`** and change its code to be the following:

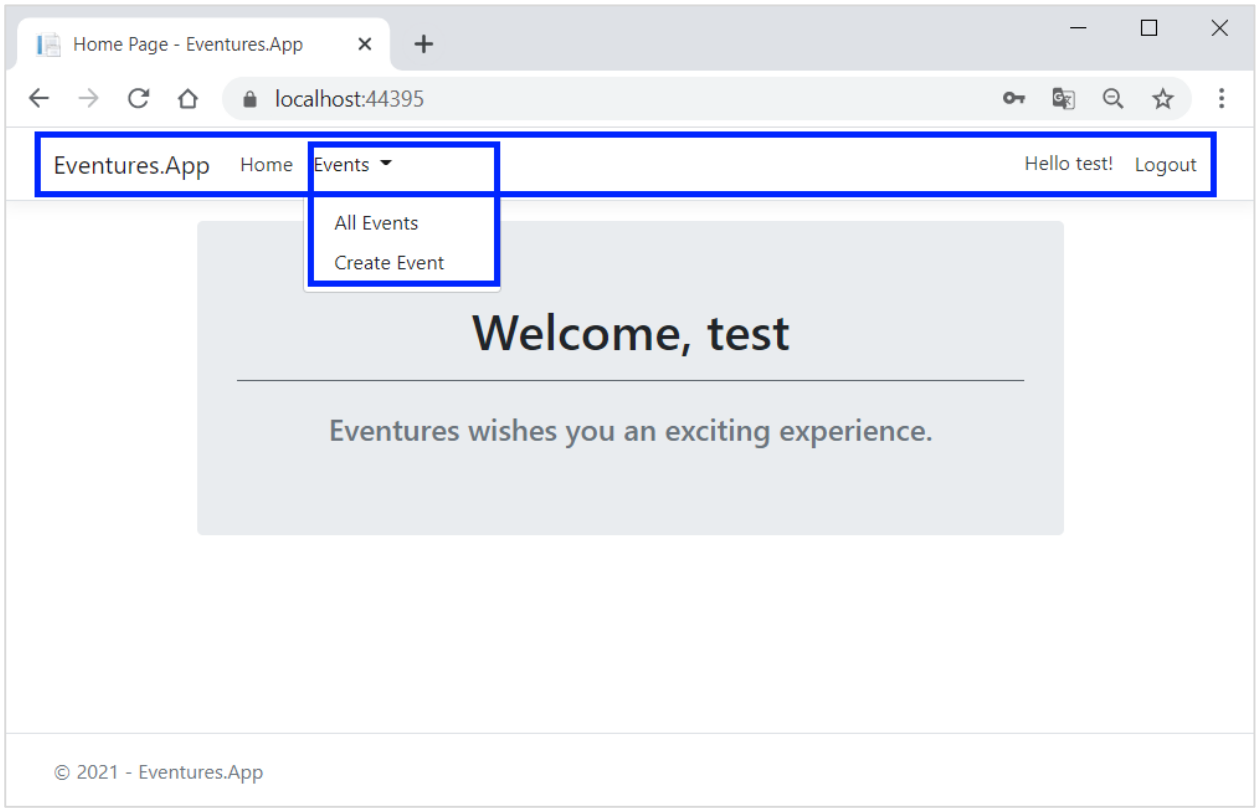
```
@{
    ViewData["Title"] = "Home Page";
}

@if (!this.User.Identity.IsAuthenticated)
{
    <div class="jumbotron bg-eventures w-75 mx-auto">
        <h1>Eventures: Events and Tickets</h1>
        <hr class="hr-2 bg-dark" />
        <h3 class="mt-4"><a href="/Identity/Account/Login">Login</a> to start "eventing".</h3>
        <h3 class="mt-4"><a href="/Identity/Account/Register">Register</a> if you don't have an
account.</h3>
    </div>
}
else
{
    <div class="jumbotron bg-eventures w-75 mx-auto">
        <h1 class="text-center">Welcome, @this.User.Identity.Name</h1>
        <hr class="hr-2 bg-secondary" />
        <h4 class="mt-4 text-secondary text-center">Eventures wishes you an exciting
experience.</h4>
        <h3 class="mt-4">View <a href="/Events/All">all events</a>.</h3>
        <h3 class="mt-4">Create a <a href="/Events/Create">new event</a>.</h3>
    </div>
}
```

Note that we use **Razor** again to add **C#** code to **HTML** and check whether the user is **logged-in** and to **dynamically** use their **name** in a **<h1>** tag. Also, we put **[Login]** and **[Register]** links using **<a>** tags.

Run the app and see if the **Home page** for not logged-in and for logged-in user is the same as on the pictures above. Test **[Login]** and **[Register]** **links**.

12. Change Navigation Bar

Now, our task is to **change** the **Navigation Bar** for **logged-in** users the following way:



Obviously, what we need to do is add an **Events dropdown menu** with **links** to **[All Events]** and **[Create Event]** pages that we will create later.

First, go to **_Layout.cshtml.cs** and **add** the following code to the **<ul class="navbar-nav flex-grow-1">** tag.

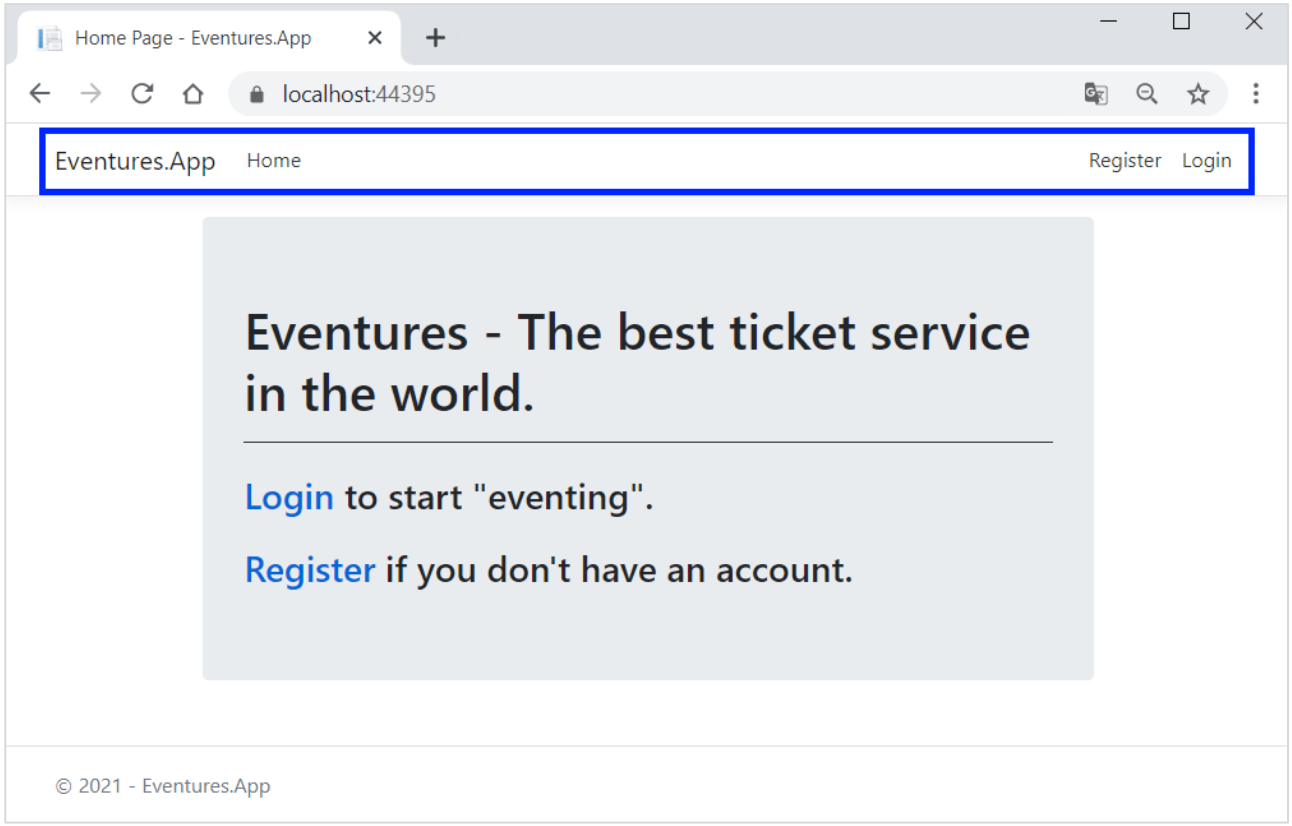
```
@if (this.User.Identity.IsAuthenticated)
{
    <li class="nav-item active">
        <div class="dropdown show">
            <a class="nav-link active dropdown-toggle" href="#"
               id="dropdownMenuLink"
               data-toggle="dropdown"
               aria-haspopup="true"
               aria-expanded="false">
                Events
            </a>

            <div class="dropdown-menu" aria-labelledby="dropdownMenuLink">
                <a class="dropdown-item" href="/Events/All">All Events</a>
                <a class="dropdown-item" href="/Events/Create">Create Event</a>
            </div>
        </div>
    </li>
}
```

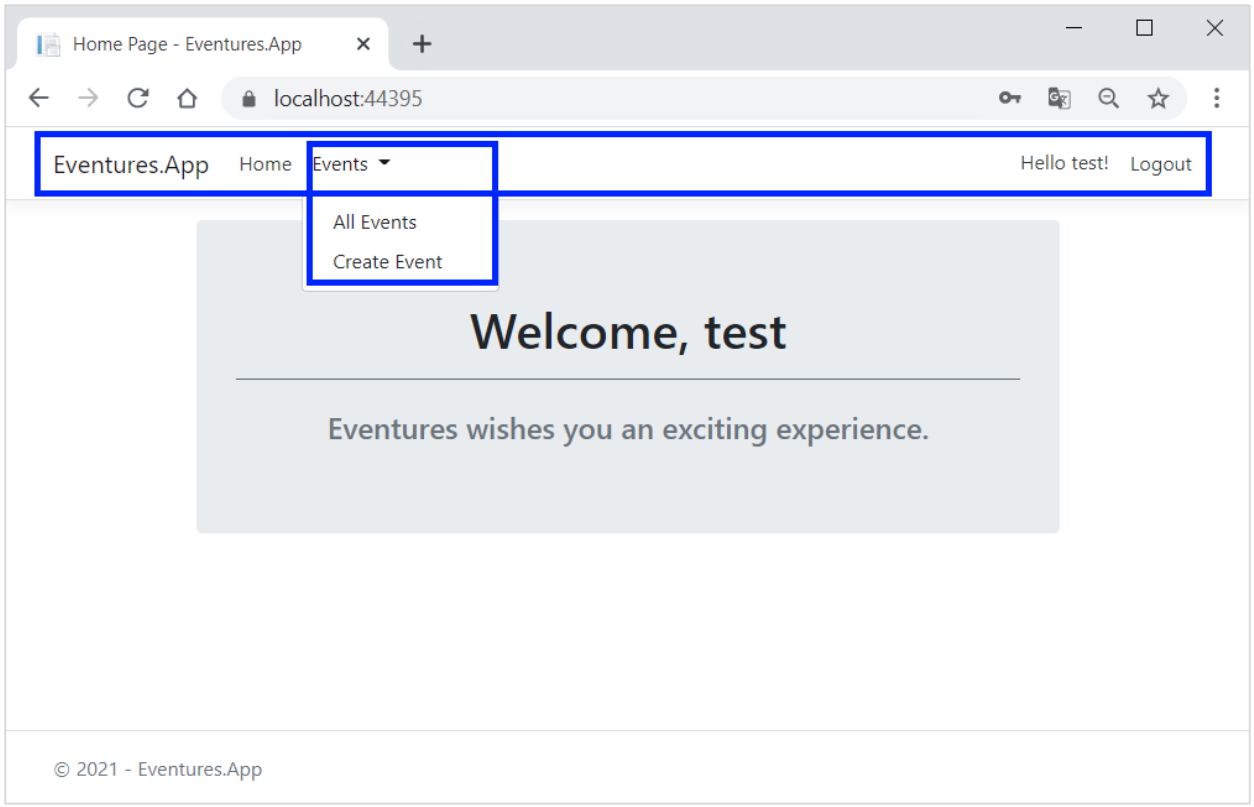
Note that, we use **Razor** to add **C# code** to our **HTML**. We use **@if** to check whether user is **authenticated**, e.g. **logged-in**, as logged-out users should not see the Events dropdown.

With the above code added, you should have the following **code structure**:

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
    <partial name="_LoginPartial" />
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        @if (this.User.Identity.IsAuthenticated)
        {
            <li class="nav-item active">
                <div class="dropdown show">
                    <a class="nav-link active dropdown-toggle" href="#"
                       id="dropdownMenuLink"
                       data-toggle="dropdown"
                       aria-haspopup="true"
                       aria-expanded="false">
                        Events
                    </a>

                    <div class="dropdown-menu" aria-labelledby="dropdownMenuLink">
                        <a class="dropdown-item" href="/Events/All">All Events</a>
                        <a class="dropdown-item" href="/Events/Create">Create Event</a>
                    </div>
                </div>
            </li>
        }
    </ul>
</div>
```

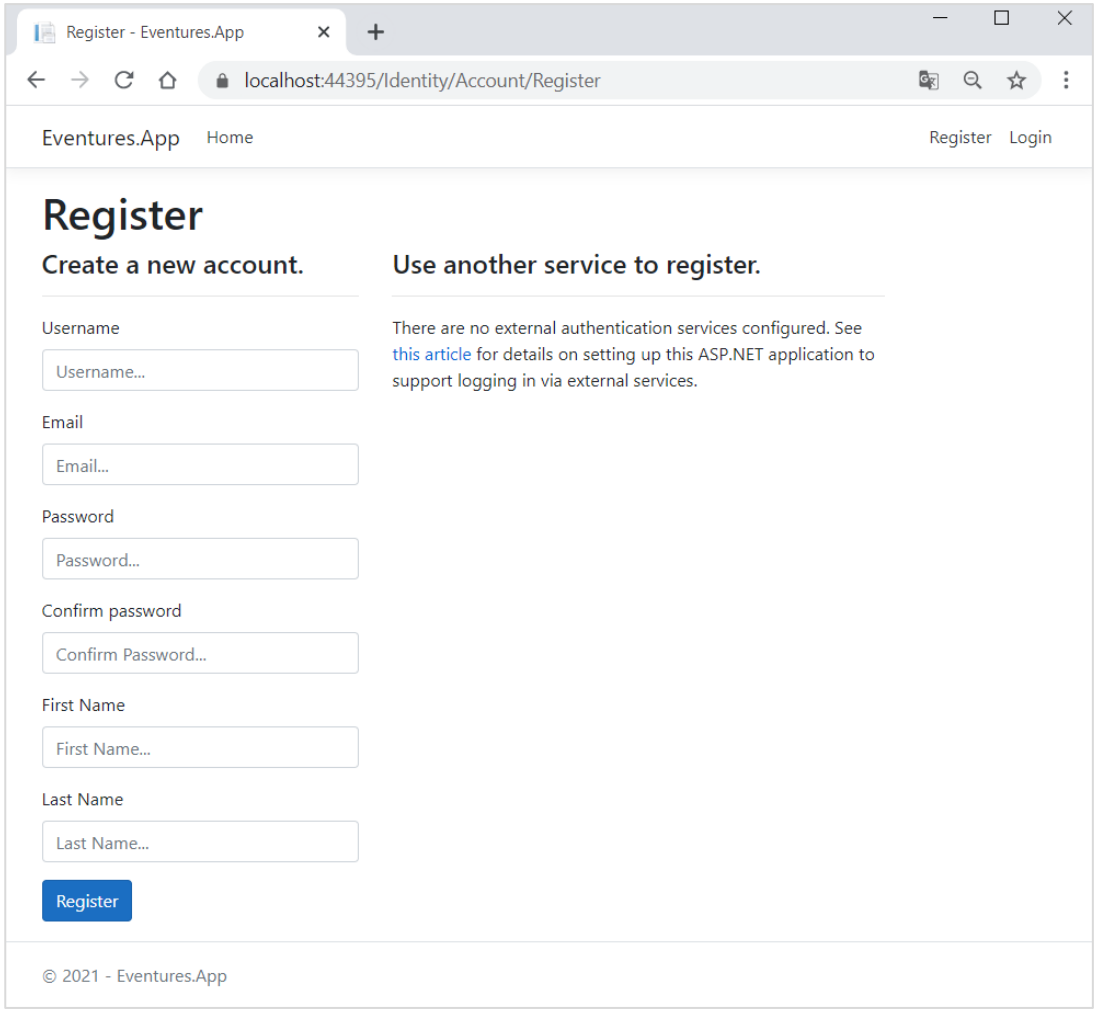
Run the app and **log in** with your user. You should see the **Events dropdown menu** in the **Navigation Bar** as on the picture above.





13. Change Registration

Our **`Registration page`** should look like this:



Modify EventuresUser

As we use our custom **`EventuresUser`**, we should **change** it. It should have additional **properties** because it doesn't inherit them from the **`IdentityUser`** class. It should look like this:

```
namespace Eventures.App.Domain
{
    16 references
    public class EventuresUser : IdentityUser
    {
        1 reference
        public string FirstName { get; set; }
        1 reference
        public string LastName { get; set; }
    }
}
```

Modify Register.cshtml.cs

Go to **`Register.cshtml.cs`** in **`Areas -> Identity -> Pages -> Account -> Register.cshtml`** and add necessary **properties** to **`InputModel`** class, as well. We are not going to add special attributes to properties, so properties should be the following:

```
public class InputModel
{
    [Required]
    [Display(Name = "Username")]
    4 references
    public string Username { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    5 references
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage =
        "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    4 references
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    3 references
    public string ConfirmPassword { get; set; }

    [Required]
    [Display(Name = "First Name")]
    4 references
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Last Name")]
    4 references
    public string LastName { get; set; }
}
```

Then, scroll down and make changes to **`user`** variable as shown below:

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = new EventuresUser
        {
            UserName = Input.Username,
            Email = Input.Email,
            FirstName = Input.FirstName,
            LastName = Input.LastName
        };
```

Modify Register.cshtml

Finally, we should make modifications to our **`Register.cshtml`**, so that new **input fields** are displayed on the page. Do it like this:

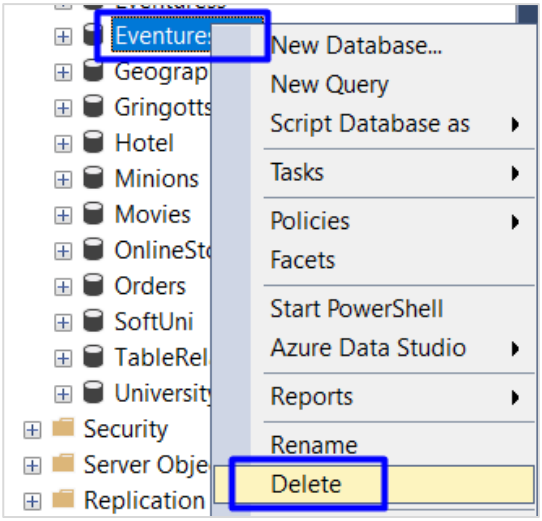
```
<div class="row">
    <div class="col-md-4">
        <form asp-route-returnUrl="@Model.ReturnUrl" method="post">
            <h4>Create a new account.</h4>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Input.Username"></label>
                <input asp-for="Input.Username" class="form-control" placeholder="Username..." />
                <span asp-validation-for="Input.Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.Email"></label>
                <input asp-for="Input.Email" class="form-control" placeholder="Email..." />
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.Password"></label>
                <input asp-for="Input.Password" class="form-control" placeholder="Password..." />
                <span asp-validation-for="Input.Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.ConfirmPassword"></label>
                <input asp-for="Input.ConfirmPassword" class="form-control" placeholder="Confirm Password..." />
                <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.FirstName"></label>
                <input asp-for="Input.FirstName" class="form-control" placeholder="First Name..." />
                <span asp-validation-for="Input.FirstName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.LastName"></label>
                <input asp-for="Input.LastName" class="form-control" placeholder="Last Name..." />
                <span asp-validation-for="Input.LastName" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-primary">Register</button>
        </form>
    </div>
```

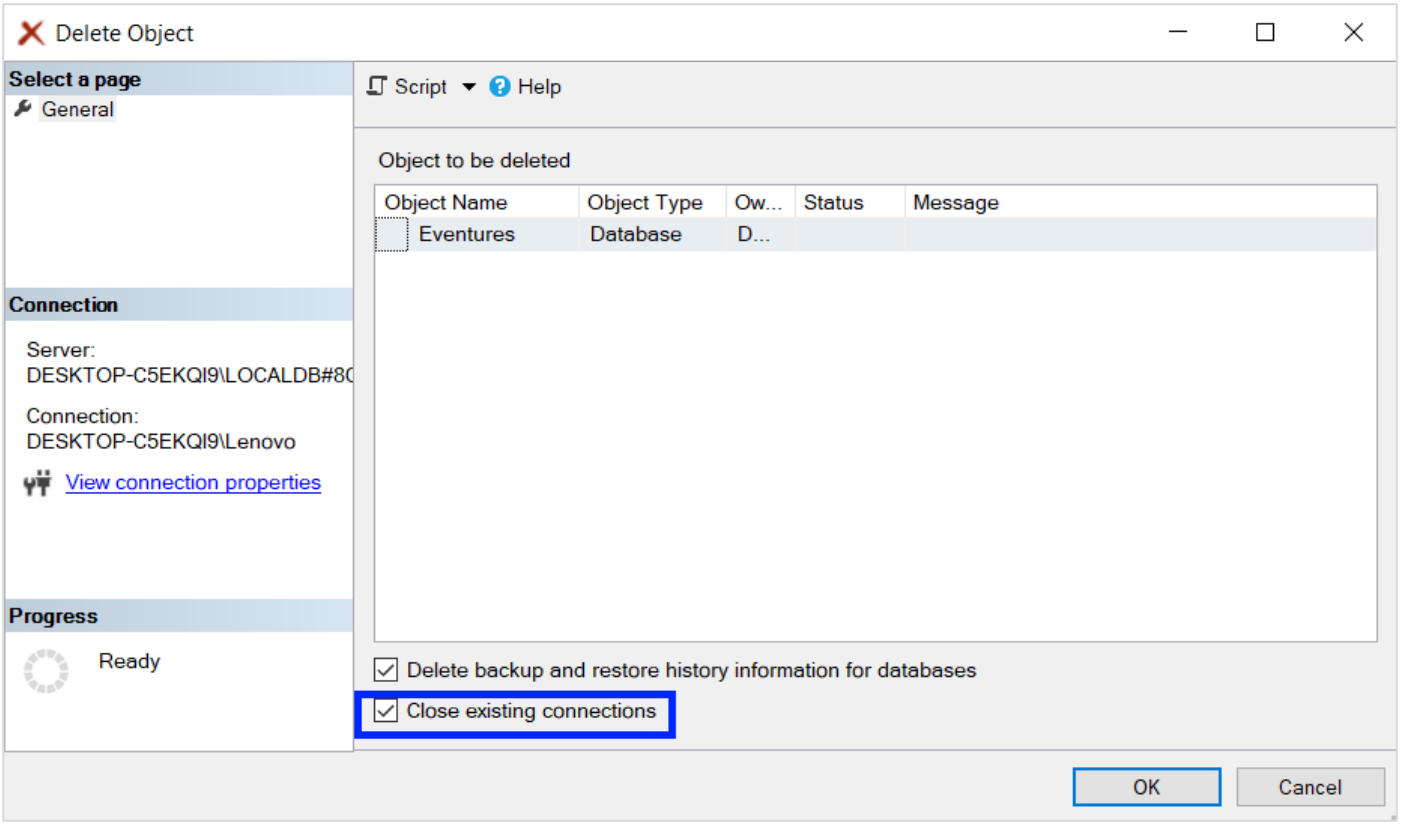
Note that we added **placeholders** to display what is to be added to the input field. It is not obligatory, but it is a good design idea. Placeholders should suit the field. This is the **placeholder** for the **Username** field:

```
<div class="form-group">
    <label asp-for="Input.Username"></label>
    <input asp-for="Input.Username" class="form-control" placeholder="Username..." />
    <span asp-validation-for="Input.Username" class="text-danger"></span>
</div>
```

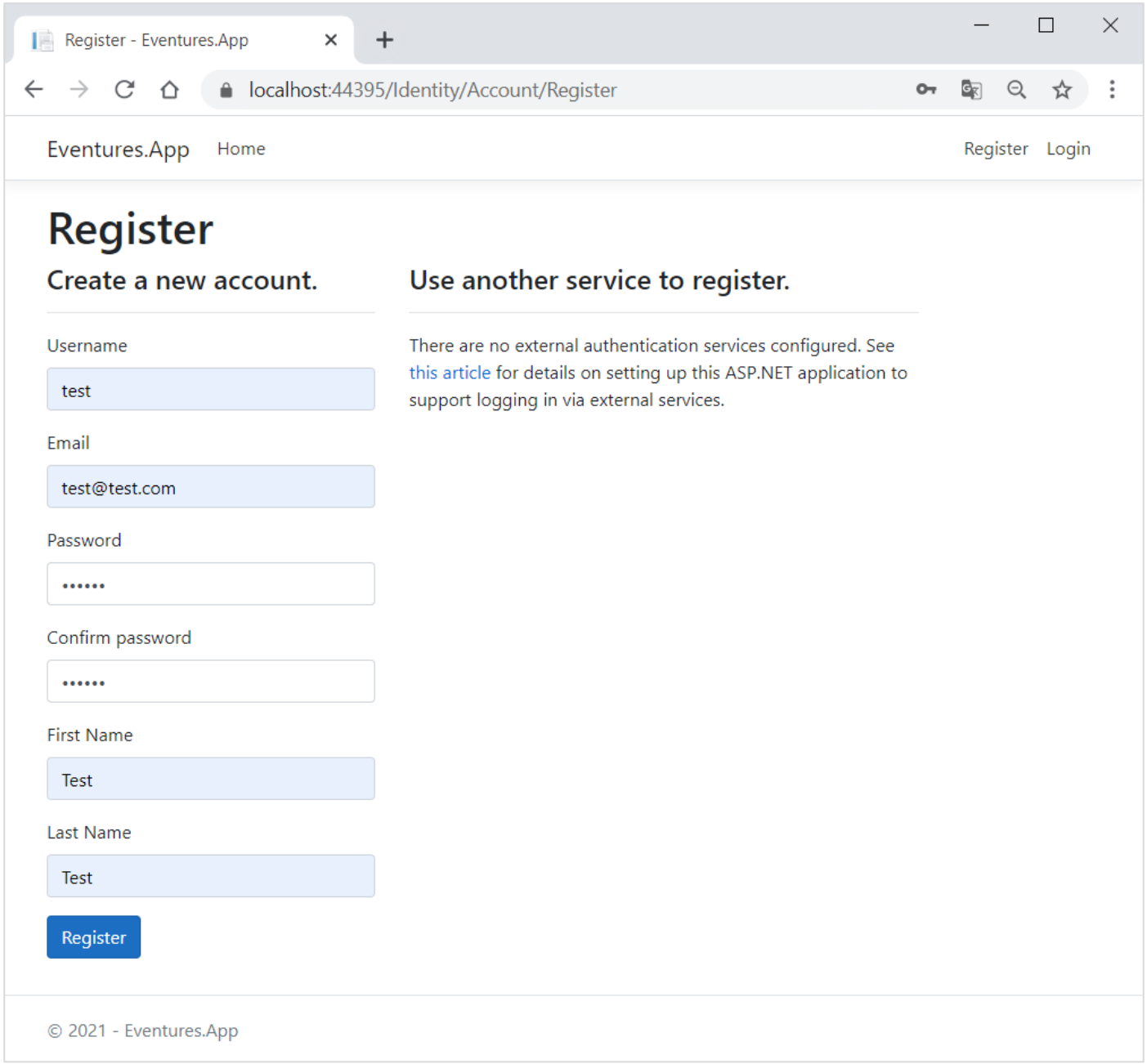
Delete the Database

Now, as we changed the **input model**, it will create a **conflict** with our current **database**. To prevent it, **delete** the **Eventures** database in **SSMS**. Do not forget to tick the **[Close existing connections]** box. When we **run** our app, the database will be created again, but our users data will be lost. **Delete** the database the following way:





Check result by **running** the app. Your **registration form** should be the same as on the picture below. Register a new user.



Your registration should be **successful**.

Check the Database

Check the database, as well. It should have changed its **`[dbo].[AspNetUsers] table`**, as we added new **properties** to **`EventuresUser`** class and it should contain our **new user**. **Right-click** on the table and choose **`[Select Top 1000 Rows]`**. You should see the following result:

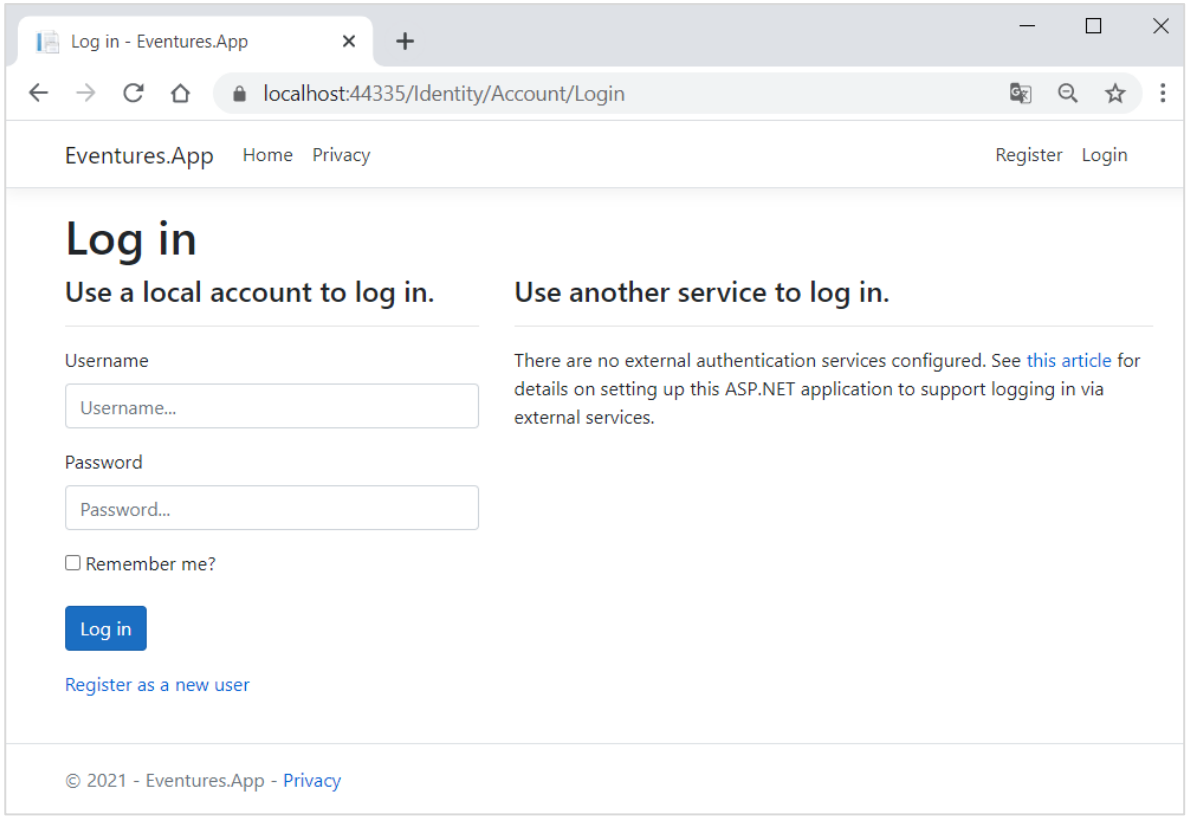
	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed
1	76c5b24a-...	test	TEST	test@test.com	TEST@TEST.COM	0

There should also have **`FirstName`** and **`LastName`** cells at the end:

FirstName	LastName
Test	Test

14. Change Log In

`Login page` should look like this:



As we can see on the picture, we want our **`Login`** form to accept **`Username`** and **`Password`**, instead of **`Email`** and **`Password`**. To change that, go to **`Login.cshtml.cs`** and change **`Email`** to **`Username`** like shown below:

```
public class InputModel
{
    [Required]
    0 references
    public string Username { get; set; }

    [Required]
    [DataType(DataType.Password)]
    4 references
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    5 references
    public bool RememberMe { get; set; }
}
```

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");

    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

    if (ModelState.IsValid)
    {
        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout, set lockoutOnFailure: true
        var result = await _signInManager.PasswordSignInAsync(Input.Username, Input.Password
            , Input.RememberMe, lockoutOnFailure: false);
        if (result.Succeeded)
        {
            _logger.LogInformation("User logged in.");
            return LocalRedirect(returnUrl);
        }
```

Then, change **`Email`** to **`Username`** in the **`Login.cshtml file`**, as well. It is a good idea to add **placeholders**, too.

```
<h4>Use a local account to log in.</h4>
<hr />
<div asp-validation-summary="All" class="text-danger"></div>
<div class="form-group">
    <label asp-for="Input.Username"></label>
    <input asp-for="Input.Username" class="form-control" placeholder="Username..." />
    <span asp-validation-for="Input.Username" class="text-danger"></span>
</div>
<div class="form-group">
    <label asp-for="Input.Password"></label>
    <input asp-for="Input.Password" class="form-control" placeholder="Password..." />
    <span asp-validation-for="Input.Password" class="text-danger"></span>
</div>
```

In addition, we should better remove "**`Forgot your password?`**" and "**`Resend email confirmation`**" **links**, as we are not going to set up their functionalities. You just need to **delete** them:

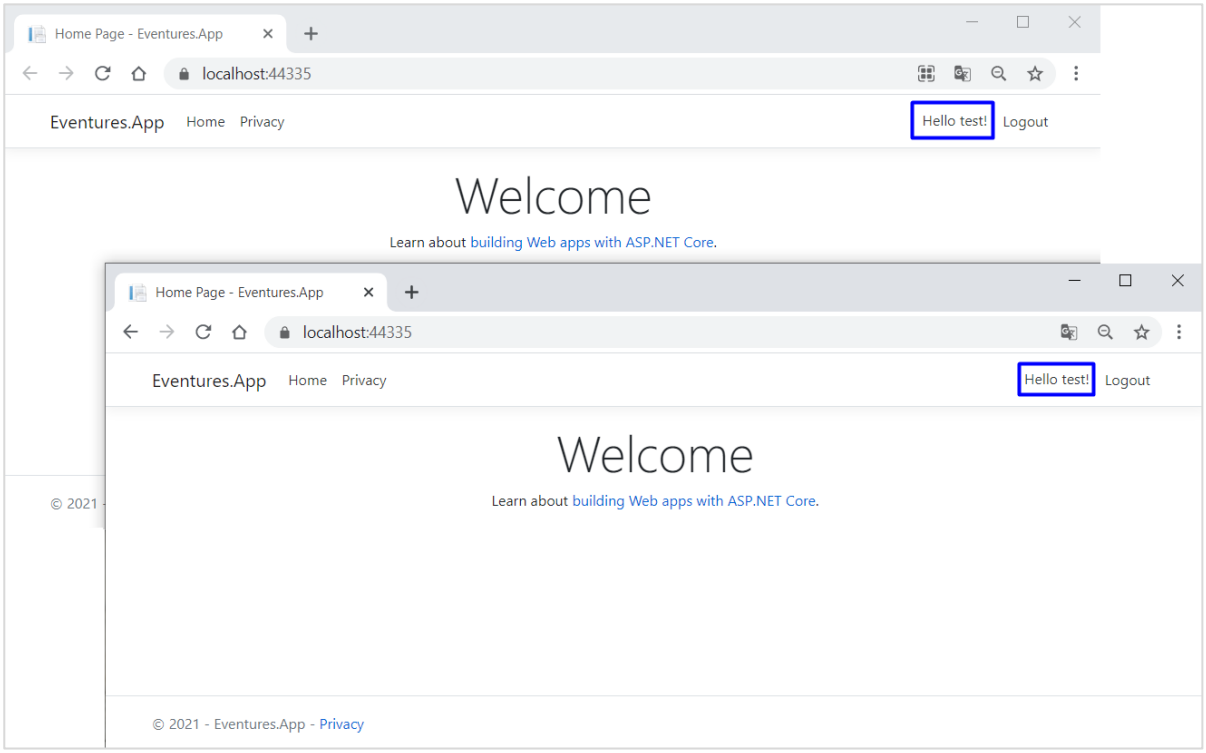
```
<div class="form-group">
    <p>
        <a id="forgot-password" asp-page="./ForgotPassword">Forgot your password?</a>
    </p>
    <p>
        <a asp-page="./Register" asp-route-returnUrl="@Model.ReturnUrl">Register as a new user</a>
    </p>
    <p>
        <a id="resend-confirmation" asp-page="./ResendEmailConfirmation">Resend email confirmation</a>
    </p>
</div>
```

When both links are **deleted**, the only one left in the **`<div>`** tag should be the following:

```
<div class="form-group">
    <p>
        <a asp-page="./Register" asp-route-returnUrl="@Model.ReturnUrl">Register as a new user</a>
    </p>
</div>
```

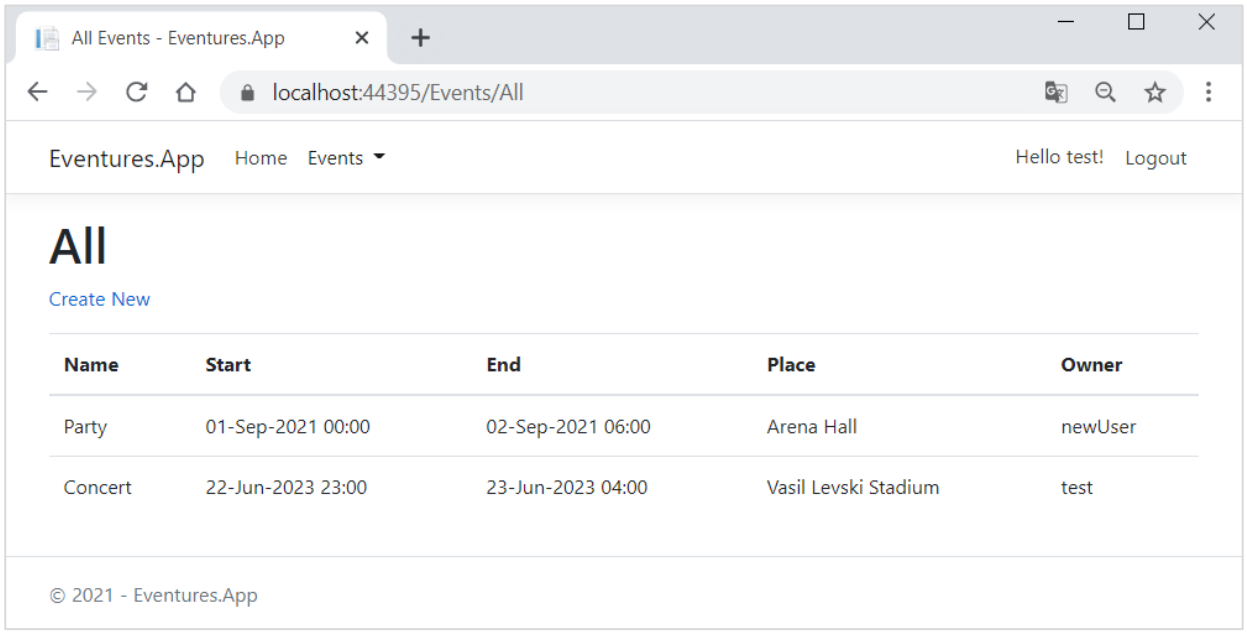
Now **run the app** and **test login functionality** by logging-in with your user. The login should be **successful** and your page should look like the one on the picture above.

a. You can also **test** the **`[Register as a new user]`** link- it should redirect you to the **`Registration page`**.
b. Test the **`Remember me`** functionality by putting a **tick** on the **`[Remember me]`** checkbox when logging-in. Then close the current window and open a new one. Your user should be logged-in.

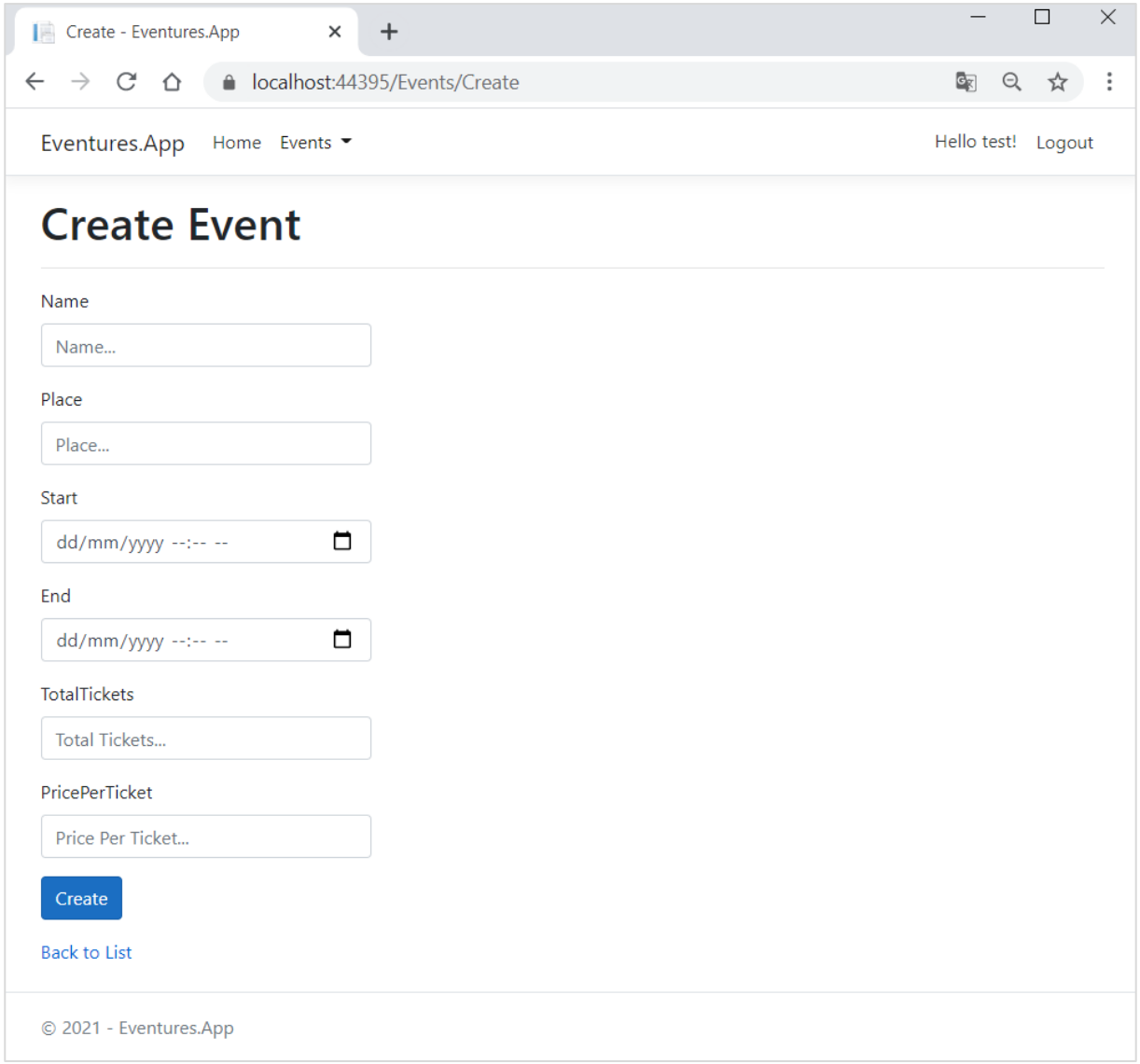


Events Setup

At the end, after we make changes, the **`All Events`** page should be the following:



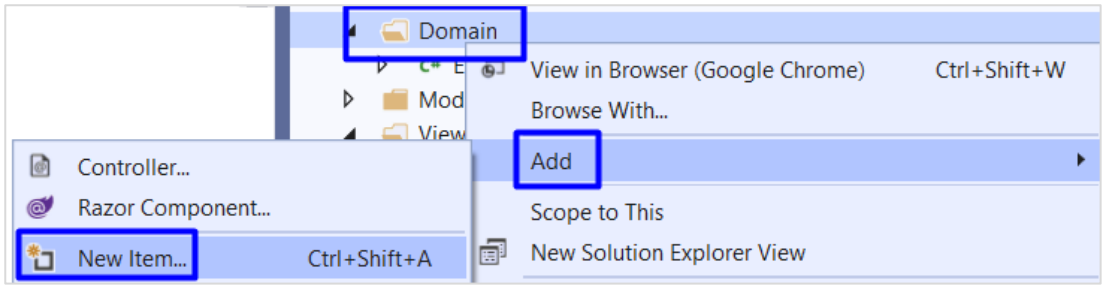
Also, our **`Create Event page`** should look like this:

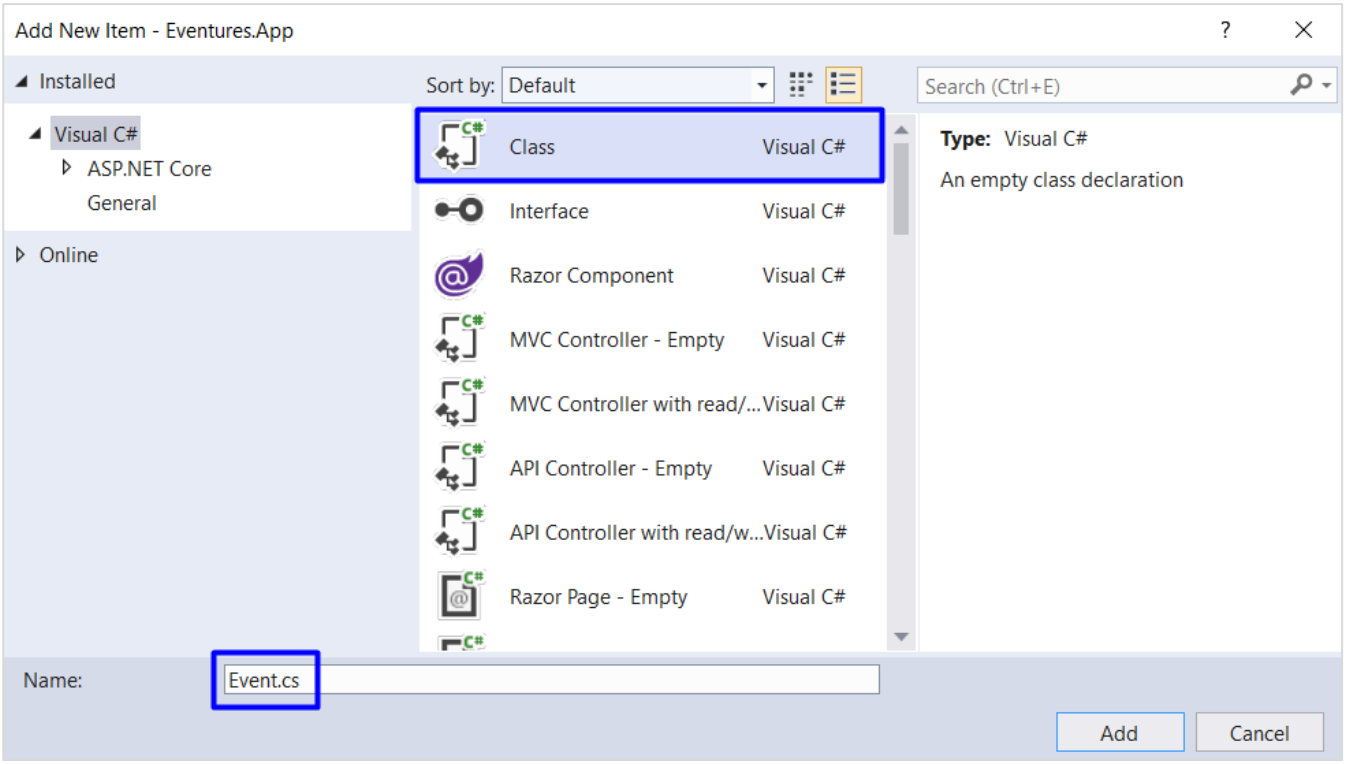


To **create** these **pages**, we will need to create an **`Event class`**, a **`controller`**, **`models`** and **`views`**. We should also make a modification to the **`context`**.

15. Create Event Class

`Events` are the main part of our **`Eventures`** app. To create an **event**, we will need to have an **`Event class`** to hold our **`Event properties`**. Create **`Events`** class in **`Domain`** folder, as shown below. Name it **`Events`**.





Our **Event** class has the following properties:

- **Id** – a **UUID (Universally unique identifier)** in the DB – a **string**.
- **Name** – a **string**.
- **Place** – a **string**.
- **Start** – a **DateTime** object.
- **End** – a **DateTime** object.
- **Total Tickets** – an **integer**.
- **Price Per Ticket** – a **decimal** value.
- **Owner** – an **EventuresUser** object.
- **OwnerId** – a **string**.

Make the **Events** class **public**. Add **properties** to it and make it look like this:

```
public class Event
{
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public string Id { get; set; }

    2 references
    public string Name { get; set; }

    2 references
    public string Place { get; set; }

    2 references
    public DateTime Start { get; set; }

    2 references
    public DateTime End { get; set; }

    1 reference
    public int TotalTickets { get; set; }

    [Column(TypeName = "decimal(12,3)")]
    1 reference
    public decimal PricePerTicket { get; set; }
    1 reference
    public EventuresUser Owner { get; set; }
    1 reference
    public string OwnerId { get; set; }
}
```

Note that we added a special **attribute** to the **Id** property, so that **Id** is generated by our **Eventures** database.

Also, it is important that **Owner** is of type **EventuresUser**- this is so that we have a connection between the **Event** and its **Owner**.

16. Change Context

We already created an **Event** class and our context should have a collection of events to be stored in the **database**. It is a **good practice** to create a special **EventsDbContext**, but we will work with the default context to keep things simple. Let's go to **ApplicationDbContext.cs** and add a **collection** of **Event**:

```
namespace Eventures.App.Data
{
    6 references
    public class ApplicationDbContext : IdentityDbContext<EventuresUser>
    {
        0 references
        public DbSet<Event> Events { get; set; }
        0 references
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
            this.Database.EnsureCreated();
        }
    }
}
```

Note that we use the **DbSet<>** class because a **DbSet** represents the **collection** of **all entities** in the **context**, or that can be queried from the **database**.

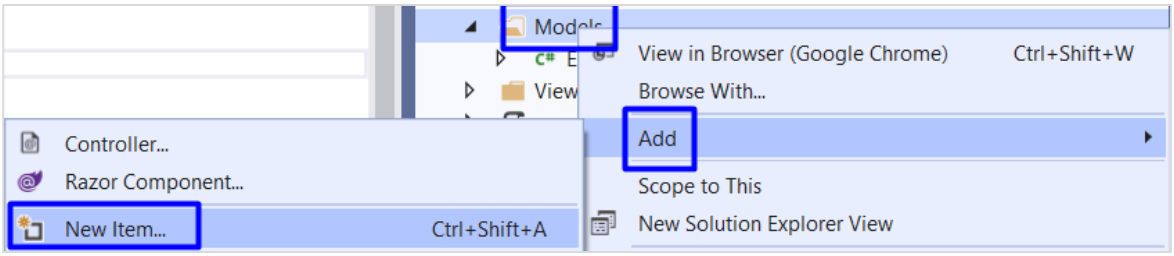
17. Create Models

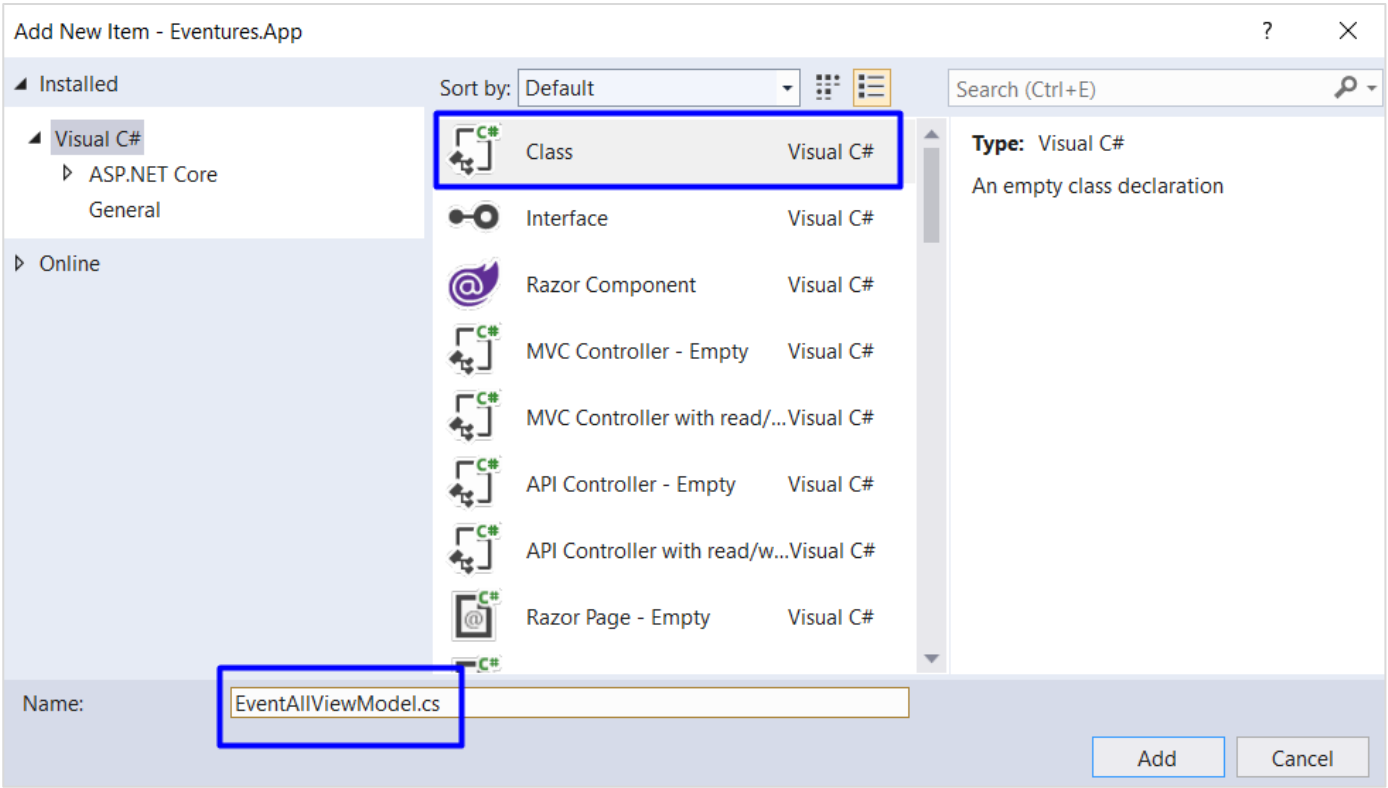
Create EventAllViewModel

For the architecture of our **`All Events`** page, we need to create a **model** with the following properties:

- **`Name`** – a **`string`**.
- **`Start`** – a **`string object`**.
- **`End`** – a **`string object`**.
- **`Place`** – a **`string`**.

First, create **`EventAllViewModel.cs`** model in **`Models`** folder:





Add the necessary **`properties`** to the class:

```
public class EventAllViewModel
{
    3 references
    public string Name { get; set; }

    3 references
    public string Start { get; set; }

    3 references
    public string End { get; set; }

    3 references
    public string Place { get; set; }
    0 references
    public string Owner { get; set; }
}
```

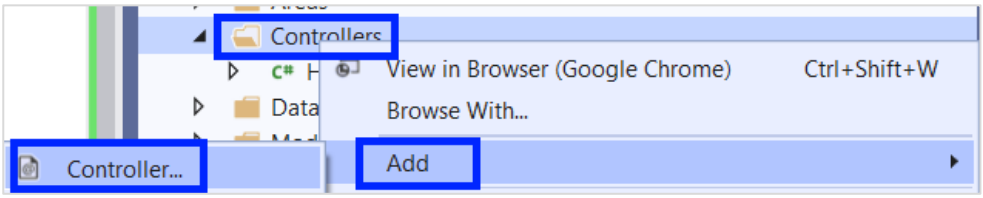
Create EventCreateBindingModel

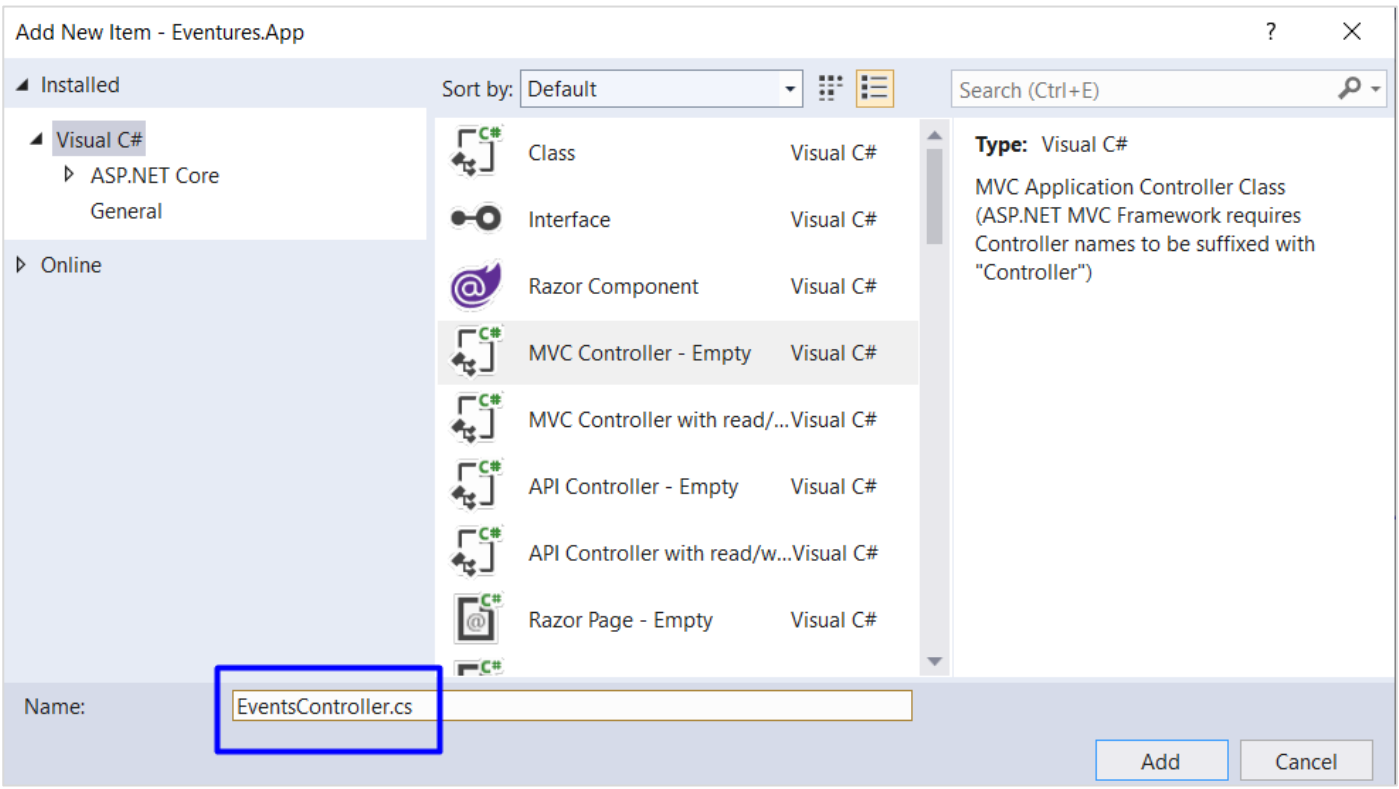
In **`Models`** folder, create one more class – **`EventCreateBindingModel.cs`** model. This will be our **`model`** for creating **`Events`**. Add the necessary **`properties`** to the class. Add **`attributes`** to properties to **restrict** input data. For example, **`TotalTickets`** and **`PricePerTicket`** should accept only **positive values**. The class should look like this:

```
namespace Eventures.App.Models
{
    0 references
    public class EventCreateBindingModel
    {
        [Required]
        [Display(Name = "Name")]
        0 references
        public string Name { get; set; }

        [Required]
        [Display(Name = "Place")]
        0 references
        public string Place { get; set; }

        [Required]
        [Display(Name = "Start")]
        0 references
        public DateTime Start { get; set; }

        [Required]
        [Display(Name = "End")]
        0 references
        public DateTime End { get; set; }

        [Required]
        [Range(0, int.MaxValue, ErrorMessage = "Total Tickets must be a positive number.")]
        [Display(Name = "TotalTickets")]
        0 references
        public int TotalTickets { get; set; }

        [Required]
        [Range(0.00, double.MaxValue, ErrorMessage = "Price Per Ticket must be a positive number.")]
        [Display(Name = "PricePerTicket")]
        0 references
        public decimal PricePerTicket { get; set; }
    }
}
```

18. Create Controller

Create an **`EventsController.cs`** to handle incoming requests to the application. Create an **`Empty MVC Controller`** in **`Controllers`** folder the following way:





First, it is important that we add the **`[Authorize]`** attribute to our **controller class**, as we don't want **unauthorized users** to be able to access our pages. **Add the attribute** like this:

```
namespace Eventures.App.Controllers
{
    [Authorize]
    1 reference
    public class EventsController : Controller
```

Then, we should create a **field** for our **context** as we are going to make changes to the context later. Also, we should create a **constructor** to accept and assign our context to the **ApplicationDbContext**:

```
namespace Eventures.App.Controllers
{
    [Authorize]
    1 reference
    public class EventsController : Controller
    {
        private readonly ApplicationDbContext context;
        0 references
        public EventsController(ApplicationDbContext context)
        {
            this.context = context;
        }
```

Next, we should create **IActionResult All()** for our **All Events** page. It should initialize a collection of **EventAllViewModel** and pass it to the corresponding **View**. The method should look like this:

```
public IActionResult All()
{
    List<EventAllViewModel> events = context.Events
        .Select(eventFromDb => new EventAllViewModel
        {
            Name = eventFromDb.Name,
            Place = eventFromDb.Place,
            Start = eventFromDb.Start.ToString("dd-MMM-yyyy HH:mm", CultureInfo.InvariantCulture),
            End = eventFromDb.End.ToString("dd-MMM-yyyy HH:mm", CultureInfo.InvariantCulture),
            Owner = eventFromDb.Owner.UserName
        })
        .ToList();

    return this.View(events);
}
```

Finally, create **IActionResult Create()** to return a **View**.

```
public IActionResult Create()
{
    return this.View();
}
```

Also, we should create **IActionResult Create(EventCreateBindingModel bindingModel)** method. In the method, create an **Event** with the **data from the binding model** and add that Event to the **context**. Do not forget to **save changes**. Then **redirect** the user to **All Events** page, if adding the Event is successful, or return a **View**. Your method should look like this:

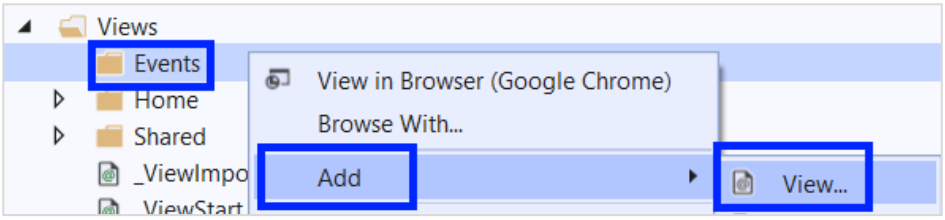
```
[HttpPost]
0 references
public IActionResult Create(EventCreateBindingModel bindingModel)
{
    if (this.ModelState.IsValid)
    {
        string currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
        Event eventForDb = new Event
        {
            Name = bindingModel.Name,
            Place = bindingModel.Place,
            Start = bindingModel.Start,
            End = bindingModel.End,
            TotalTickets = bindingModel.TotalTickets,
            PricePerTicket = bindingModel.PricePerTicket,
            OwnerId = currentUserId
        };

        context.Events.Add(eventForDb);
        context.SaveChanges();

        return this.RedirectToAction("All");
    }

    return this.View();
}
```

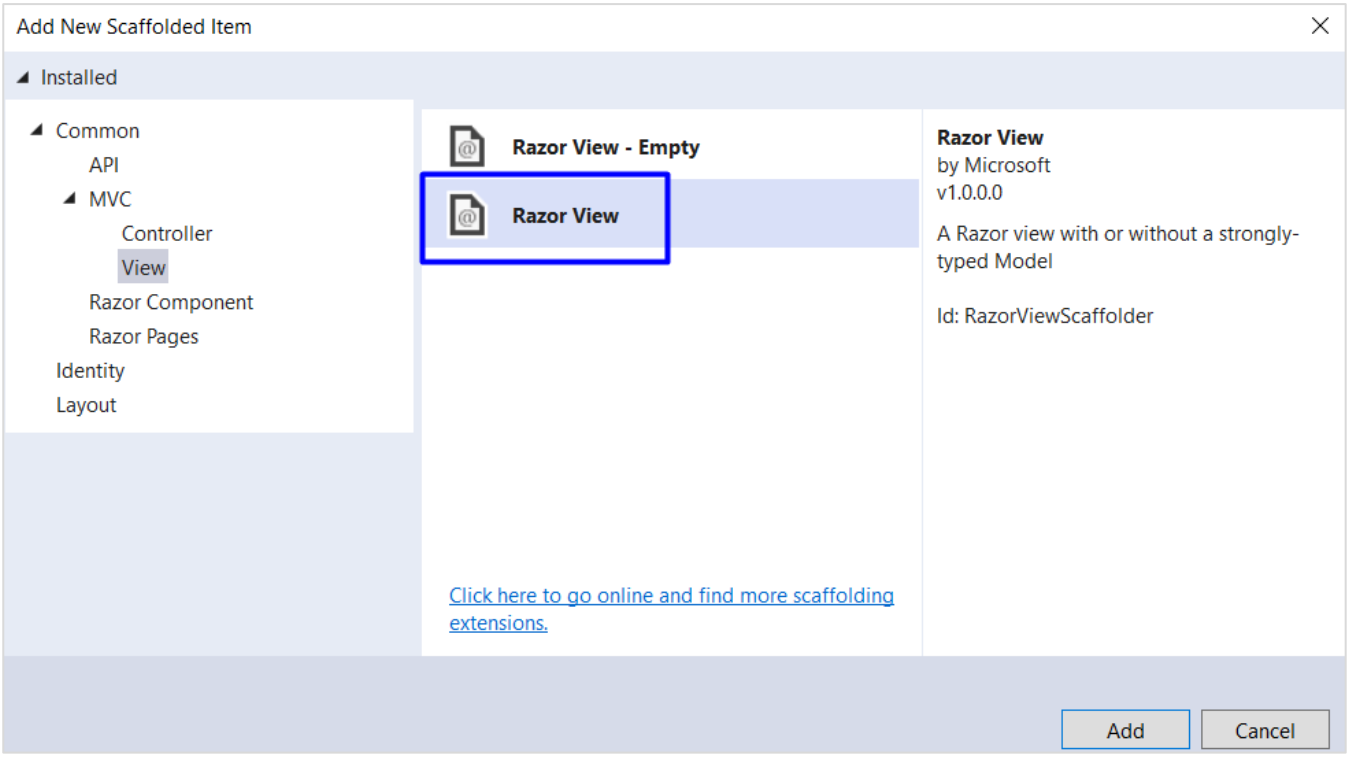
Note that we create the **currentUserId** variable to store the Id of the **user** we are **currently logged-in** with, who creates the current **Event**. This way we don't need to exclusively **point out** who the **Owner** of the **Event** is.

19. Create Views

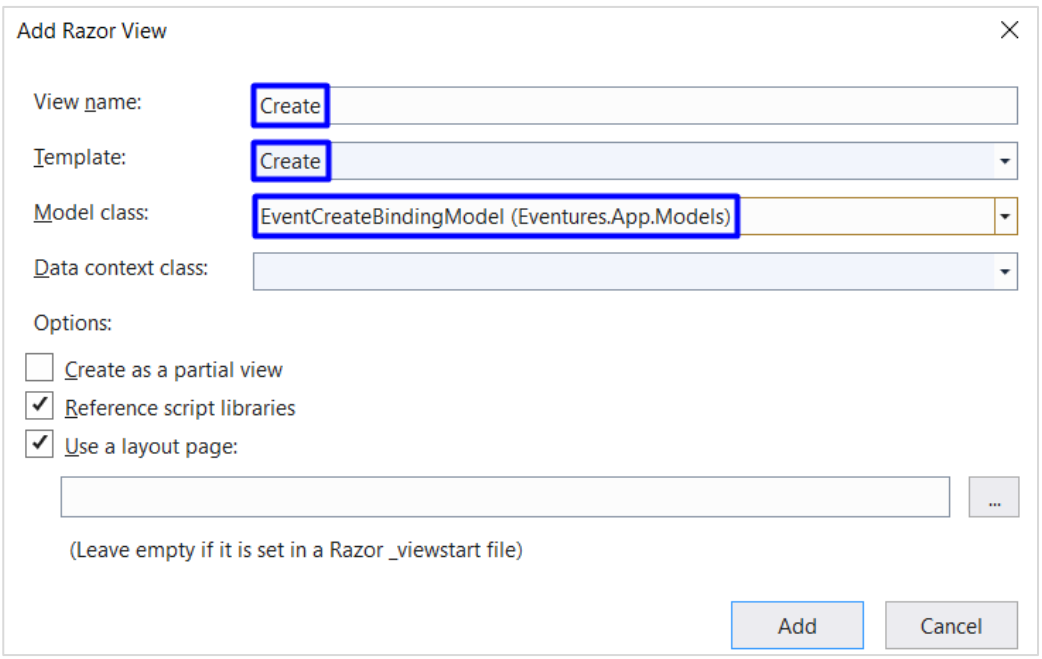
Create View

First, add a new **Events** folder to **Views** folder, where we will put our Views for the Event. Then, **right-click** on the **Events** folder and **add a Razor View** as shown below.





Name the view **`Create`**, choose **`Create`** from templates and choose **`EventCreateBindingModel`** as a model class, as shown below. Press **`[Add]`**.

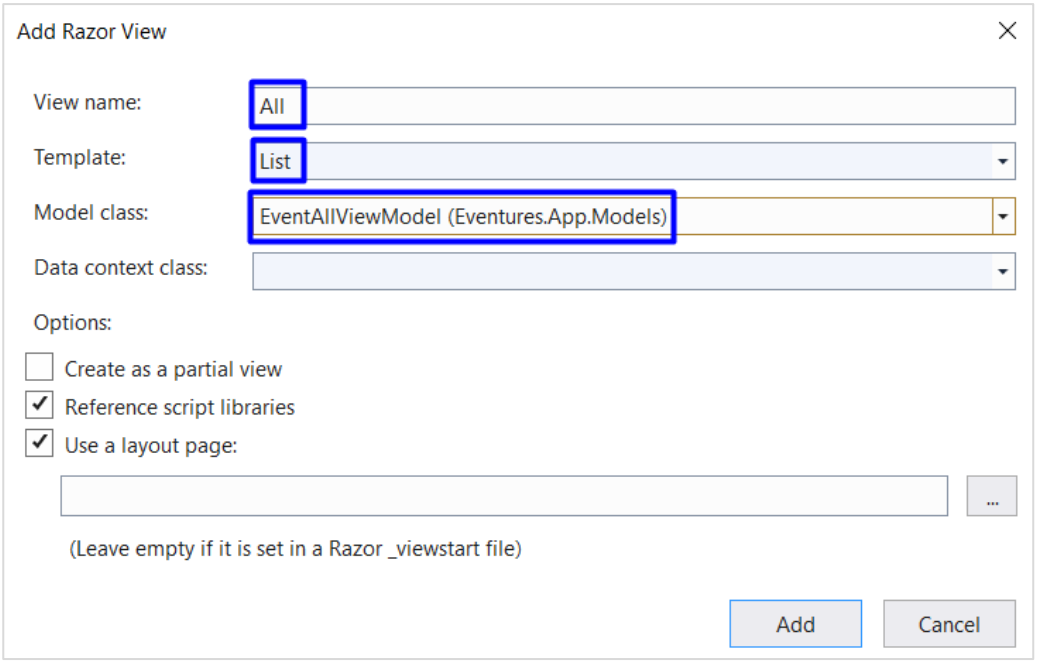


Then, you can see that your **`Create`** View is automatically **generated**. Change title in **`<h1>`** tag to **`Create Event`** and **remove** the **`<h4>`** tag as we don't need it. Add **`placeholders`**. However, we **do not need** placeholders for **`Start`** and **`End`**, as they are of type **`DateTime`** and have **default** ones. These are the changes that need to be made:

```
@model Eventures.App.Models.EventCreateBindingModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create Event</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" placeholder="Name..." />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Place" class="control-label"></label>
                <input asp-for="Place" class="form-control" placeholder="Place..." />
                <span asp-validation-for="Place" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Start" class="control-label"></label>
                <input asp-for="Start" class="form-control" />
                <span asp-validation-for="Start" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="End" class="control-label"></label>
                <input asp-for="End" class="form-control" />
                <span asp-validation-for="End" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="TotalTickets" class="control-label"></label>
                <input asp-for="TotalTickets" class="form-control" placeholder="Total Tickets..." />
                <span asp-validation-for="TotalTickets" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PricePerTicket" class="control-label"></label>
                <input asp-for="PricePerTicket" class="form-control" placeholder="Price Per Ticket..." />
                <span asp-validation-for="PricePerTicket" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

All View

Right-click on the **`Events`** folder and **add a Razor `View`** again. Name the view **`All`**, choose **`List`** from templates and choose **`EventAllViewModel`** as a model class, as shown below. Press **`[Add]`**.



After the **View** is generated, change the **title** to **All Events** and **remove** the **<td>** tag containing **ActionLinks** as we won't configure them:

```
@{
    ViewData["Title"] = "All Events";
}
```

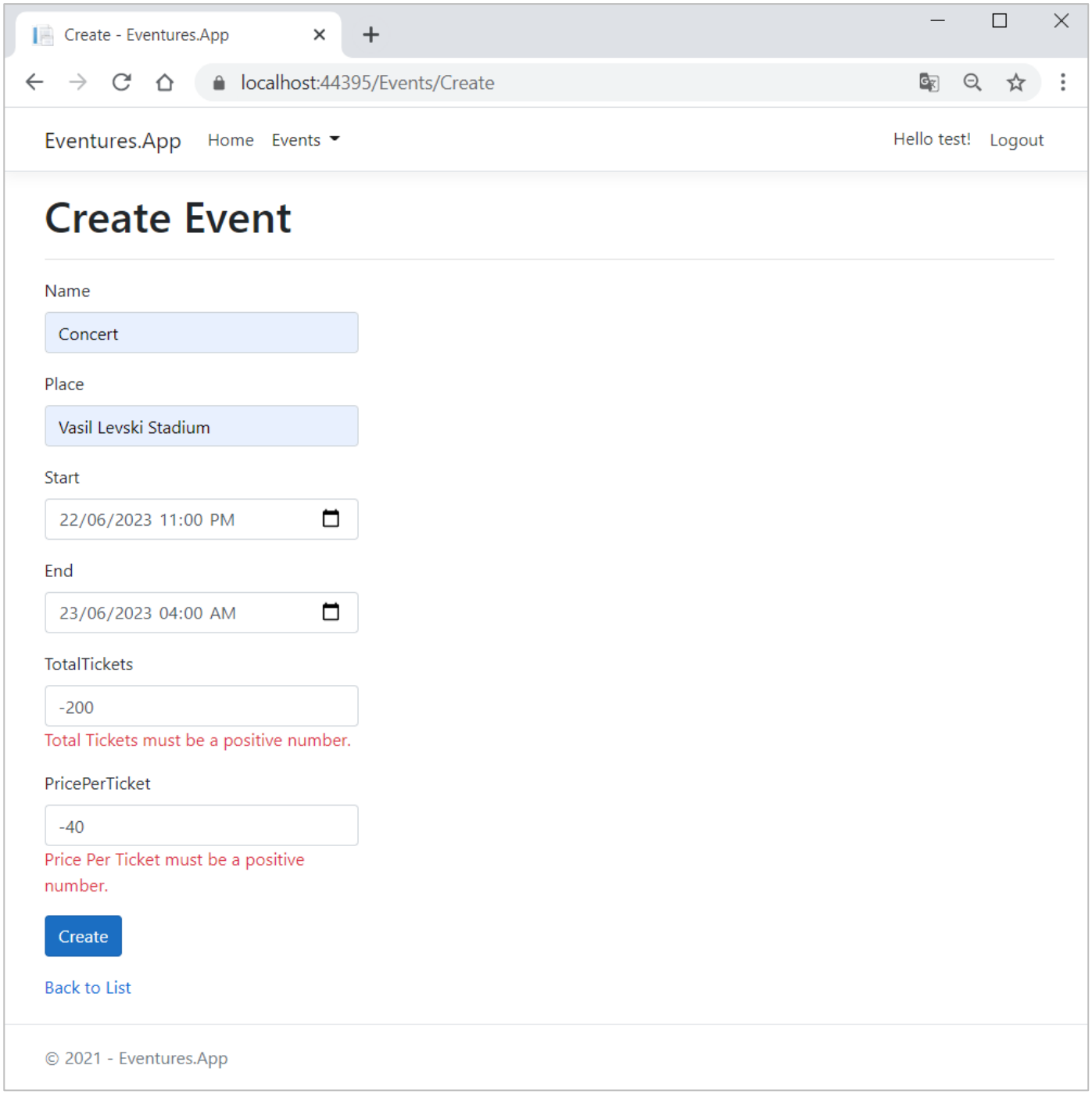
```
<td>
    @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
    @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
    @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
</td>
```

20. Test App

Before we try to create an **Event**, we should **delete** our current **database** as we made changes to it, otherwise an error will be thrown. Go to **SSMS** and delete the **Eventures** database. Do not forget to check the **[Close existing connections]** box. Then, run the app again using **[Ctrl+F5]**. After that, **register** a new user again and use it to **log in**.

Test Create Event

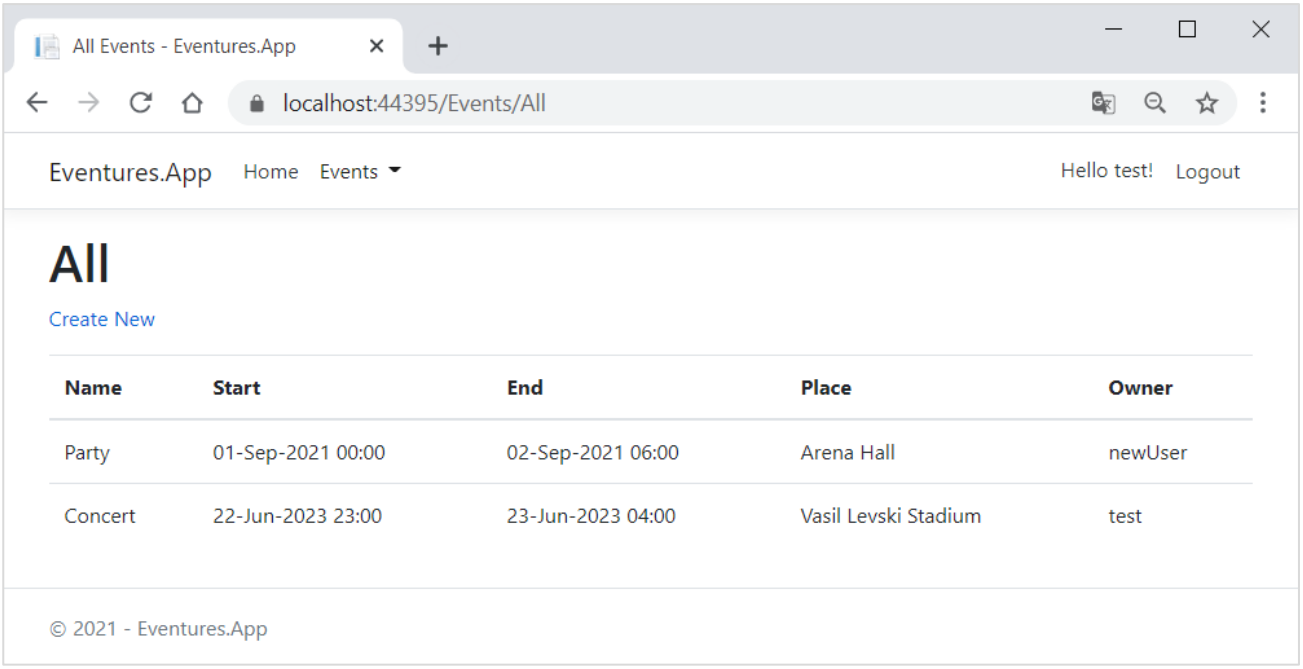
From the **Navigation Bar**, go to **Events** -> **Create Event** and **fill in data**. Try filling in **wrong data**, as shown below- an **error** messages should appear:



In case of **successful event creation**, you should be **redirected** to **`All Events`** page.

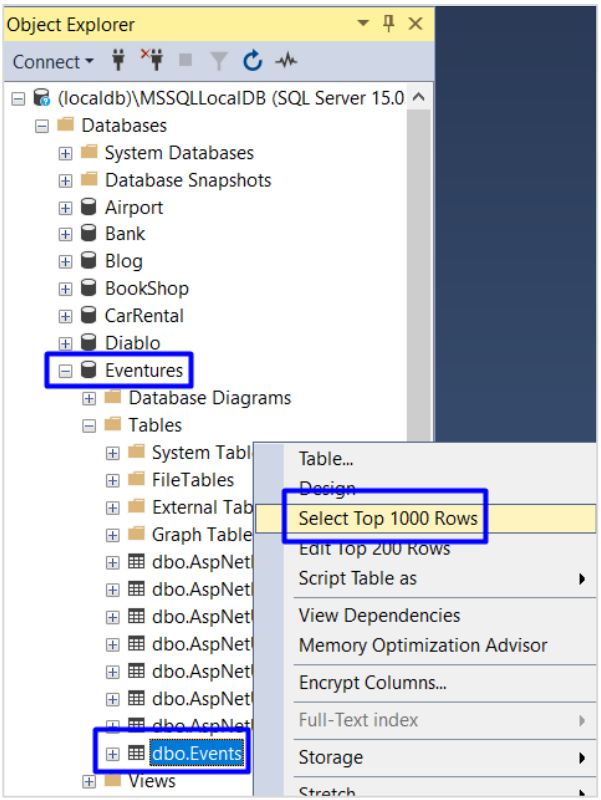
Test All Events

All **`Events`** you have created should be displayed on the **`All Events`** page. Go to **`Events -> All Events`** from the **`Navigation Bar`** and check if **`Events`** are shown. You can create more **`Users`** and **`Events`** and each of them should be displayed with its **`Owner`** and **other information** like this:



Check the Database

Go to SSMS and look **`Eventures`** database's tables. You should have **`dbo.Events`** table holding **`Events`**. Check if **`Events`** we created are present in the **database**:



Your result should be the following:

	Id	Name	Place	Start	End	TotalTickets	PricePerTicket	OwnerId
1	7ae76f...	Party	Arena Hall	2021-09-01 00...	2021-09-02 06...	30	15.000	bb7672a...
2	a1ebd...	Concert	Vasil Levski Stadium	2023-06-22 23...	2023-06-23 04...	200	40.000	d708976...

And now our **`ASP.NET MVC App "Eventures"`** is fully ready with all its **functionalities** and **pages**. You can always add more if you like.