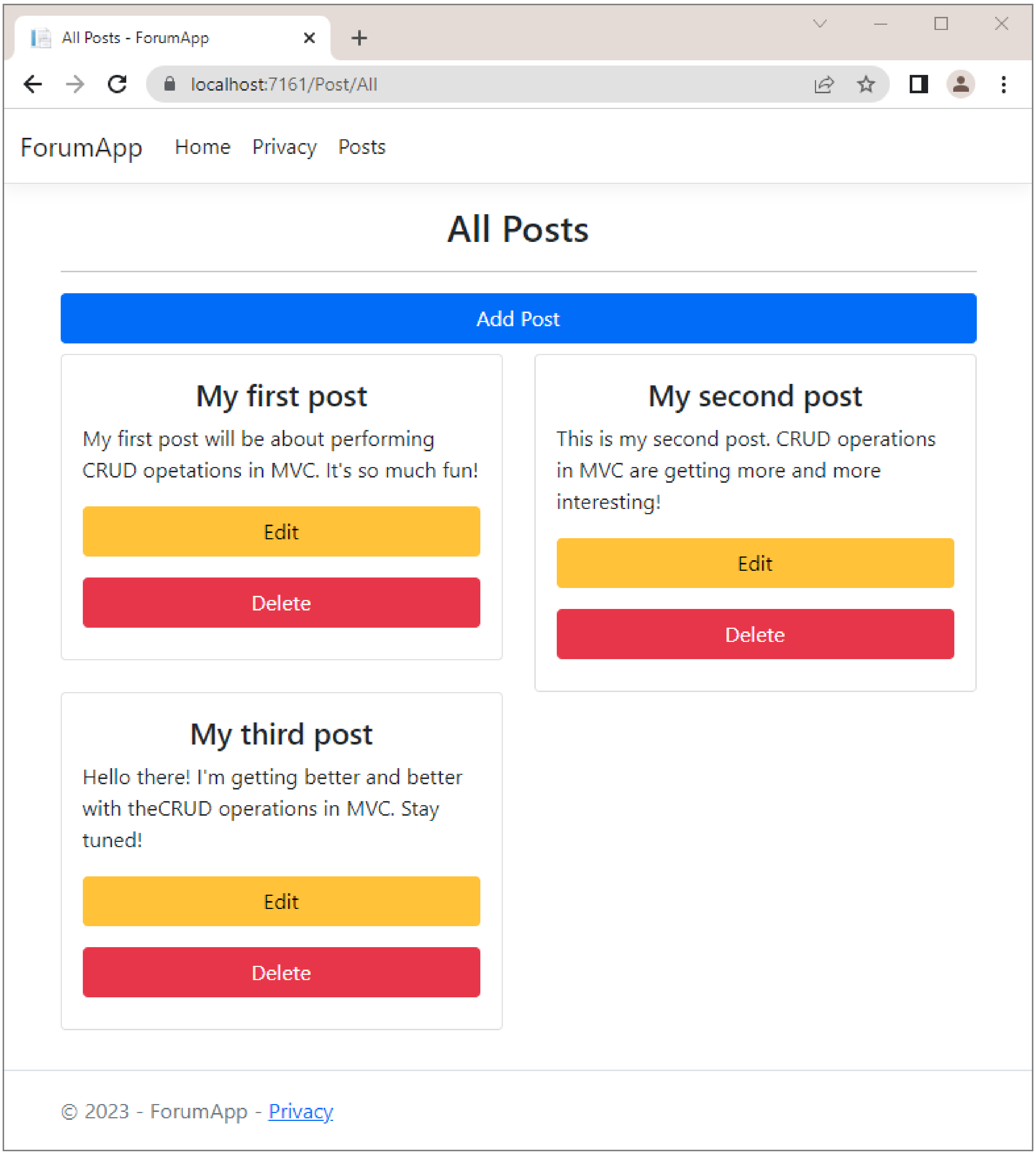
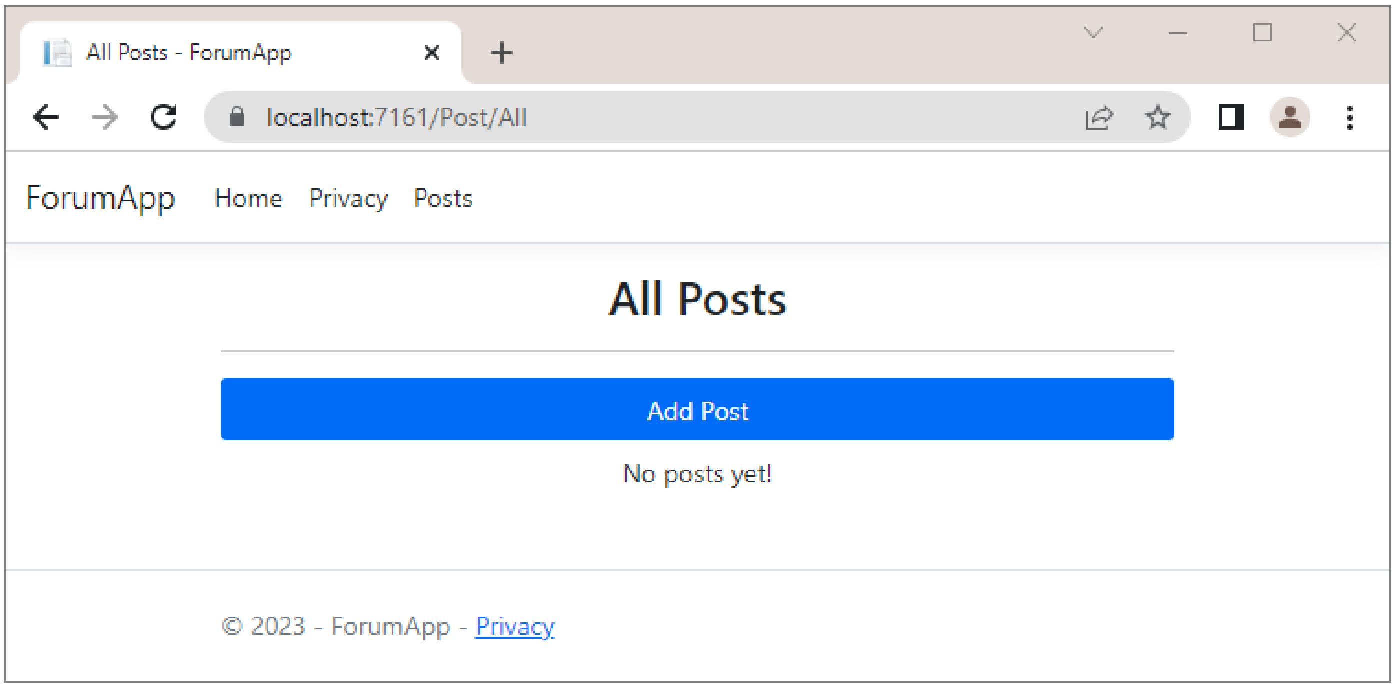
Workshop: ASP.NET and Databases

Workshop for the ["ASP.NET Core Fundamentals" course @ SoftUni](#)

In this workshop we will create a **simple `ForumApp`, similar to the `ShoppingListApp`** from the slides from the previous session. Our app will have a page for **displaying all added forum posts** with their content and with [**`Edit`**] and [**`Delete`**] buttons and an [**Add Post**] button. It also should display a form for adding a post and editing a post and it should ask for a confirmation when deleting a post. It will look like this:


All Posts - ForumApp
localhost:7161/Post/All
ForumApp
Home
Privacy
Posts
All Posts
Add Post
My first post
My first post will be about performing CRUD opetations in MVC. It's so much fun!
Edit
Delete
My second post
This is my second post. CRUD operations in MVC are getting more and more interesting!
Edit
Delete
My third post
Hello there! I'm getting better and better with theCRUD operations in MVC. Stay tuned!
Edit
Delete
© 2023 - ForumApp - Privacy



All Posts

Add Post

No posts yet!

© 2023 - ForumApp - Privacy

1. Create a New App

Create a new **`ASP.NET Core Web App (Model-View-Controller)`** and name it **`ForumApp`**.

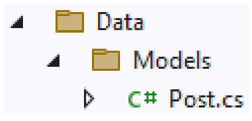
Remove the unnecessary code from the **`HomeController`** class. It should look like this:

```
public class HomeController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }

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

    [ResponseCache(Duration = 0,
        Location = ResponseCacheLocation.None, NoStore = true)]
    0 references
    public IActionResult Error()
    {
        return View(new ErrorViewModel
                  { RequestId = Activity.Current?.Id
                      ?? HttpContext.TraceIdentifier });
    }
}
```

2. Define Entities

We should start with creating the data model, which we will need for our database. We will have only one data model class – **`Post`**.

Create a "**Data**" folder in the project. It will hold the context of our app and the data models. The data models will be placed in a separate folder in the "**Data**" folder, which we will name "**Models**". It's a good practice for the data model classes to be in a separate folder from the **ForumAppDbContext** class. Create a new class and name it **Post**:



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

- **Id** – an unique **integer**, primary key
- **Title** – a **string** with min length **10** and max length **50** (required)
- **Content** – a **string** with min length **30** and max length **1500** (required)

First, create the **Post** class in the "**Models**" folder with its **properties without restrictions:**

```
public class Post
{
    0 references
    public int Id { get; init; }
    0 references
    public string Title { get; set; } = null!;
    0 references
    public string Content { get; set; } = null!;
}
```

Note that we use **init** setters only for **properties**, which **won't be changed after the initialization**. For example, the **Id** property will always be the same.

Now, add the **[Required] attribute** to those **properties**, which **must have a value**.

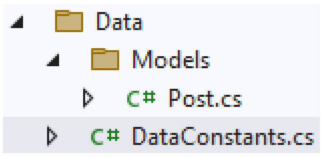
```
public class Post
{
    0 references
    public int Id { get; init; }

    [Required]
    0 references
    public string Title { get; set; } = null!;

    [Required]
    0 references
    public string Content { get; set; } = null!;
}
```

Note that you don't need to set the **Id** property as a **primary key** in any way – this is done **by default** in **EF Core** because of the "**Id**" property name.

Now we should **restrict the properties**. The only restriction we will set is for the **max length**, as the database doesn't care about other validations. **Min length**, **range** and other **restrictions** are added to **model classes**.

The **properties**, which should have a **max length,** are **Title** and **Content**.

It is a good idea to create a **separate class with constants** for the max length values. In the "**Data**" **folder**, create the **DataConstants class**:



In this class, create a **`Post`** class for the constants, connected to a post entity, and the constants themselves:

```
public class DataConstants
{
    0 references
    public class Post
    {
        public const int TitleMaxLength = 50;
        public const int ContentMaxLength = 1500;
    }
}
```

Add the needed namespace to use the constant class in the **`Post`** class:

```
using static ForumApp.Data.DataConstants.Post;
```

Use the **constants** in the **`[MaxLength]` attribute** to **set the max length** like this:

```
public class Post
{
    0 references
    public int Id { get; init; }

    [Required]
    [MaxLength(TitleMaxLength)]
    0 references
    public string Title { get; set; } = null!;

    [Required]
    [MaxLength(ContentMaxLength)]
    0 references
    public string Content { get; set; } = null!;
}
```

Now the model class is ready.

3. Define the DbContext

Now let's add our **`ForumAppDbContext`** class in the "**`Data`**" folder. In order for the ForumAppDbContext to inherit the DbContext class, we have to install the **`Microsoft.EntityFrameworkCore`** package. Don't forget to install the following packages, too, as we will need them for this workshop:

- **`Microsoft.EntityFramewrokCore.Design`**
- **`Microsoft.EntityFramewrokCore.SqlServer`**
- **`Microsoft.EntityFramewrokCore.Tools`**

```
public class ForumAppDbContext : DbContext
{
    0 references
    public ForumAppDbContext(DbContextOptions<ForumAppDbContext> options)
        : base(options)
    {
    }
}
```

We should use the **`Migrate()`** method in the constructor in order for the changes to be applied to tbe database directly.

```
public class ForumAppDbContext : DbContext
{
    0 references
    public ForumAppDbContext(DbContextOptions<ForumAppDbContext> options)
        : base(options)
    {
        Database.Migrate();
    }
}
```

Create the **DbSet** properties for the table in the database:

```
    public DbSet<Post> Posts { get; init; }
```

EF Core uses a "**cascade**" **delete** by default when **removing an entity**. This means that if a record in the **parent table is deleted**, then the **corresponding records** in the child table **will automatically be deleted**. To prevent this from happening, it's a good practice to set the **delete behavior** to "**restrict**", so that an entity, which has **connections** to other entities in the database, **won't be deleted.**

In our case, there's no entity that has connections to other entities. If there was such entity, the **OnModelCreating(ModelBuilder builer)** method in the **ForumAppDbContext** class should be overridden like in the example below:

```
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder
            .Entity<Post>()
            .HasMany(p => p.PostAnswer)
            .WithOne(pa => pa.Post)
            .OnDelete(DeleteBehavior.Restrict);
    }
```

At the end, invoke the **base OnModelCreating() method**:

```
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder
            .Entity<Post>();

        base.OnModelCreating(modelBuilder);
    }
```

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

4. Seed the Database

Now, we want to **populate the database** with an **initial set of data**. This will include **three posts.**

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

```
        private Post FirstPost { get; set; }
        2 references
        private Post SecondPost { get; set; }
        2 references
        private Post ThirdPost { get; set; }
```

Then, we will use a **separate method** to **add data to these objects**, which will be **added** to the **corresponding database table** in the **`OnModelCreating(ModelBuilder builder)` method**. Add the following lines of code to the method, before invoking the base one:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    SeedPosts();
    modelBuilder
        .Entity<Post>()
        .HasData(FirstPost, SecondPost, ThirdPost);

    base.OnModelCreating(modelBuilder);
}
```

When **invoking more that one seeding method**, it is important that the **seeding methods are invoked in the correct order**, as they **depend** on each other.

Let's implement the **`SeedPosts()`** class, which will create the posts for the database.

```
private void SeedPosts()
{
    FirstPost = new Post()
    {
        Id = 1,
        Title = "My first post",
        Content = "My first post will be about performing " +
        "CRUD opetations in MVC. It's so much fun!"
    };

    SecondPost = new Post()
    {
        Id = 2,
        Title = "My second post",
        Content = "This is my second post. " +
        "CRUD operations in MVC are getting more and more interesting!"
    };

    ThirdPost = new Post()
    {
        Id = 3,
        Title = "My third post",
        Content = "Hello there! I'm getting better and better with the" +
        "CRUD operations in MVC. Stay tuned!"
    };
}
```

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

5. Create a Migration

Now we will now **create a migration** to the database.

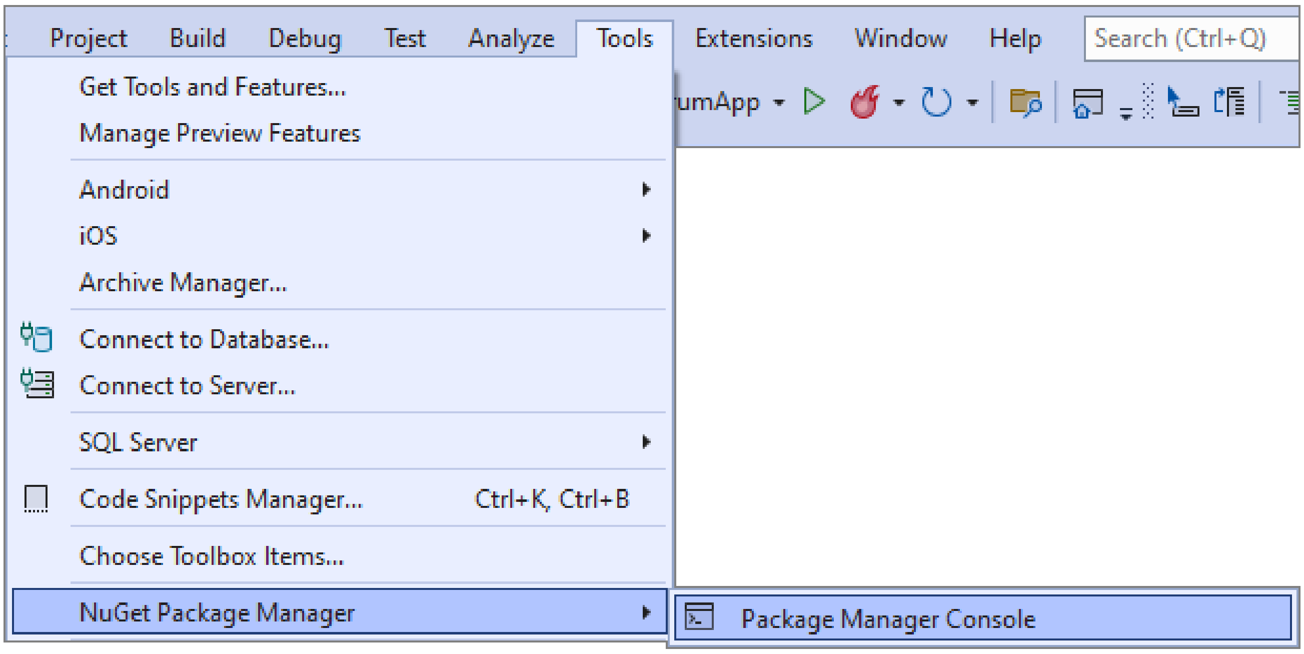
To do that, first we need to add the connection string in the "**`appsettings.json`**" file:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=ForumApp;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
```

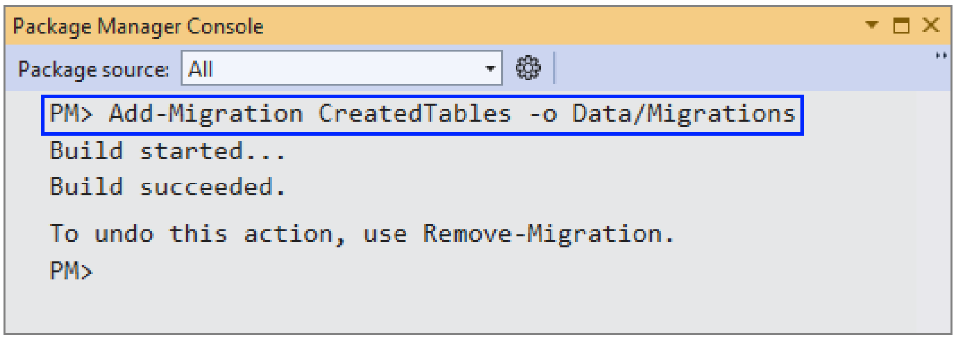
and set the **DbContext** to use SQL with the connection string in the **Program** class.

```
var connectionString = builder
    .Configuration
    .GetConnectionString("DefaultConnection");

builder
    .Services.AddDbContext<ForumAppDbContext>(
        options => options.UseSqlServer(connectionString));
```

Next, **open** the **Package Manager Console** from **[Tools]** → **[NuGet Package Manager]** → **[Package Manager Console]** to write **commands** for **managing migrations**:

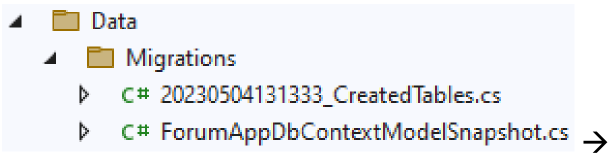


In the **console**, write a command for **adding a migration** to the "**Data/Migrations**" **folder** with a given **name** and **press [Enter]** to **execute it**. The **result** should be the following:



```
PM> Add-Migration CreatedTables -o Data/Migrations
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

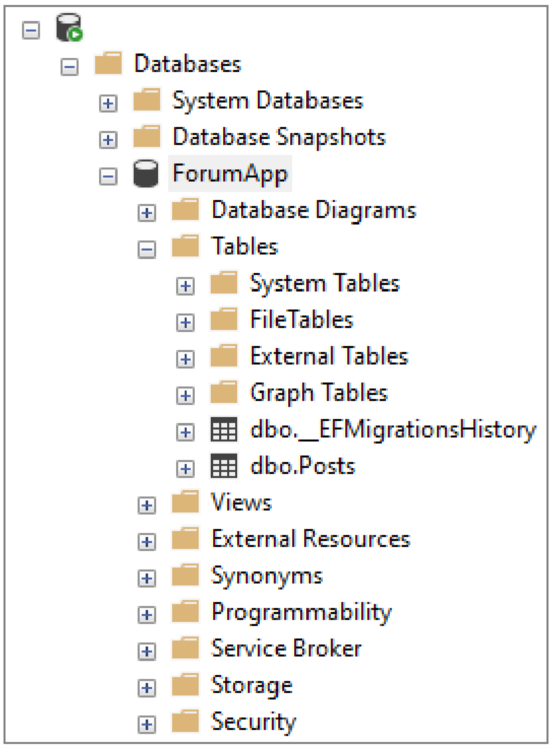
Now you should have a new migration in the "**Migrations**" folder:


Data
Migrations
20230504131333_CreatedTables.cs
ForumAppDbContextModelSnapshot.cs

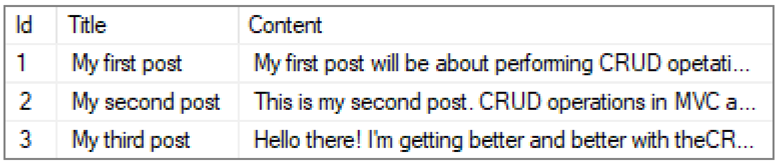
```
public partial class CreatedTables : Migration
{
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Posts",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Title = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                Content = table.Column<string>(type: "nvarchar(1500)", maxLength: 1500, nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Posts", x => x.Id);
            });

        migrationBuilder.InsertData(
            table: "Posts",
            columns: new[] { "Id", "Content", "Title" },
            values: new object[] { 1, "My first post will be about performing CRUD opetations in MVC. It's so much fun!", "My first post" });

        migrationBuilder.InsertData(
            table: "Posts",
            columns: new[] { "Id", "Content", "Title" },
            values: new object[] { 2, "This is my second post. CRUD operations in MVC are getting more and more interesting!", "My second post" });

        migrationBuilder.InsertData(
            table: "Posts",
            columns: new[] { "Id", "Content", "Title" },
            values: new object[] { 3, "Hello there! I'm getting better and better with theCRUD operations in MVC. Stay tuned!", "My third post" });
    }

    0 references
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Posts");
    }
}
```

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

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


```
⊟ 
  ⊟ Databases
    ⊞ System Databases
    ⊞ Database Snapshots
    ⊟ ForumApp
      ⊞ Database Diagrams
      ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Graph Tables
        ⊞ dbo.__EFMigrationsHistory
        ⊞ dbo.Posts
      ⊞ Views
      ⊞ External Resources
      ⊞ Synonyms
      ⊞ Programmability
      ⊞ Service Broker
      ⊞ Storage
      ⊞ Security
```

Look at the "**Posts**" table – it should contain the three posts that we seeded:



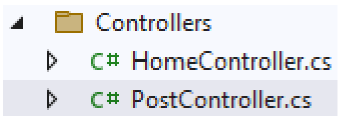
Id	Title	Content
1	My first post	My first post will be about performing CRUD opetati...
2	My second post	This is my second post. CRUD operations in MVC a...
3	My third post	Hello there! I'm getting better and better with theCR...

6. CRUD operations

After we have populated our database, we can start executing CRUD operations.

Reading Data

First, we need to create a new **PostController** in the "**Controllers**" folder".



```
◢ Controllers
  ▷ C# HomeController.cs
  ▷ C# PostController.cs
```

Let's inject the **ForumAppDbContext** through the constructor and assign it to a variable to use it:

```
public class PostController : Controller
{
    private readonly ForumAppDbContext _data;

    0 references
    public PostController(ForumAppDbContext data)
    {
        _data = data;
    }
}
```

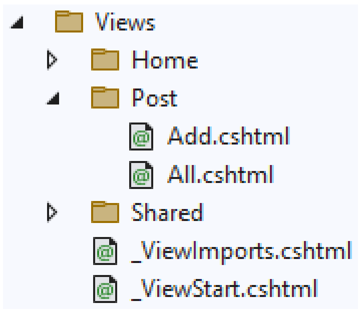
Create a "**Post**" folder in the "**Models**" folder of the project. Our next step should be creating the **PostViewModel** in the **"Models/Post"** folder:

```
public class PostViewModel
{
    0 references
    public int Id { get; set; }
    0 references
    public string Title { get; set; } = null!;
    0 references
    public string Content { get; set; } = null!;
}
```

After the **PostViewModel** is created, we'll go back to the **ProductController** and add a new method **All()**. The task of this method is to extract the products from the database to a model collection, which will be passed to a view.

```
    public async Task<IActionResult> All()
    {
        var posts = await _data
            .Posts
            .Select(p => new PostViewModel()
            {
                Id = p.Id,
                Title = p.Title,
                Content = p.Content
            })
            .ToListAsync();

        return View(posts);
    }
```

Now we should create the "**`All.csthml`**" view file. We will create all of the views in a new folder, called "**`Post`**", which will be located in the "**`Views`**" folder.



→

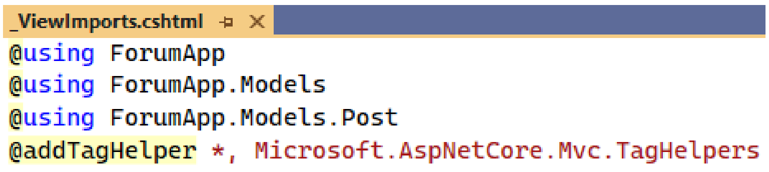
All.cshtml

```
@model List<PostViewModel>

@{
    ViewBag.Title = "All Posts";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />
<div class="d-grid gap-2 mb-2">
    <a asp-controller="Post" asp-action="Add" class="btn btn-primary">Add Post</a>
</div>


@if (Model.Count() > 0)
{
    <div class="row">
        @foreach (var post in Model)
        {
            <div class="col-sm-6">
                <div class="card ">
                    <div class="card-body">
                        <h4 class="card-title text-center">@post.Title</h4>
                        <p class="card-text">@post.Content</p>
                        <div class="d-grid gap-2">
                            <a asp-controller="Post" asp-action="Edit" asp-route-id="@post.Id" class="btn btn-warning">Edit</a>
                            <form class="mt-2" asp-controller="Post" asp-action="Delete" asp-route-id="@post.Id">
                                <div class="d-grid gap-2">
                                    <input type="submit" value="Delete" class="btn btn-danger mb-2" />
                                </div>
                            </form>
                        </div>

                    </div>
                </div>
            </div>
        }
    </div>
}
else
{
    <p class="text-center">No posts yet!</p>
}
```

As this file contains more code, you can copy it from here:

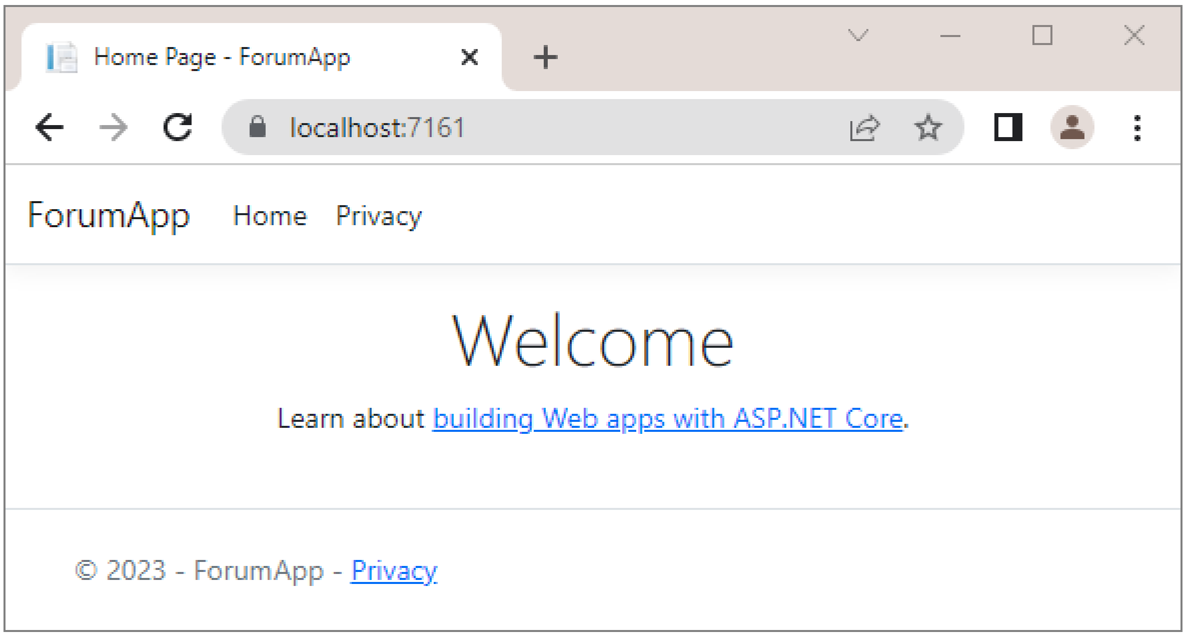
```
@model List<PostViewModel>

@{
    ViewBag.Title = "All Posts";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />
<div class="d-grid gap-2 mb-2">
    <a asp-controller="Posts" asp-action="Add" class="btn btn-primary">Add Post</a>
</div>


@if (Model.Count() > 0)
{
    <div class="row">
        @foreach (var post in Model)
        {
            <div class="col-sm-6">
                <div class="card ">
                    <div class="card-body">
                        <h4 class="card-title text-center">@post.Title</h4>
                        <p class="card-text">@post.Content</p>
                        <div class="d-grid gap-2">
                            <a asp-controller="Post" asp-action="Edit" asp-route-
id="@post.Id" class="btn btn-warning">Edit</a>
                            <form class="mt-2" asp-controller="Post" asp-
action="Delete" asp-route-id="@post.Id">
                                <div class="d-grid gap-2">
                                    <input type="submit" value="Delete" class="btn
btn-danger mb-2" />
                                </div>
                            </form>
                        </div>

                    </div>
                </div>
            </div>
        }
    </div>
}
else
{
    <p class="text-center">No posts yet!</p>
}
```

Don't forget that you have to add the "**Views/Post**" folder in the **_ViewImports.cshtml** file in order for the application to be able to use the models in the "**/View/Post**" folder:


_ViewImports.cshtml

```
@using ForumApp
@using ForumApp.Models
@using ForumApp.Models.Post
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Now **run the app** in the browser – there should not be **any errors** and the **Home page** should be displayed:


Home Page - ForumApp

localhost:7161

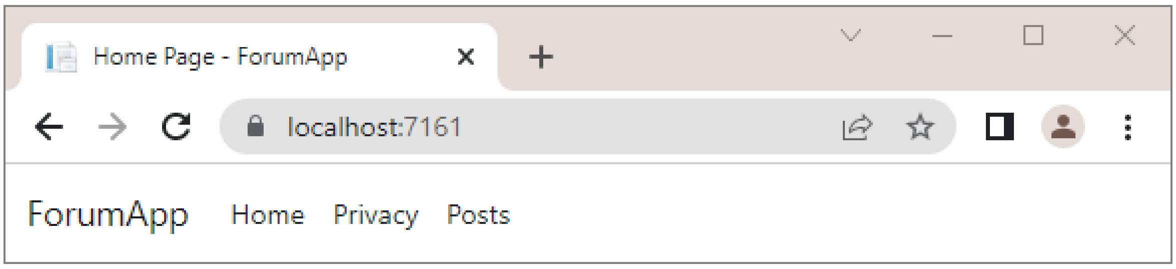
ForumApp Home Privacy

Welcome

Learn about building Web apps with ASP.NET Core.

© 2023 - ForumApp - Privacy

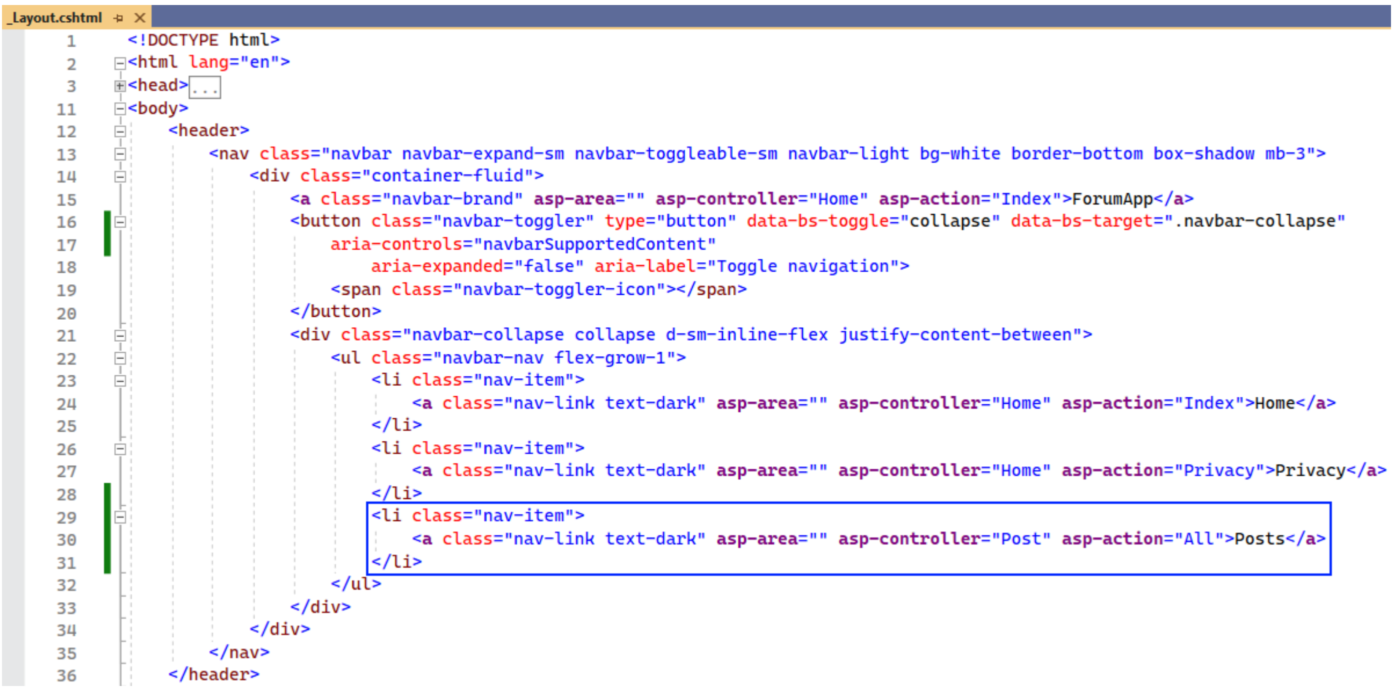
You can access the page that contains the collection with the posts on "`/Posts/All`". For our convenience, we will **add a link to the navigation pane** to access the **Posts** page. The navigation pane should look like this:


Home Page - ForumApp

localhost:7161

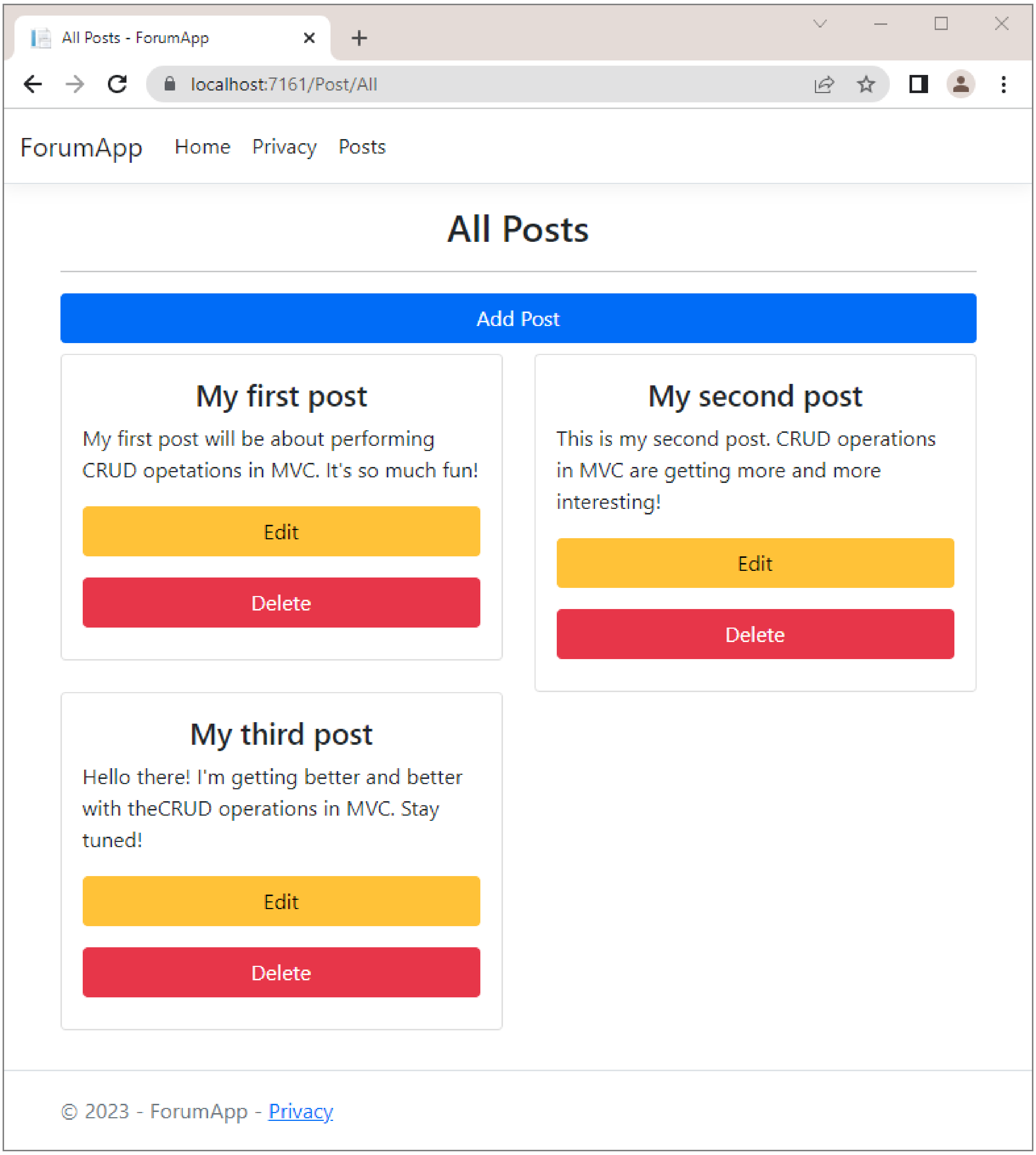
ForumApp Home Privacy Posts

To **add links**, go the **`_Layout.cshtml` partial view** in the "**`/Views/Shared`**" **folder**, as this view is responsible for the **common design** of all pages. Add the following lines:


_Layout.cshtml

```html
<!DOCTYPE html>
<html lang="en">
<head>...
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ForumApp</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
                    aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </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="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>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Post" asp-action="All">Posts</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
```

If we **follow** the **newly added link**, we should be able to see the **Posts** page and it should look like this:



All Posts

My first post

My first post will be about performing CRUD opetations in MVC. It's so much fun!

My second post

This is my second post. CRUD operations in MVC are getting more and more interesting!

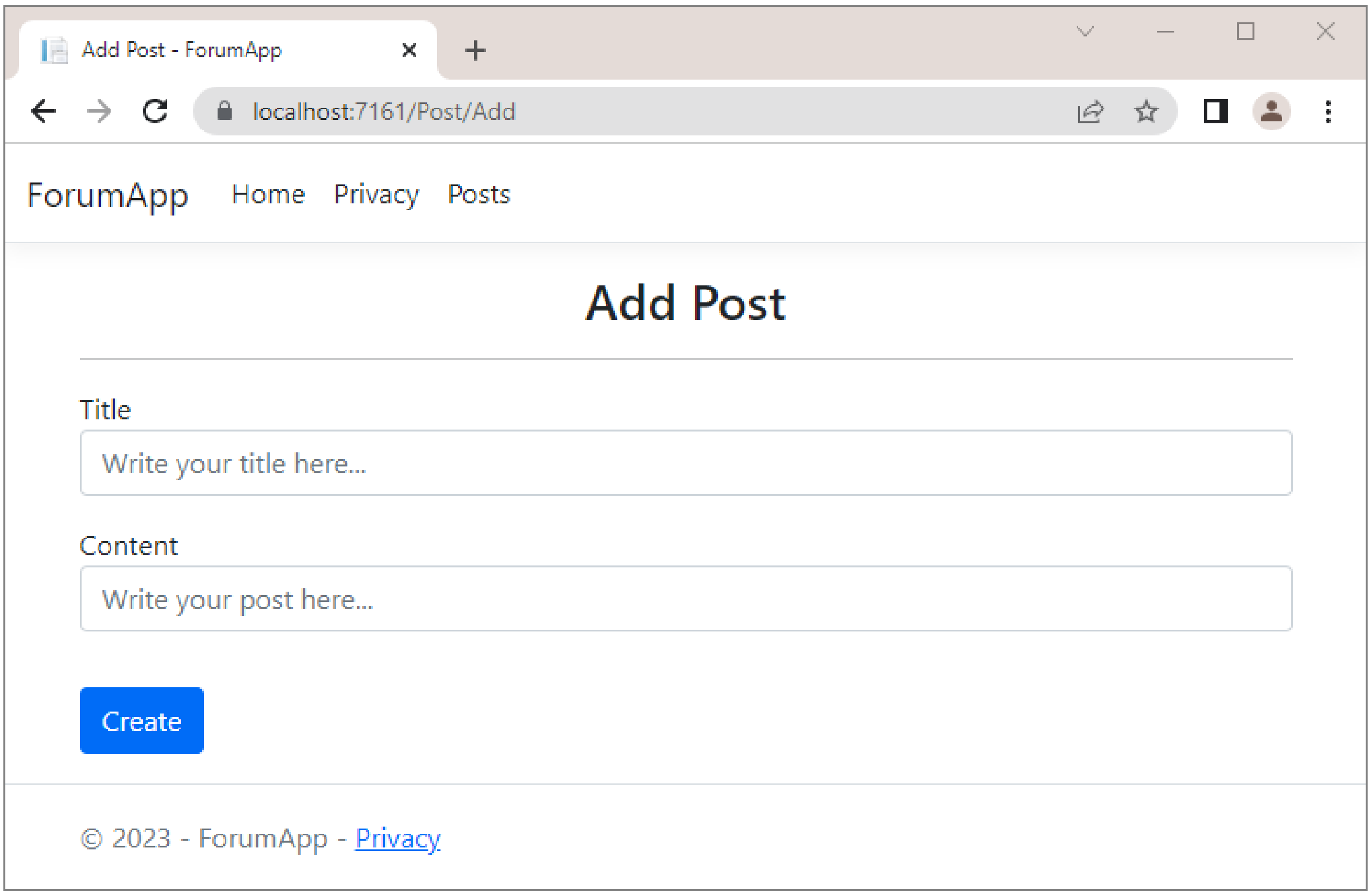
My third post

Hello there! I'm getting better and better with theCRUD operations in MVC. Stay tuned!

© 2023 - ForumApp - Privacy

Creating New Data

The next step in creating the **ForumApp** is adding the "**Add Post**" page. It will display a **form** for **adding a post** and it will look like this:


Add Post - ForumApp

localhost:7161/Post/Add

ForumApp Home Privacy Posts

Add Post

Title

Write your title here...

Content

Write your post here...

Create

© 2023 - ForumApp - Privacy

First, we need to create the **`PostViewModel`** in the **`"Models"`** folder.

Before that, let's go back to the **`DataContants`** class in the "**`Data`**" folder and add the values for the min length constants:

```
public class DataConstants
{
    1 reference
    public class Post
    {
        public const int TitleMaxLength = 50;
        public const int TitleMinLength = 10;

        public const int ContentMaxLength = 1500;
        public const int ContentMinLength = 30;
    }
}
```

We will add validation attributes to the model properties of the **`PostViewModel`**. The **`[Required]`** attribute will check if the model property holds any value and the **`[StringLength]`** will check the length of the string that is held as a value.

```
public class PostFormModel
{
    [Required]
    [StringLength(TitleMaxLength, MinimumLength = TitleMinLength)]
    0 references
    public string Title { get; set; } = null!;

    [Required]
    [StringLength(ContentMaxLength, MinimumLength = ContentMinLength)]
    0 references
    public string Content { get; set; } = null!;
}
```

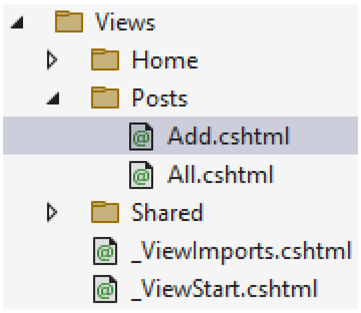
Don't forget to add the needed namespace to use the constant class in the **`PostFormModel`** class:

```
using static ForumApp.Data.DataConstants.Post;
```

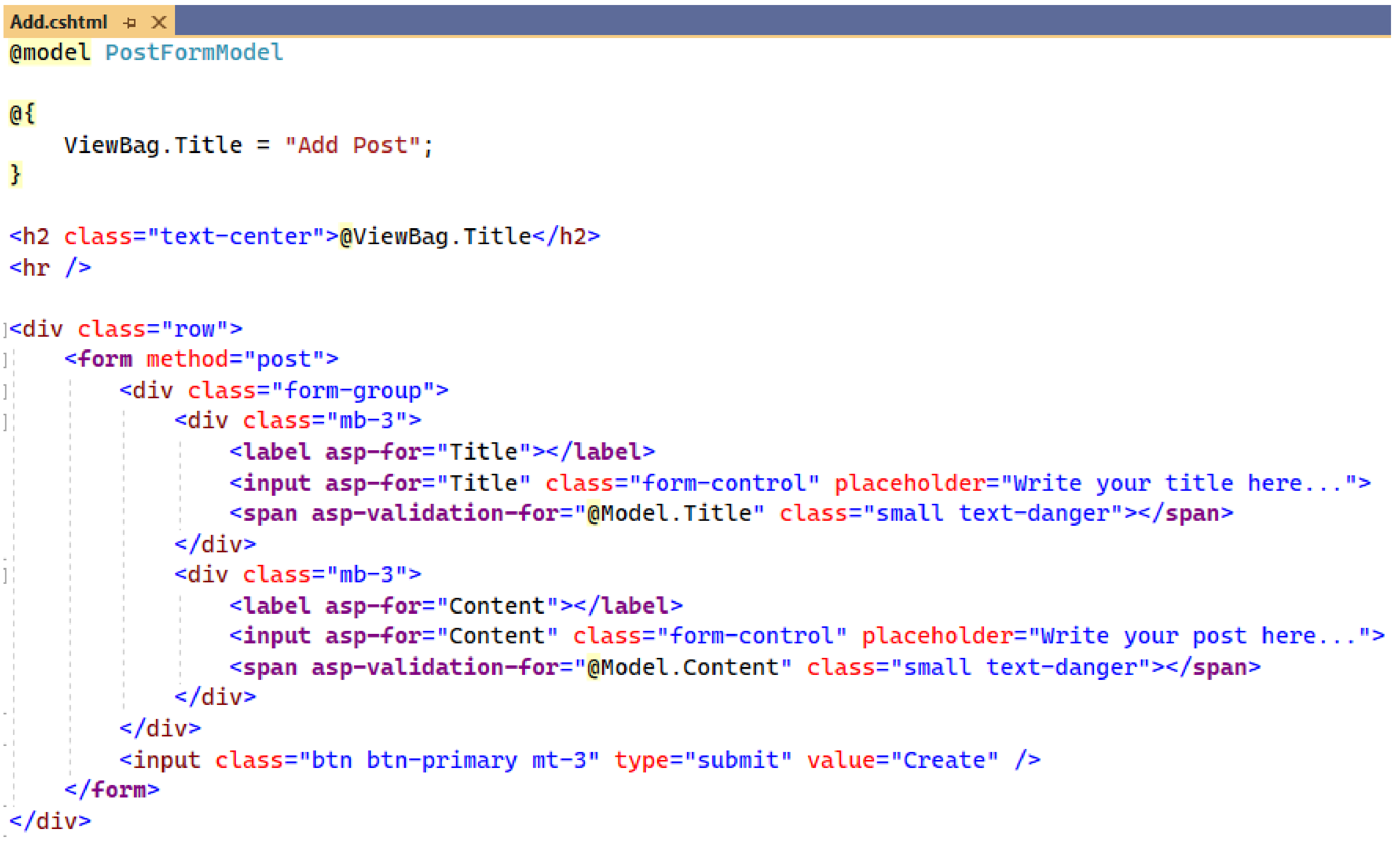
After we have created the **`PostFormModel`** we should go to the **`PostController`** and implement the **`Add()`** method. This method will create a new **`Post`** object and then add it to the **`DbSet`**.

```
    public async Task<IActionResult> Add()
        => View();

    [HttpPost]
    0 references
    public async Task<IActionResult> Add(PostFormModel model)
    {
        var post = new Post()
        {
            Title = model.Title,
            Content = model.Content
        };

        await _data.Posts.AddAsync(post);
        await _data.SaveChangesAsync();

        return RedirectToAction("All");
    }
```

Finally, we have to add a new "**`Add.cshtml`**" in the **"`/Views/Posts`"** folder for the **"Add Product"** Page view:



The file should look like this:


Add.cshtml

```
@model PostFormModel

@{
    ViewBag.Title = "Add Post";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />

<div class="row">
    <form method="post">
        <div class="form-group">
            <div class="mb-3">
                <label asp-for="Title"></label>
                <input asp-for="Title" class="form-control" placeholder="Write your title here...">
                <span asp-validation-for="@Model.Title" class="small text-danger"></span>
            </div>
            <div class="mb-3">
                <label asp-for="Content"></label>
                <input asp-for="Content" class="form-control" placeholder="Write your post here...">
                <span asp-validation-for="@Model.Content" class="small text-danger"></span>
            </div>
        </div>
        <input class="btn btn-primary mt-3" type="submit" value="Create" />
    </form>
</div>
```

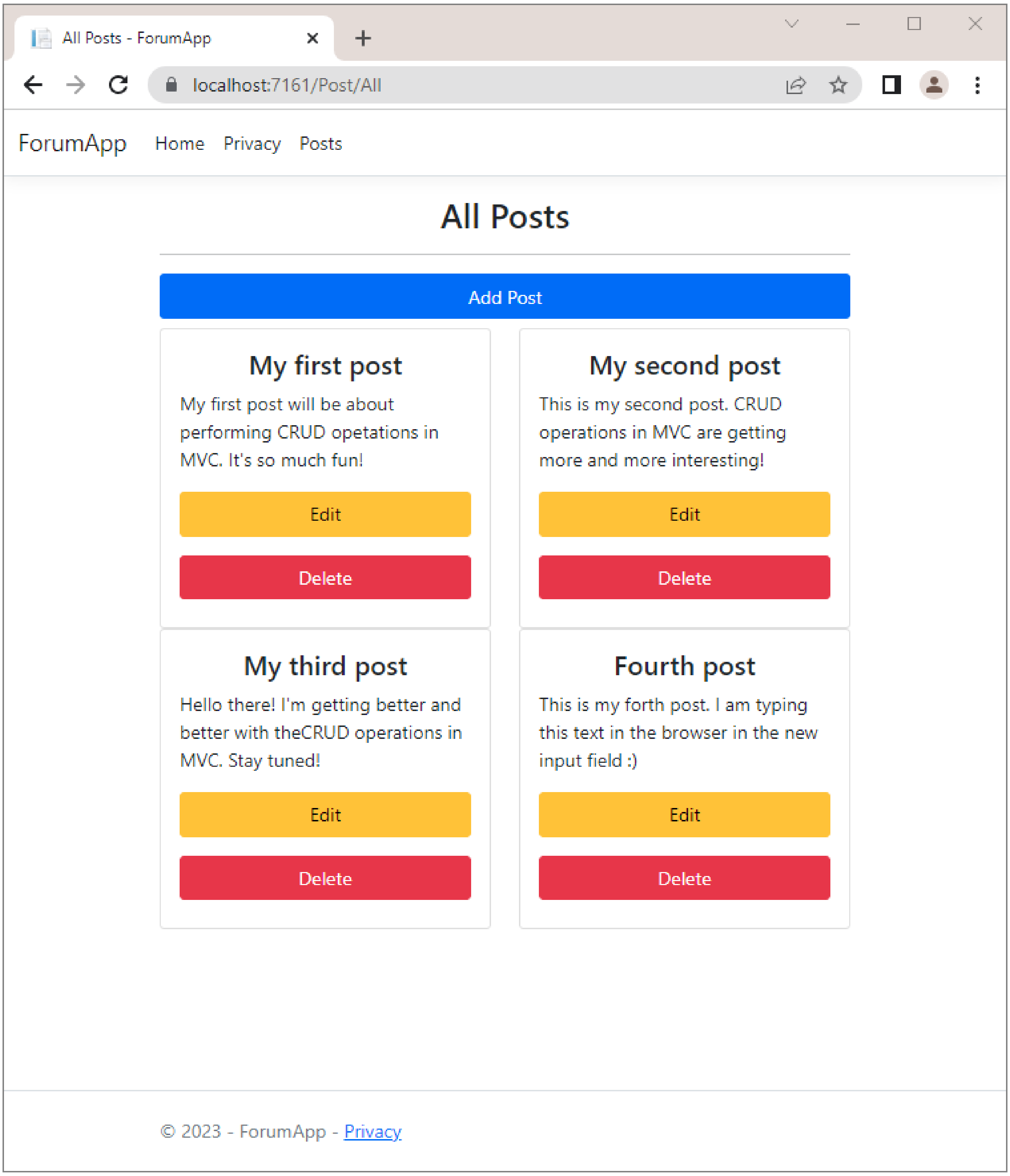
You can copy the code from here:

```
@model PostFormModel

@{
    ViewBag.Title = "Add Post";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />

<div class="row">
    <form method="post">
        <div class="form-group">
            <div class="mb-3">
                <label asp-for="Title"></label>
                <input asp-for="Title" class="form-control" placeholder="Write your 
title here...">
                <span asp-validation-for="@Model.Title" class="small text-
danger"></span>
            </div>
            <div class="mb-3">
                <label asp-for="Content"></label>
                <input asp-for="Content" class="form-control" placeholder="Write 
your post here...">
                <span asp-validation-for="@Model.Content" class="small text-
danger"></span>
            </div>
        </div>
        <input class="btn btn-primary mt-3" type="submit" value="Create" />
    </form>
</div>
```

Let's not forget to add the following code in the **Add.cshtml** file in order for the app to be able to validate the input data.

```
@section Scripts{
    <partial name="_ValidationScriptsPartial" />
}
```

Now we can run the app in the browser and add a new post. After adding the new post, the **All Posts** page should look like this:



All Posts

Add Post

My first post

My first post will be about performing CRUD opetations in MVC. It's so much fun!

Edit

Delete

My second post

This is my second post. CRUD operations in MVC are getting more and more interesting!

Edit

Delete

My third post

Hello there! I'm getting better and better with theCRUD operations in MVC. Stay tuned!

Edit

Delete

Fourth post

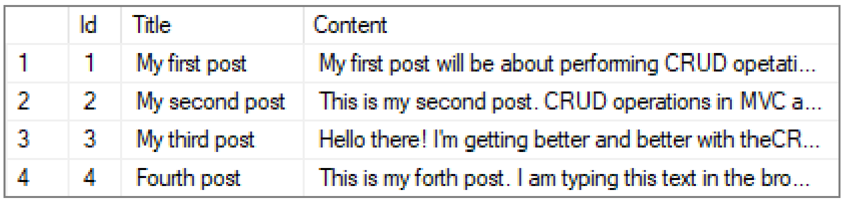
This is my forth post. I am typing this text in the browser in the new input field :)

Edit

Delete

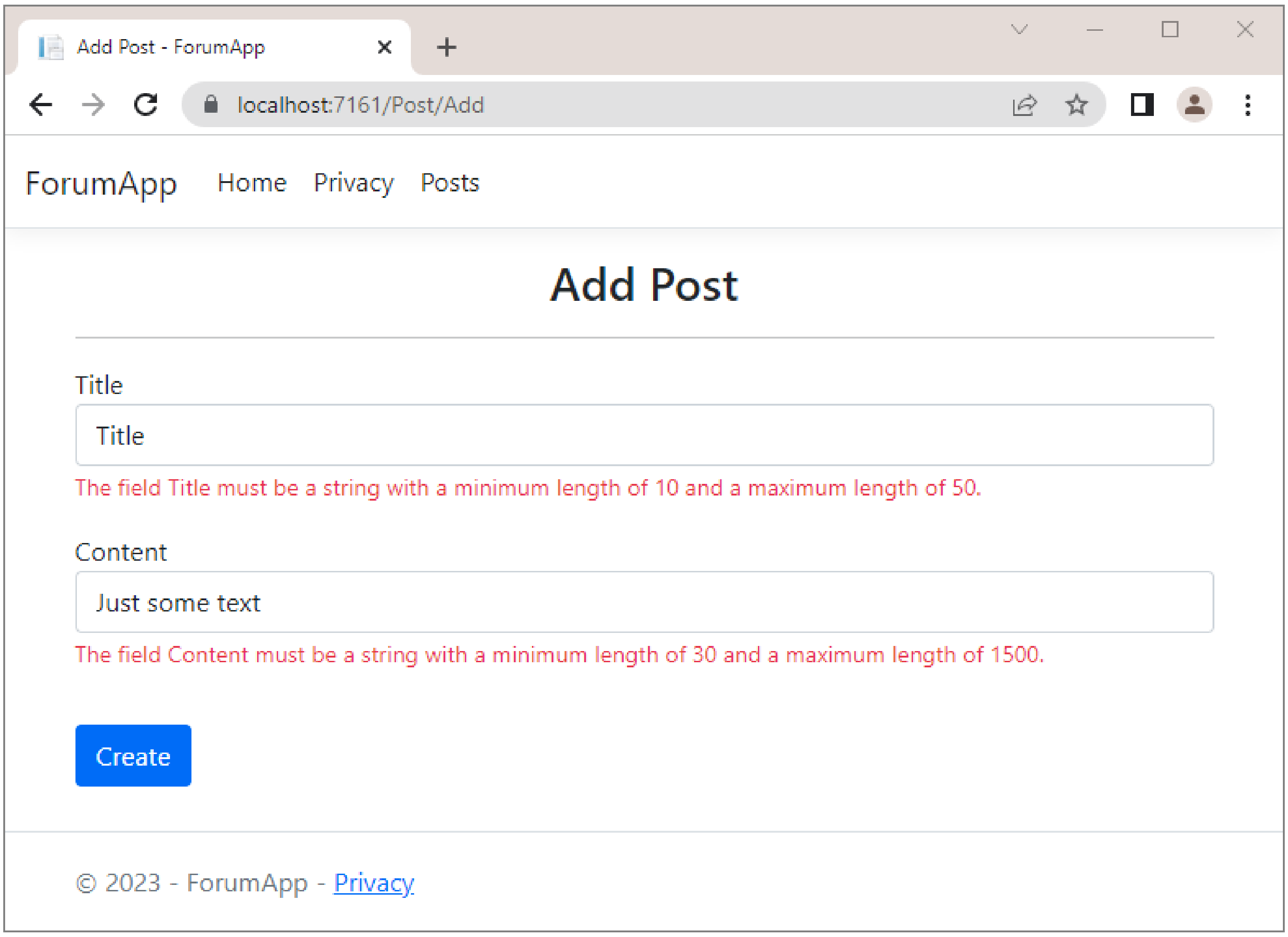
© 2023 - ForumApp - Privacy

Now let's check the database in SQL Server Management Studio and **examine the "`Posts`" table** – it should **contain the new post that we just created:**



	Id	Title	Content
1	1	My first post	My first post will be about performing CRUD opetati...
2	2	My second post	This is my second post. CRUD operations in MVC a...
3	3	My third post	Hello there! I'm getting better and better with theCR...
4	4	Fourth post	This is my forth post. I am typing this text in the bro...

We can check if the app validates the title and content, which we want to add for our post. If their length doesn't meet the requirements, the app should look like this:



Add Post

Title

Title

The field Title must be a string with a minimum length of 10 and a maximum length of 50.

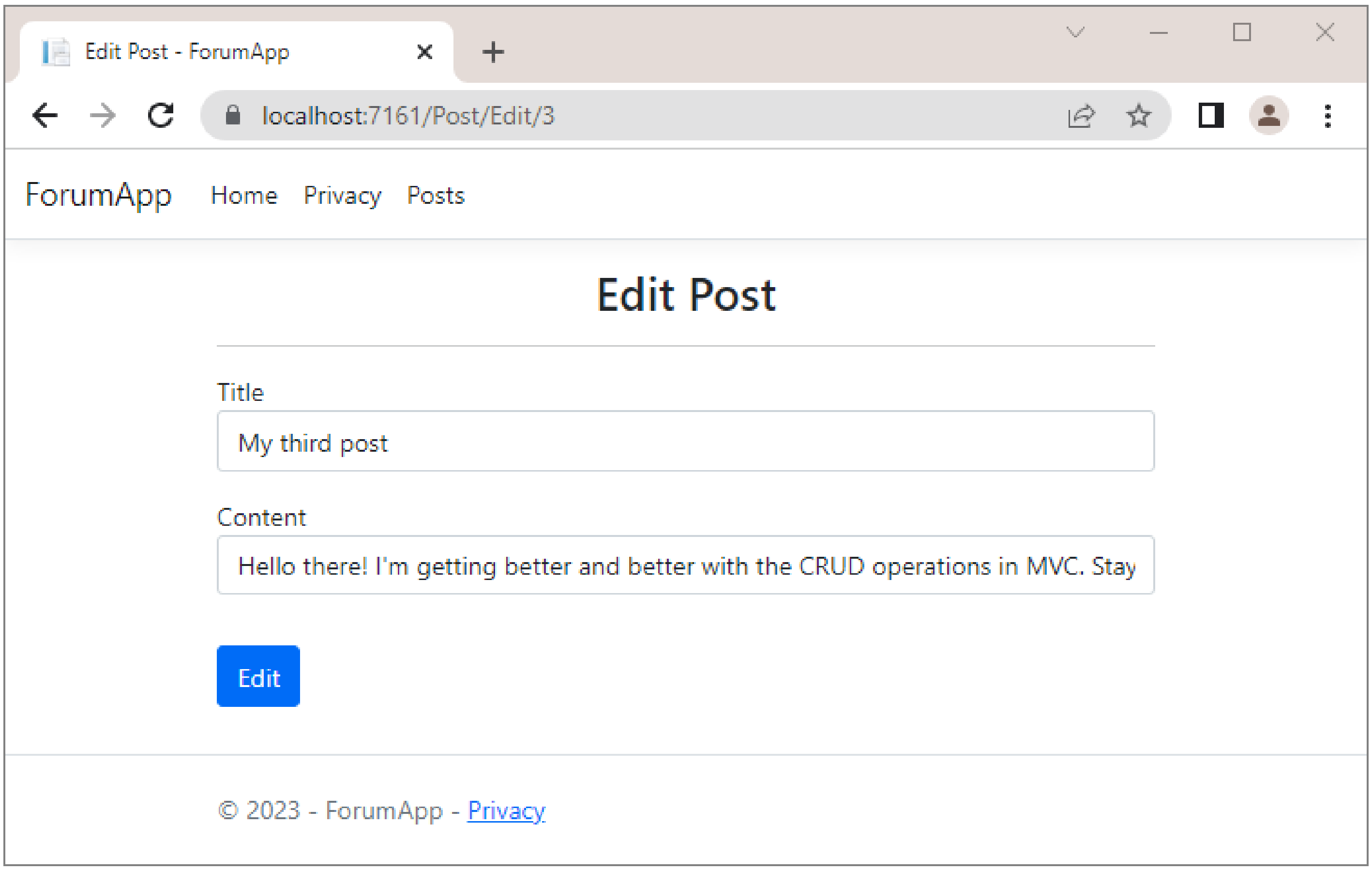
Content

Just some text

The field Content must be a string with a minimum length of 30 and a maximum length of 1500.

Updating Existing Data

The next step in creating the **`ForumApp`** is adding the "**Edit Post**" page. It will display a **form** for **editing a post** and it will look like this:



Edit Post

Title

My third post

Content

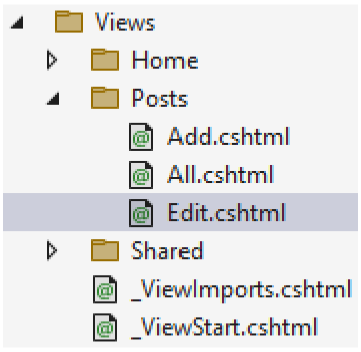
Hello there! I'm getting better and better with the CRUD operations in MVC. Stay

© 2023 - ForumApp - Privacy

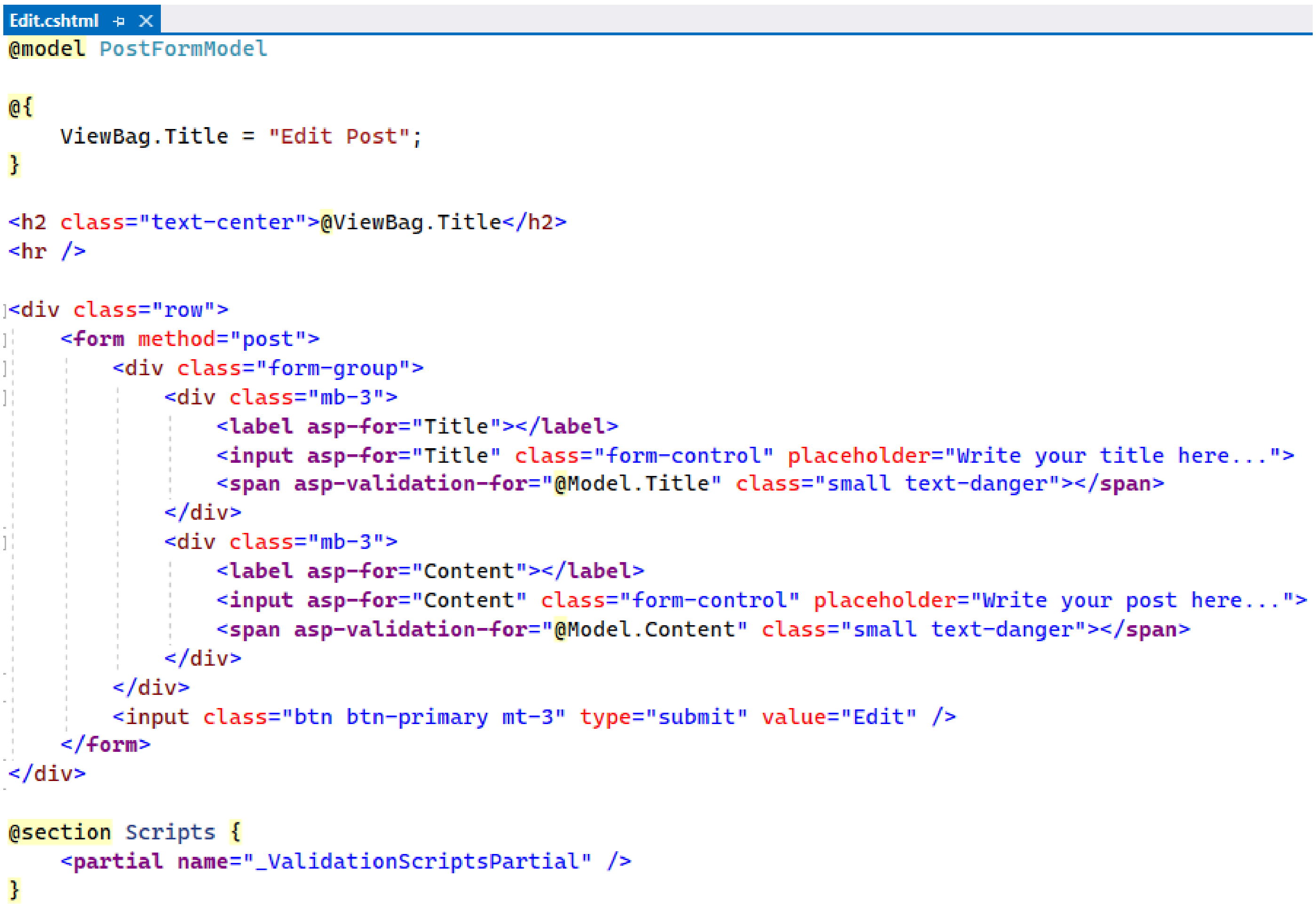
Now let's edit one of the posts that we already have in the collection. First, we need to modify the **`ProductsController`** and add an **`Edit()`** method, which will pass a **`Post`** model to the view and find and update the post in the database.

```
public async Task<IActionResult> Edit(int id)
{
    var post = await _data.Posts.FindAsync(id);

    return View(new PostFormModel()
    {
        Title = post.Title,
        Content = post.Content
    });
}

[HttpPost]
0 references
public async Task<IActionResult> Edit
    (int id, PostFormModel model)
{
    var post = await _data.Posts.FindAsync(id);

    post.Title = model.Title;
    post.Content = model.Content;

    await _data.SaveChangesAsync();

    return RedirectToAction("All");
}
```

Our next step is to add a new "**`Edit.cshtml`**" file to display the form for editing posts.



Let's not forget that we have to add validation for the length of the post's title and content.



```
@model PostFormModel

@{
    ViewBag.Title = "Edit Post";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />

<div class="row">
    <form method="post">
        <div class="form-group">
            <div class="mb-3">
                <label asp-for="Title"></label>
                <input asp-for="Title" class="form-control" placeholder="Write your title here...">
                <span asp-validation-for="@Model.Title" class="small text-danger"></span>
            </div>
            <div class="mb-3">
                <label asp-for="Content"></label>
                <input asp-for="Content" class="form-control" placeholder="Write your post here...">
                <span asp-validation-for="@Model.Content" class="small text-danger"></span>
            </div>
        </div>
        <input class="btn btn-primary mt-3" type="submit" value="Edit" />
    </form>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

You can copy the code from here:

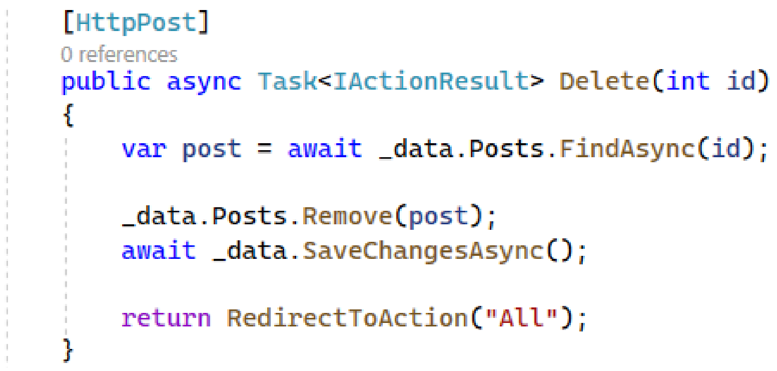
```
@model PostFormModel

@{
    ViewBag.Title = "Edit Post";
}

<h2 class="text-center">@ViewBag.Title</h2>
<hr />

<div class="row">
    <form method="post">
        <div class="form-group">
            <div class="mb-3">
                <label asp-for="Title"></label>
                <input asp-for="Title" class="form-control" placeholder="Write your
title here...">
                <span asp-validation-for="@Model.Title" class="small text-
danger"></span>
            </div>
            <div class="mb-3">
                <label asp-for="Content"></label>
                <input asp-for="Content" class="form-control" placeholder="Write
your post here...">
                <span asp-validation-for="@Model.Content" class="small text-
danger"></span>
            </div>
        </div>
        <input class="btn btn-primary mt-3" type="submit" value="Edit" />
    </form>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Now run the app in the browser and test it. You can also examine the "**Posts**" table in the SQL Server Management Studio.

Deleting Existing Data

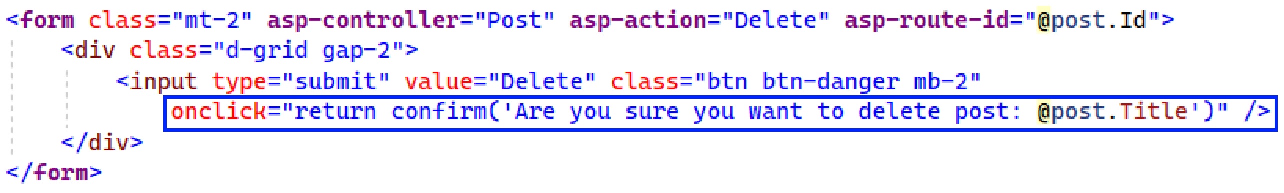
The final step in creating our **ForumApp** is adding the functionality to perform a **Delete** operation. This means that when we click on the [**Delete**] button of a post, it has to send a **POST** reques to the controller. It's a good practice to add a client-side delete confirmation, so we will do that, too.

In order to do that, we need to add a new method **Delete()** in the **PostController**:



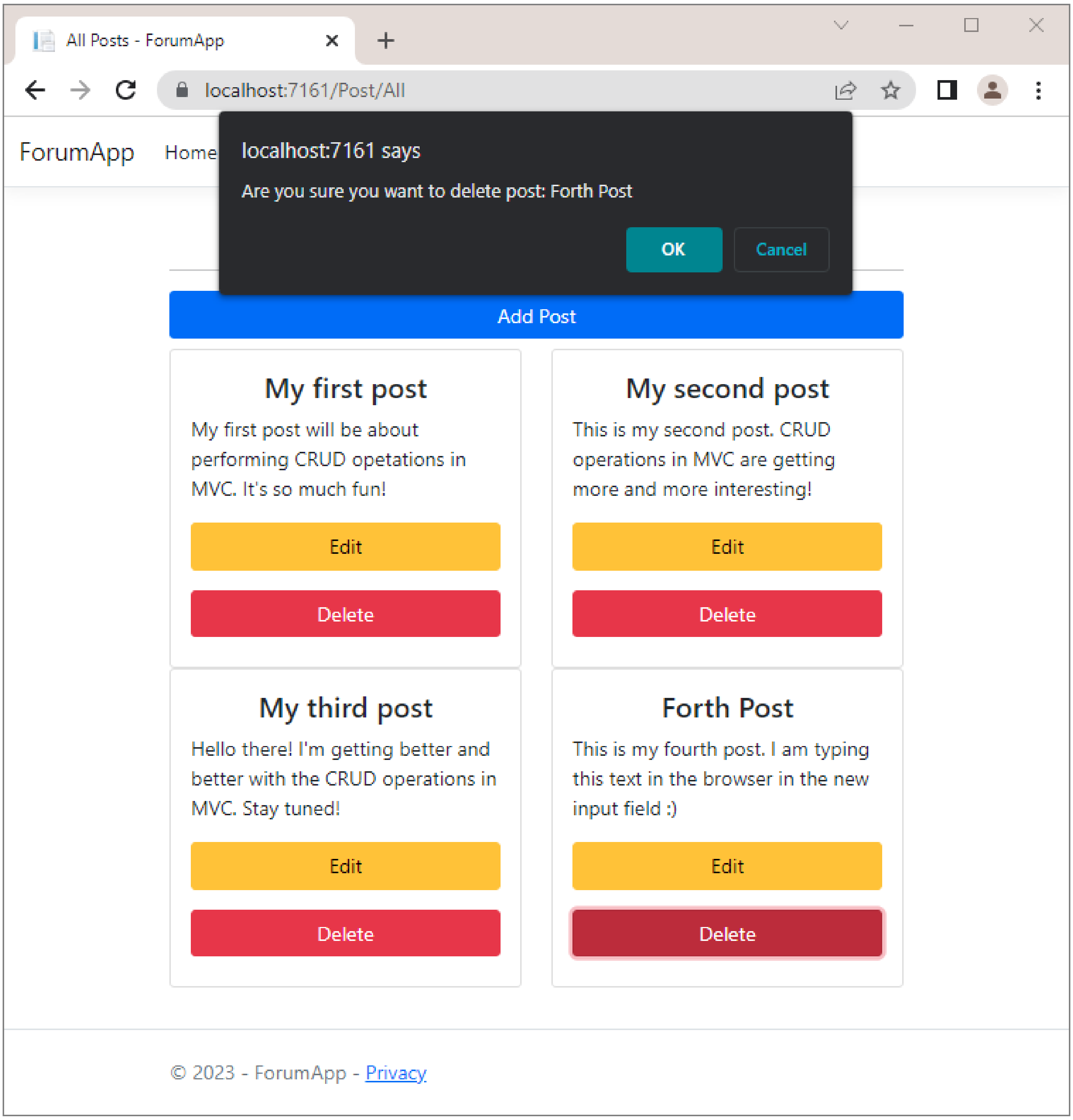
```
[HttpPost]
0 references
public async Task<IActionResult> Delete(int id)
{
    var post = await _data.Posts.FindAsync(id);

    _data.Posts.Remove(post);
    await _data.SaveChangesAsync();

    return RedirectToAction("All");
}
```

Next, we need to go to the "**All.cshtml**" file and add the following code:



```
<form class="mt-2" asp-controller="Post" asp-action="Delete" asp-route-id="@post.Id">
    <div class="d-grid gap-2">
        <input type="submit" value="Delete" class="btn btn-danger mb-2"
            onclick="return confirm('Are you sure you want to delete post: @post.Title')" />
    </div>
</form>
```

The **onclick attribute** of the [**Delete**] button enables the JavaScript **confirm()** function, which displays the confirmation dialog to the user. If the [**Cancel**] button is clicked, the **confirm()** function returns **false** and nothing happens. If the [**OK**] button is clicked, the form is submitted and the post is deleted from the database.


All Posts - ForumApp

localhost:7161/Post/All

ForumApp Home

localhost:7161 says

Are you sure you want to delete post: Forth Post

OK Cancel

Add Post

My first post

My first post will be about performing CRUD opetations in MVC. It's so much fun!

Edit

Delete

My second post

This is my second post. CRUD operations in MVC are getting more and more interesting!

Edit

Delete

My third post

Hello there! I'm getting better and better with the CRUD operations in MVC. Stay tuned!

Edit

Delete

Forth Post

This is my fourth post. I am typing this text in the browser in the new input field :)

Edit

Delete

© 2023 - ForumApp - Privacy

Now run the app in the browser and test it. You can also examine the "**`Posts`**" table in the SQL Server Management Studio.