

ASP.Net MVC Core

Behnam Faghih

August 2022

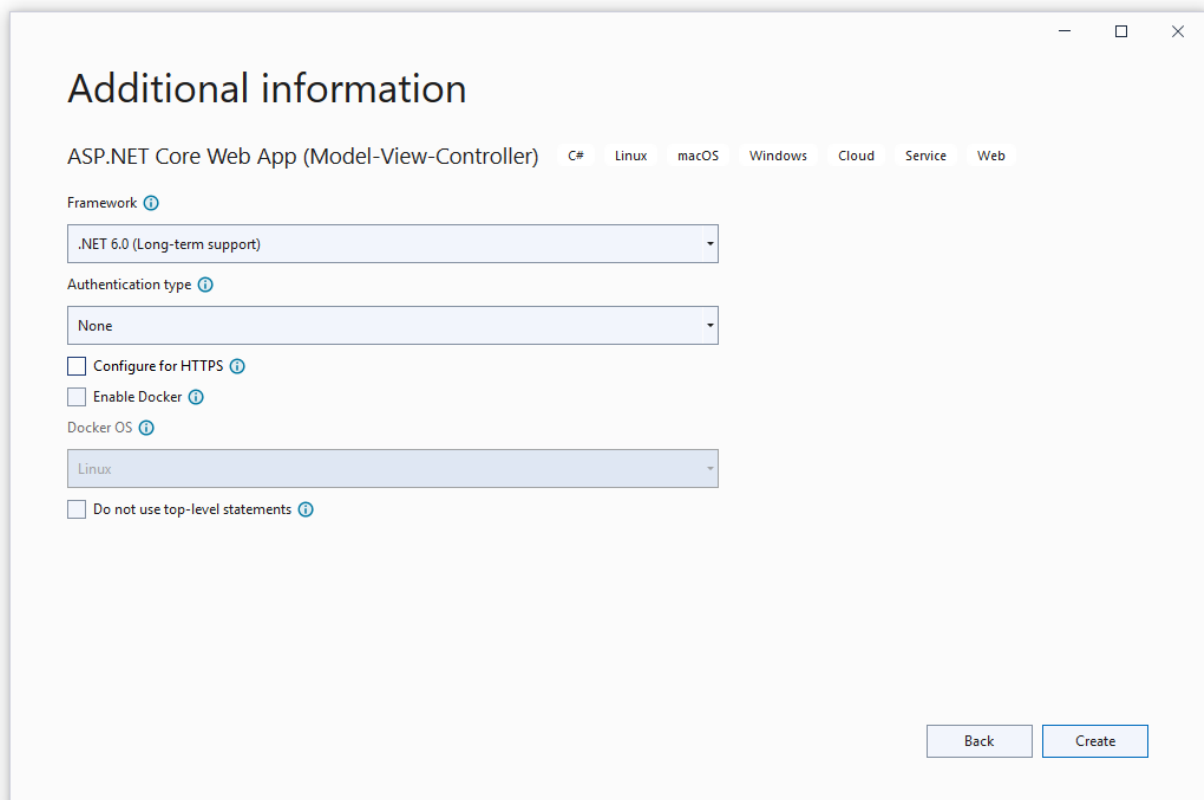
This tutorial aims to introduce the latest Microsoft technologies for creating web applications. The main tools we will use during this tutorial are ASP.NET MVC, .Net 6, Microsoft SQL Server, Entity Framework, and LINQ. A simple project will be developed by employing these tools.

We want to create a Phonebook which will be able to do the four main functionalities (Create, Read, Update and Delete). In order to create the web application, we should follow the following steps:

Note: details are presented in the Online session.

- 1) Create an “ASP.NET Core Web Application (Model-View-Controller)” in Microsoft Visual. Next, write the name of your application, “PhonebookProject”.

In addition, the other settings should be similar to the following image.



Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Framework ⓘ
.NET 6.0 (Long-term support)

Authentication type ⓘ
None

☐ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

Docker OS ⓘ
Linux

☐ Do not use top-level statements ⓘ

Back Create

2) Create a Model (class) called Phonebook in the Models folder with the following properties.

```
public class Phonebook
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
    public string HomeNumber { get; set; }
    public string MobileNumber { get; set; }
    public string WorkNumber { get; set; }
    public string EmailAddress { get; set; }
}
```

3) In the Controllers folder, create a controller (MVC Controller - Empty) called "PhonebookController".

Add the following namespace at the top of the controller

```
using PhonebookProject.Models;
```

then write the following codes in the Index method.

```
List<Phonebook> items = new List<Phonebook>();
Phonebook item = new Phonebook();
item.Id = 1;
item.FirstName = "User 1";
item.LastName = "AAA";
item.MobileNumber = "089123456";
item.EmailAddress = "UA@mu.ie";
items.Add(item);

item = new Phonebook();
item.Id = 2;
item.FirstName = "User 2";
item.LastName = "BBB";
item.MobileNumber = "08900000";
item.EmailAddress = "UB@mu.ie";
items.Add(item);

return View(items);
```

4) Create a folder named "Phonebook" in the Views folder.

5) In the Phonebook folder in the Views folder, create a view with the following features:

- Select Razor View
- Set the name to Index
- Select List in the Template
- Select Phonebook as Model class
- Uncheck "Create as a partial view"
- Select "Use a layout page"
- Put this address as the layout page: "~/Views/Shared/_Layout.cshtml"
- Finally, click on the add button.

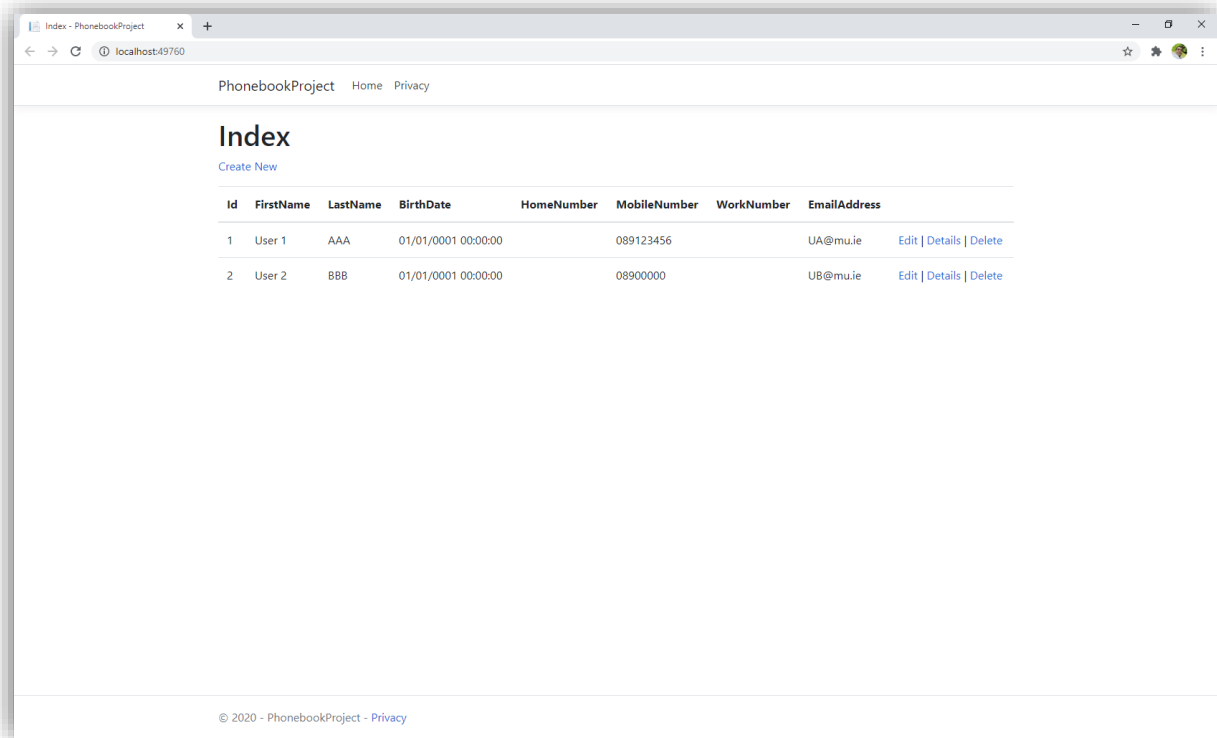
6) Open the Program.cs file, and at the end of the file, change the following line

```
pattern: "{controller=Home}/{action=Index}/{id?}");
```

To

```
pattern: "{controller=Phonebook}/{action=Index}/{id?}");
```

Now, if you run your application, your output should be similar to the following image.



However, none of the links (Create New, Edit, Details, and Delete) is working. To active them, we should create actions in PhonebookController for them. Before that, we want to work with a database instead of hardcoded for information. We want to use Entity Framework as a high-level interface to work with a database in this example. The rest of the steps are working with a database.

7) Open "Package Manager Console" from Tools-> NuGet Package Manager and write the following statement to install the last version of Entity Framework

- Install-Package EntityFramework

8) In the Models folder, create a class called AppDbContext, which drives from DbContext.

First, add the following namespace to the top of the file

```
using System.Data.Entity;
```

then, change the class to become similar to the following codes

```
public class AppDbContext:DbContext
{
    public DbSet<Phonebook> phonebooks { get; set; }
    public AppDbContext():base("DefaultConnection")
    {
    }
}
```

9) In “Package Manager Console”, write the following statements to create the database in SQL Server.

- enable-migrations
- add-migration <a name for the migration>
- update-database

10) To seed the database, we should add a new migration, then add the required SQL statements in UP or Down methods, and finally update the database to affect the changes in the database. Please follow the following steps:

10-1) run Add-migration <name> in “Package Manager Console”

10-2) find and open the migration with the name that you created in the previous step and write the following codes in the UP method (Note: there should be folder named migrations, and all the migrations are automatically generated there).

```
public override void Up()
{
    Sql("INSERT INTO Phonebooks (FirstName, LastName, BirthDate, HomeNumber, MobileNumber, WorkNumber, EmailAddress) VALUES ('Jack', 'Phillips', '02-03-1990', '08955555', '', '01369258', 'Jack.Phillips@gmail.com')");
    Sql("INSERT INTO Phonebooks (FirstName, LastName, BirthDate, HomeNumber, MobileNumber, WorkNumber, EmailAddress) VALUES ('Lee', 'Holland', '05-04-2000', '089333333', '00353199999', '0177777', '')");
    Sql("INSERT INTO Phonebooks (FirstName, LastName, BirthDate, HomeNumber, MobileNumber, WorkNumber, EmailAddress) VALUES ('Robin', 'Walker', '10-06-1995', '08111111', '01888888', '+353133333', 'Robin.Walker@gmail.com')");
    Sql("INSERT INTO Phonebooks (FirstName, LastName, BirthDate, HomeNumber, MobileNumber, WorkNumber, EmailAddress) VALUES ('Mathew', 'Edwards', '09-01-2005', '089765432', '', '+01321456', '')");
}
```

10-3) run the update-database in “Package Manager Console”.

11) Open the PhonebookController and declare a class variable like this:

```
private AppDbContext _dbContext;
```

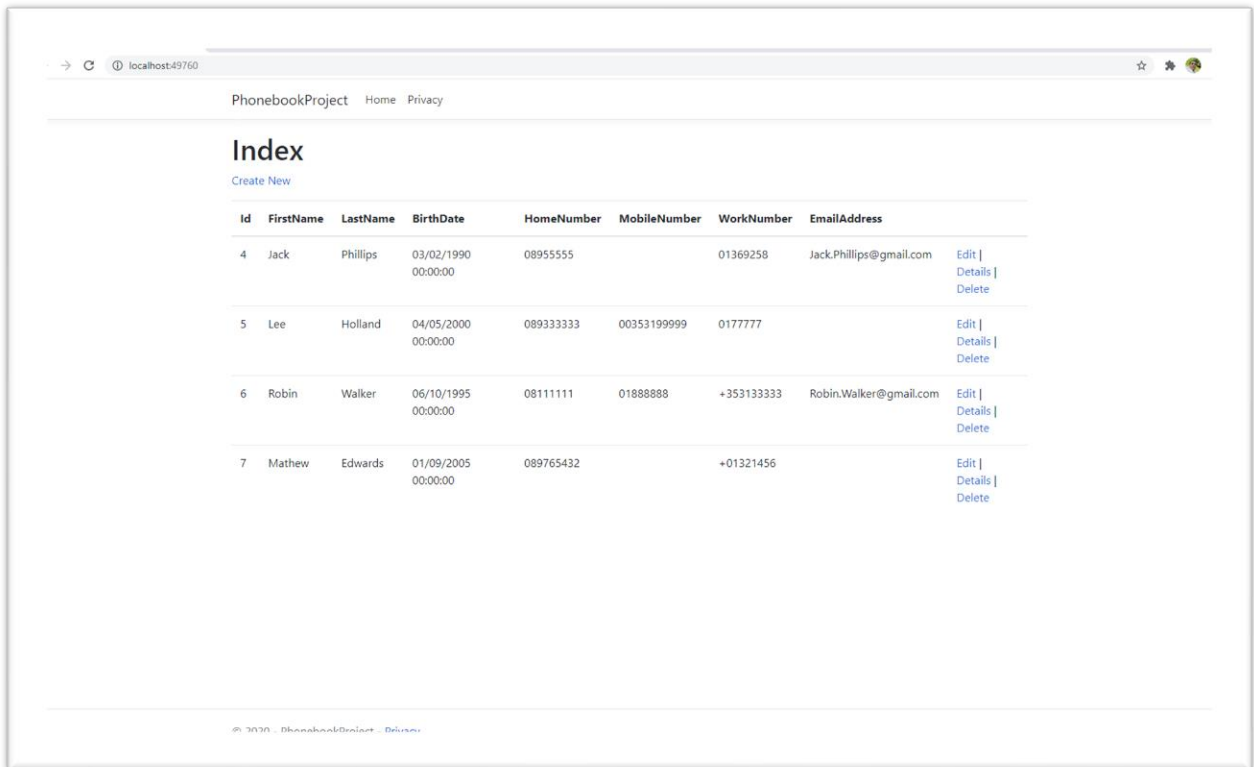
Then, create a constructor like the following for the PhonebookController class

```
public PhonebookController()
{
    _dbContext = new AppDbContext();
}
```

12) Open the PhonebookController and change the Index method to become similar to the following codes:

```
public IActionResult Index()
{
    List<Phonebook> items = _dbContext.phonebooks.ToList();
    return View(items);
}
```

Now, if you run your project, it should read information from the database and show it. The output should be similar to the following image.



13) We want to add some annotations to our Phonebook model. First, open the class and change the file according to the following steps.

13-1) Add the DataAnnotations namespace like this

`using System.ComponentModel.DataAnnotations;`

13-2)

Change the class to be similar to the following code.

```
public class Phonebook
{
    public int Id { get; set; }
    [Required]
    [StringLength(50)]
    public string FirstName { get; set; }
    [Required]
    [StringLength(50)]
    public string LastName { get; set; }

    [Display(Name = "Date of Birth")]
    public DateTime BirthDate { get; set; }

    [Display(Name = "Home Number")]
    public string HomeNumber { get; set; }

    [Display(Name = "Mobile Number")]
    public string MobileNumber { get; set; }
```

```

[Display(Name = "Work Number")]
public string WorkNumber { get; set; }

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

```

13-3) In “Package Manager Console” write the following statements to update the database.

- add-migration <a name for the migration>
- update-database

14) Inserting to Database

14-1)

Make an action named Create in PhonebookController like this

```

public IActionResult Create()
{
    return View();
}

```

14-2) Create a view in the Views->Phonebook folder with the following features:

- Select Razor View
- Set the name to Create
- Select Create in the Template
- Select Phonebook as Model class
- Uncheck “Create as a partial view”
- Select “Use a layout page”
- Put this address as the layout page: “~/Views/Shared/_Layout.cshtml”
- Finally, click on the add button.

If you run your project and then click on Create New at the top left, you should navigate to Create a page which should be similar to the following image

PhonebookProject Home Privacy

Create

Phonebook

Id

FirstName

LastName

BirthDate

HomeNumber

MobileNumber

WorkNumber

EmailAddress

[Create](#)

[Back to List](#)

© 2020 - PhonebookProject - [Privacy](#)

14-3) Create another action with the name “Create” with the following codes:

```
[HttpPost]
public ActionResult Create(Phonebook phonebook)
{
    _dbContext.phonebooks.Add(phonebook);
    _dbContext.SaveChanges();
    return RedirectToAction("Index", "Phonebook");
}
```

If you run your code, the create new form should add the new person to the database.

15) Updating

15-1)

At the end of the Index.cshtml file, change the following line

```
@Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ })|
To
```

```
@Html.ActionLink("Edit", "Edit", new { id=item.Id })|
```

15-2)

Create an action for Edit similar to this:

```
public ActionResult Edit(int id)
{
    Phonebook phonebook = _dbContext.phonebooks.SingleOrDefault(t => t.Id == id);
    if (phonebook == null)
        return StatusCode(404);
    else
        return View("Create", phonebook);
}
```

15-3) Change the action created in 14-3 to the following codes:

```
public ActionResult Create(Phonebook phonebook)
{
    if (phonebook.Id == 0)
        _dbContext.phonebooks.Add(phonebook);
    else
    {
        Phonebook phonebookInUpdate = _dbContext.phonebooks.SingleOrDefault(t => t
.Id == phonebook.Id);
        phonebookInUpdate.FirstName = phonebook.FirstName;
        phonebookInUpdate.LastName = phonebook.LastName;
        phonebookInUpdate.BirthDate = phonebook.BirthDate;
        phonebookInUpdate.MobileNumber = phonebook.MobileNumber;
        phonebookInUpdate.HomeNumber = phonebook.HomeNumber;
        phonebookInUpdate.WorkNumber = phonebook.WorkNumber;
        phonebookInUpdate.EmailAddress = phonebook.EmailAddress;
    }
    _dbContext.SaveChanges();
    return RedirectToAction("Index", "Phonebook");
}
```

15-4)

Add the following code in the Create.cshtml view.

@Html.HiddenFor(t=>t.Id)

It can be added at the end of the file before the following lines:

```
<div class="form-group">
    <input type="submit" value="Create" class="btn btn-primary" />
</div>
```

16) Details

16-1)

At the end of the Index.cshtml file, change the following line

@Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
To

@Html.ActionLink("Details", "Details", new { id=item.Id }) |

16-2)

Create an action for Details similar to this

```
public ActionResult Details(int id)
{
    Phonebook phonebook = _dbContext.phonebooks.SingleOrDefault(t => t.Id == id);
    return View(phonebook);
}
```

16-3) Create a view in the Views->Phonebook folder with the following features:

- Select Razor View
- Set the name to Details
- Select Details in the Template
- Select Phonebook as Model class
- Uncheck "Create as a partial view"

- Select “Use a layout page”
- Put this address as the layout page: “~/Views/Shared/_Layout.cshtml”
- Finally, click on the add button.

If you run your code, the details links in should work correctly.

17) Delete

17-1)

At the end of the Index.cshtml file, change the following line

```
@Html.ActionLink("Delete", "Delete", new { /*id=item.PrimaryKey */ })
```

To

```
@Html.ActionLink("Delete", "Delete", new { id=item.Id })
```

17-2)

Create an action for Delete similar to this

```
public ActionResult Delete(int id)
{
    Phonebook phonebook = _dbContext.phonebooks.SingleOrDefault(t => t.Id == id);
    _dbContext.phonebooks.Remove(phonebook);
    _dbContext.SaveChanges();
    return RedirectToAction("Index", "Phonebook");
}
```

If you run your project, all the features should work correctly.