

ASP.NET MVC Apps with Databases

MVC Apps with Entity Framework and SQL Server.
Implementing CRUD Operations with Scaffolding



CREATE

C



READ

R



UPDATE

U



DELETE

D



SoftUni Team
Technical Trainers



SoftUni

Software University

<https://softuni.bg>

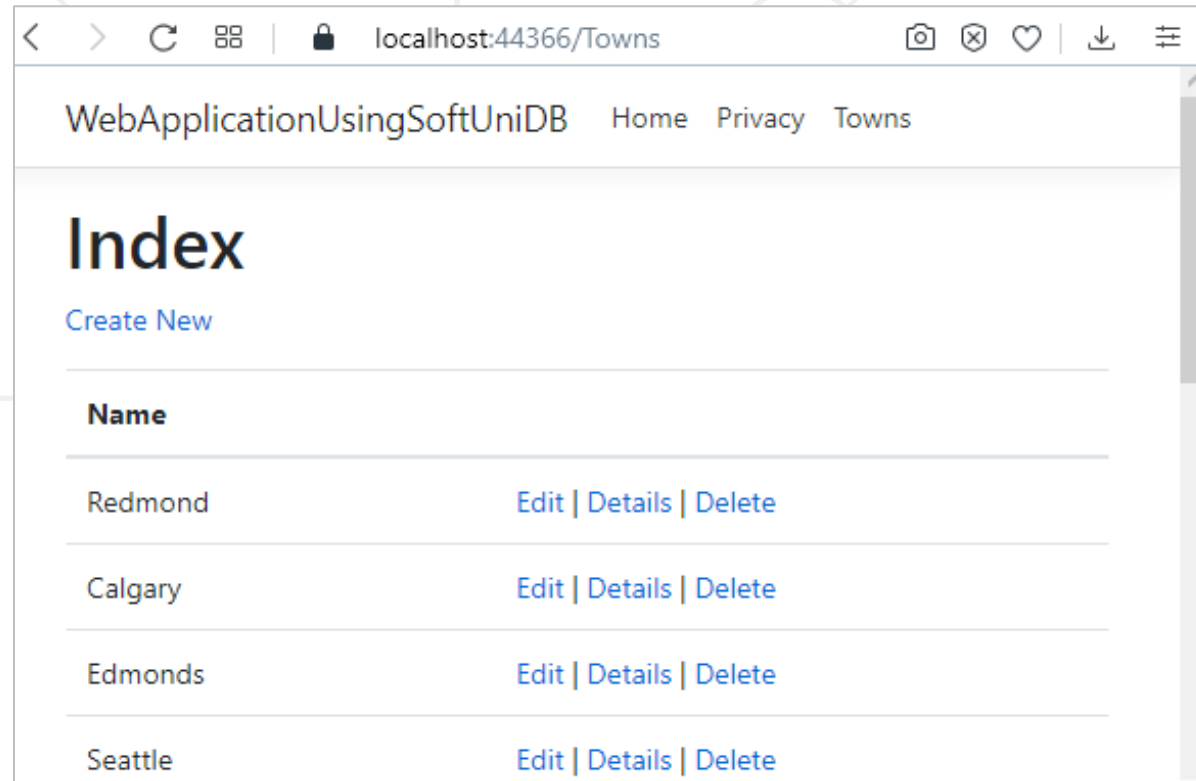
1. Implementing **CRUD operations** in ASP.NET MVC + Entity Framework + SQL Server

- Creating ASP.NET Core MVC App
- Scaffolding **Entity Framework** data classes
- Scaffolding **MVC Controllers** and **Views** by data class

2. CRUD **controllers** and **views**

- Inside the generated CRUD controllers
- Inside the generated CRUD views

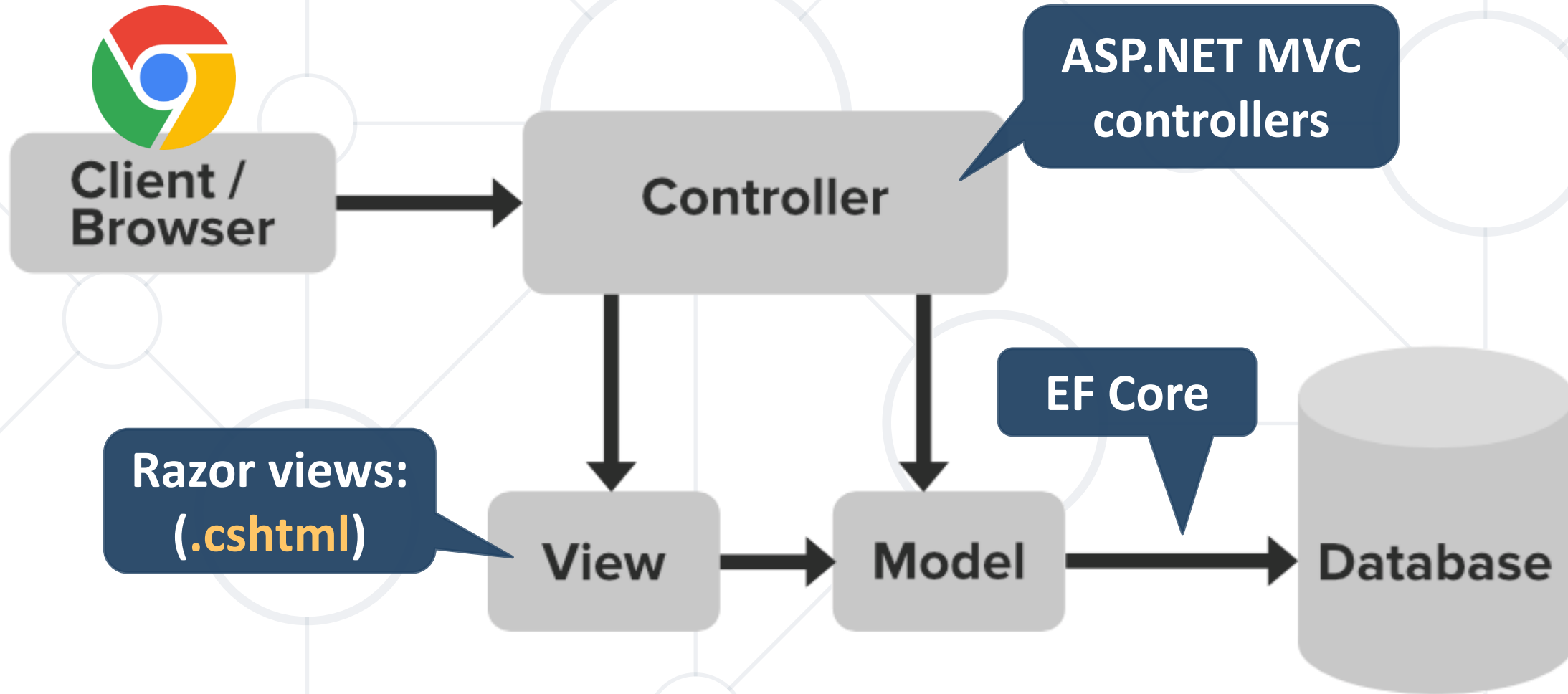




Build ASP.NET MVC Data-Driven App

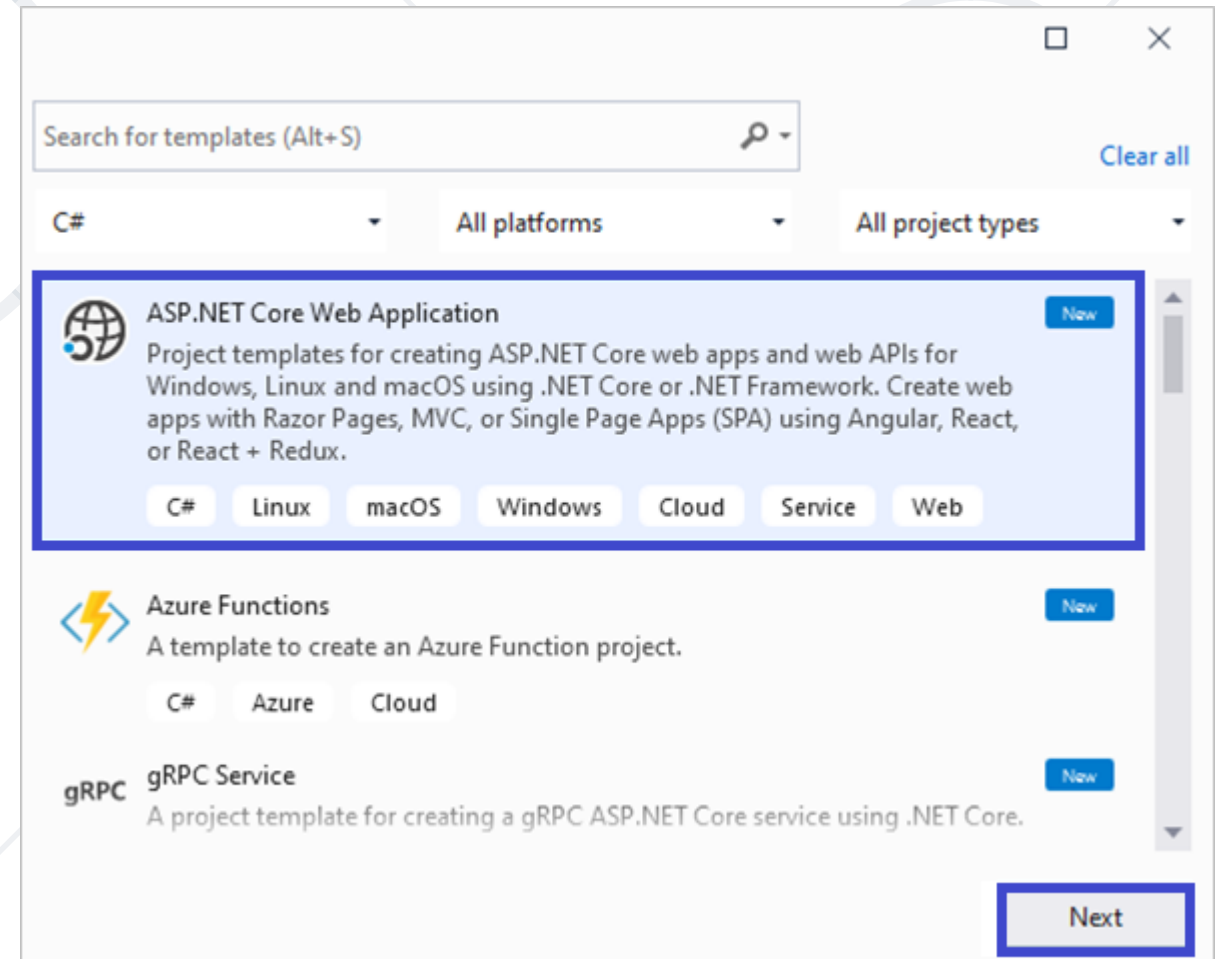
Create a CRUD Application with "SoftUni" Database

ASP.NET MVC + Entity Framework



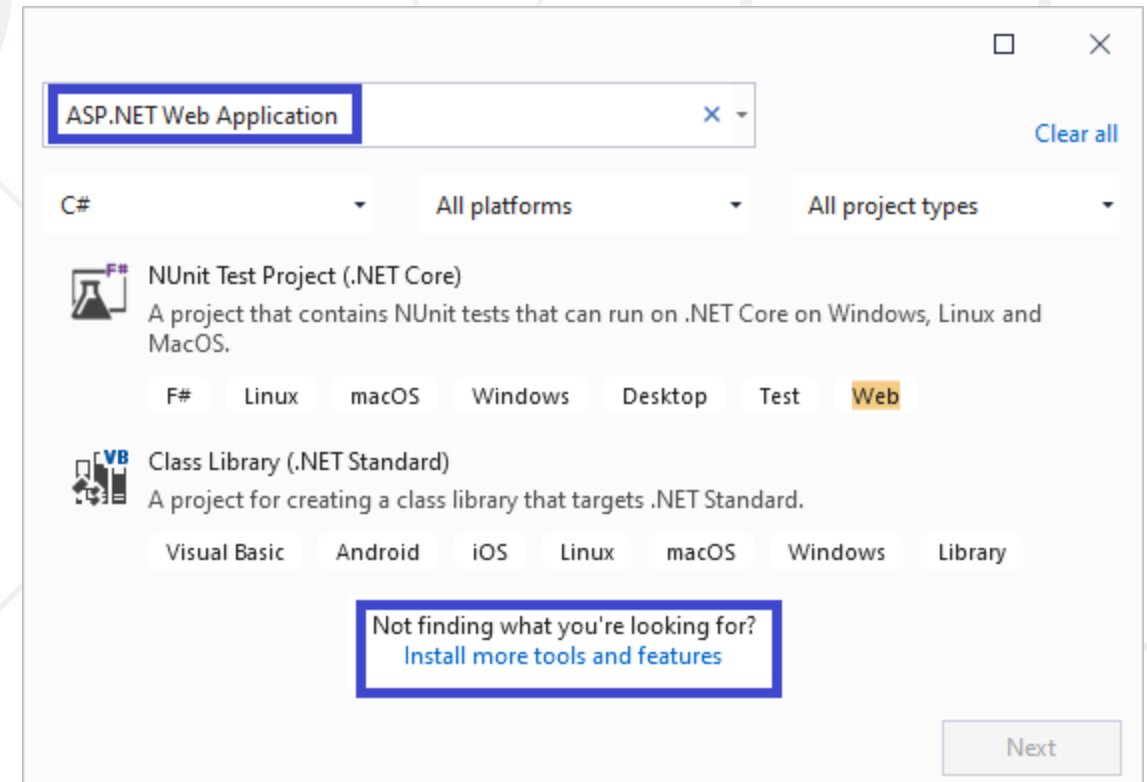
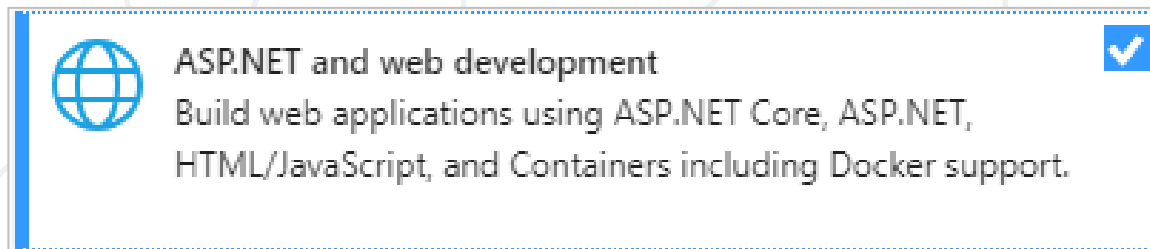
Creating an ASP.NET Core Project

- Open **Microsoft SQL Management Studio** and make sure you have the "**SoftUni**" DB
- In **Visual Studio** create a new C# Web project
 - Use the **ASP.NET Web Application (.NET Core)** template



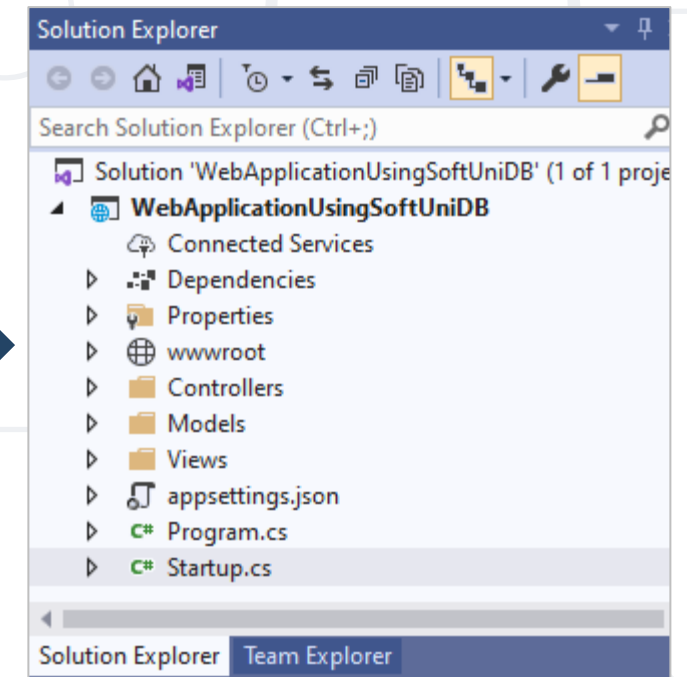
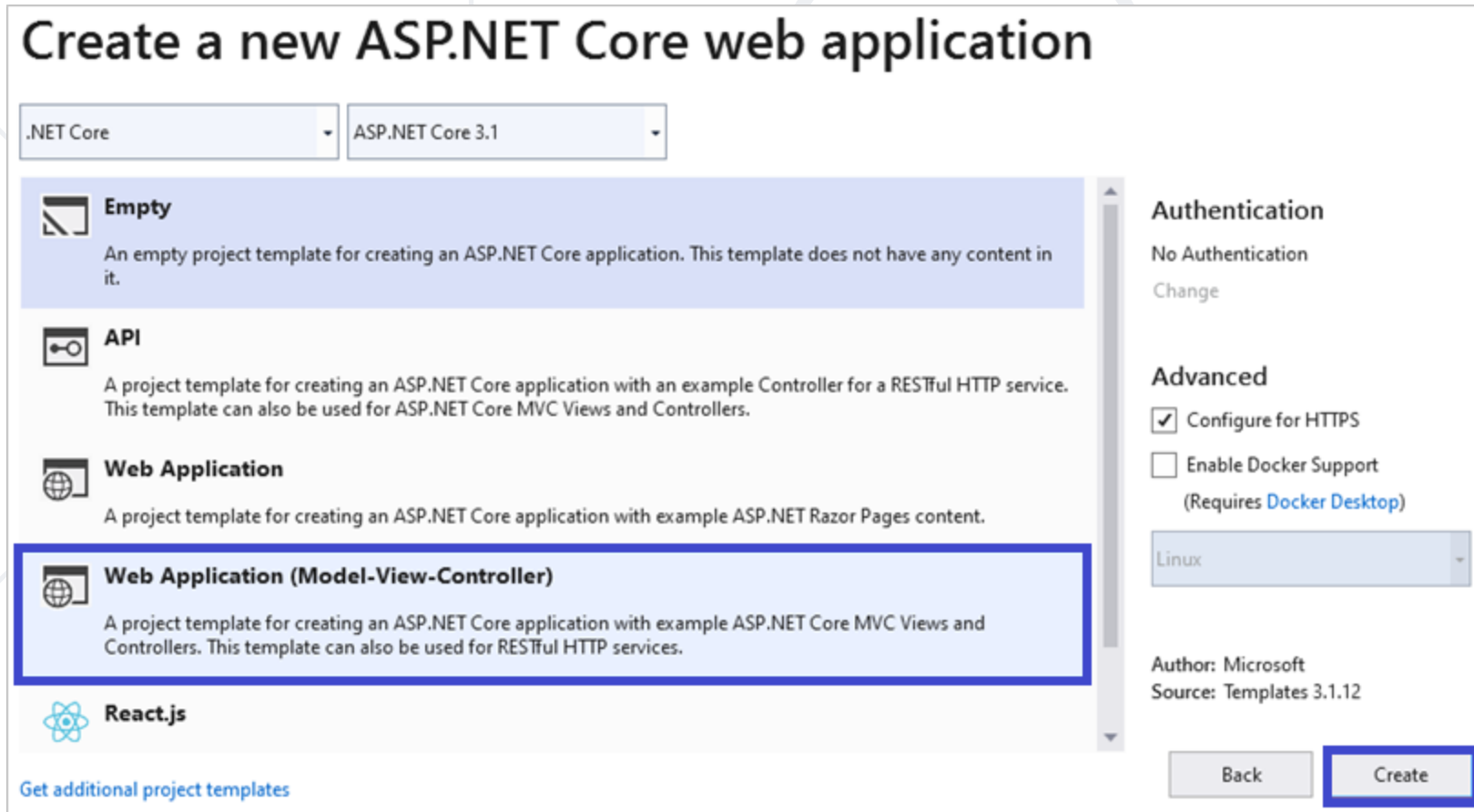
Install ASP.NET Core Web Application

- If you do not see the **ASP.NET Core Web Application** template
 - Choose the "**Install more tools and features**" link
- Install the feature "**ASP.NET and web development**"



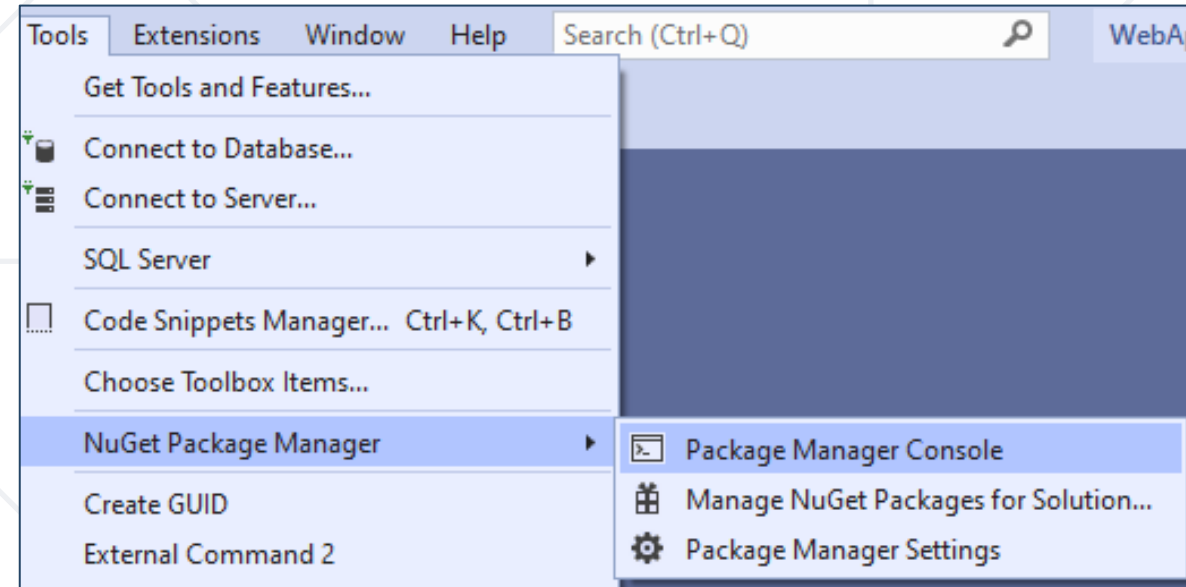
Create ASP.NET Core Web Application

- Choose the **Model-View-Controller (MVC)** project template:



Integrating Entity Framework Core Libs

- Install **EF Core** libraries
 - Go to **[Tools]** → **[NuGet Package Manager]** → **[Package Manager Console]** and **run** the following commands individually:

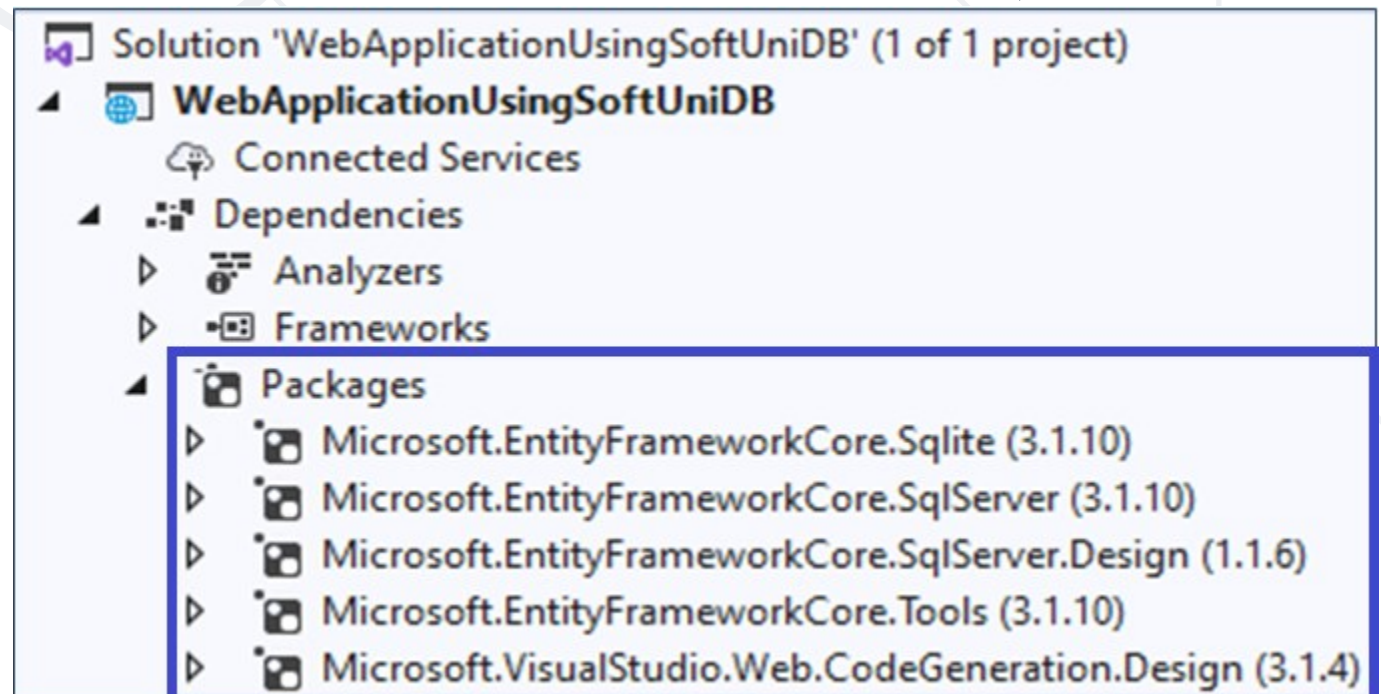
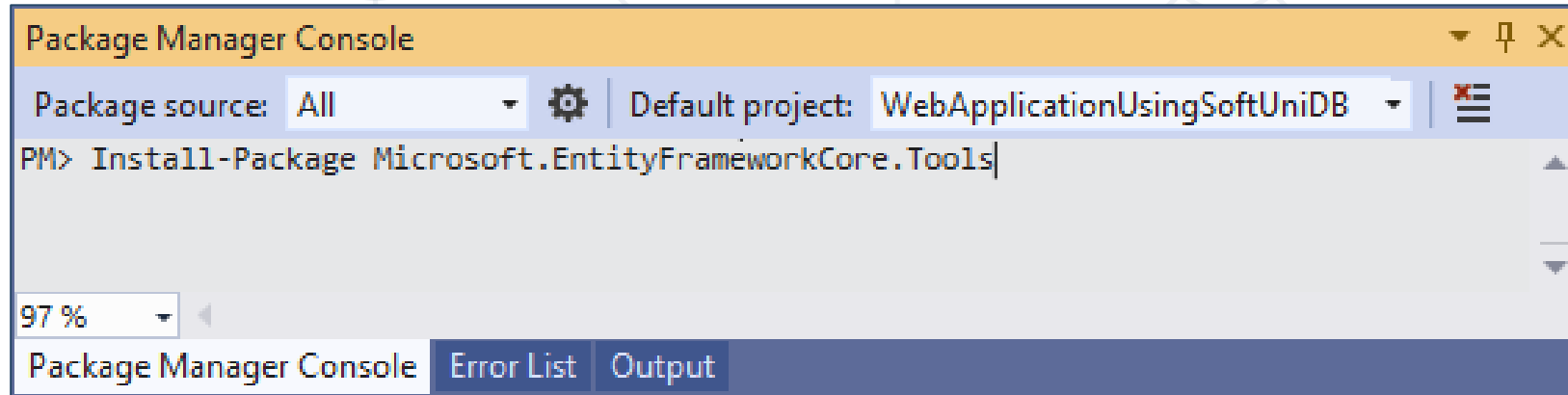


```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

```
Install-Package  
Microsoft.EntityFrameworkCore.SqlServer.Design
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```


Integrating Entity Framework Core Libs (2)

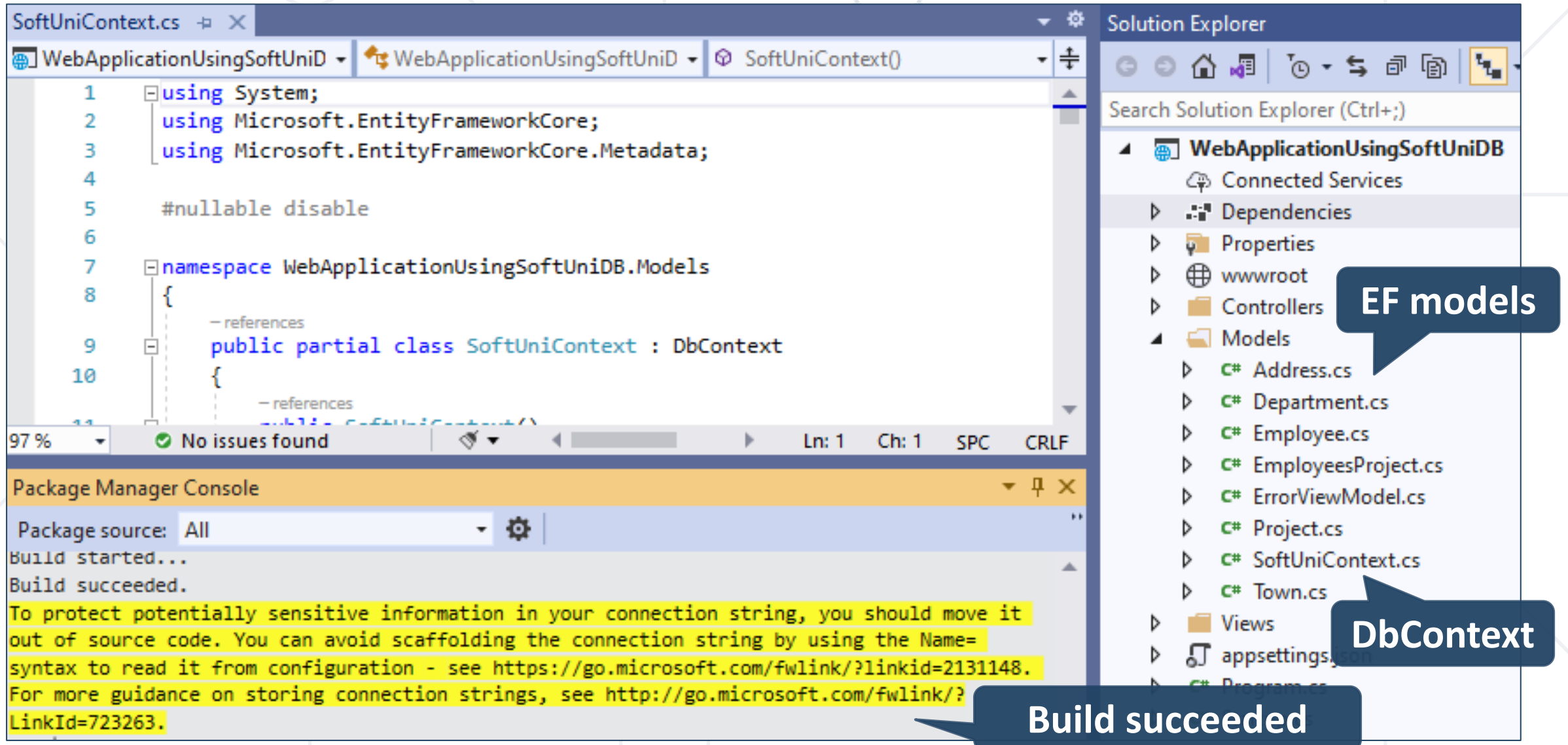


- In the **Package Manager Console** run the following command:

```
Scaffold-DbContext -Connection  
"Server=(localdb)\MSSQLLocalDB;Database=SoftUni;Integrated  
Security=True;" -Provider  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

- The command above will **generate data models and DbContext from the existing database** in the "**Models**" folder:
 - **Town.cs, SoftUniContext.cs, ...**

The "Models" Folder



The screenshot displays the Visual Studio IDE with the following components:

- Code Editor:** Shows `SoftUniContext.cs` with the following code:

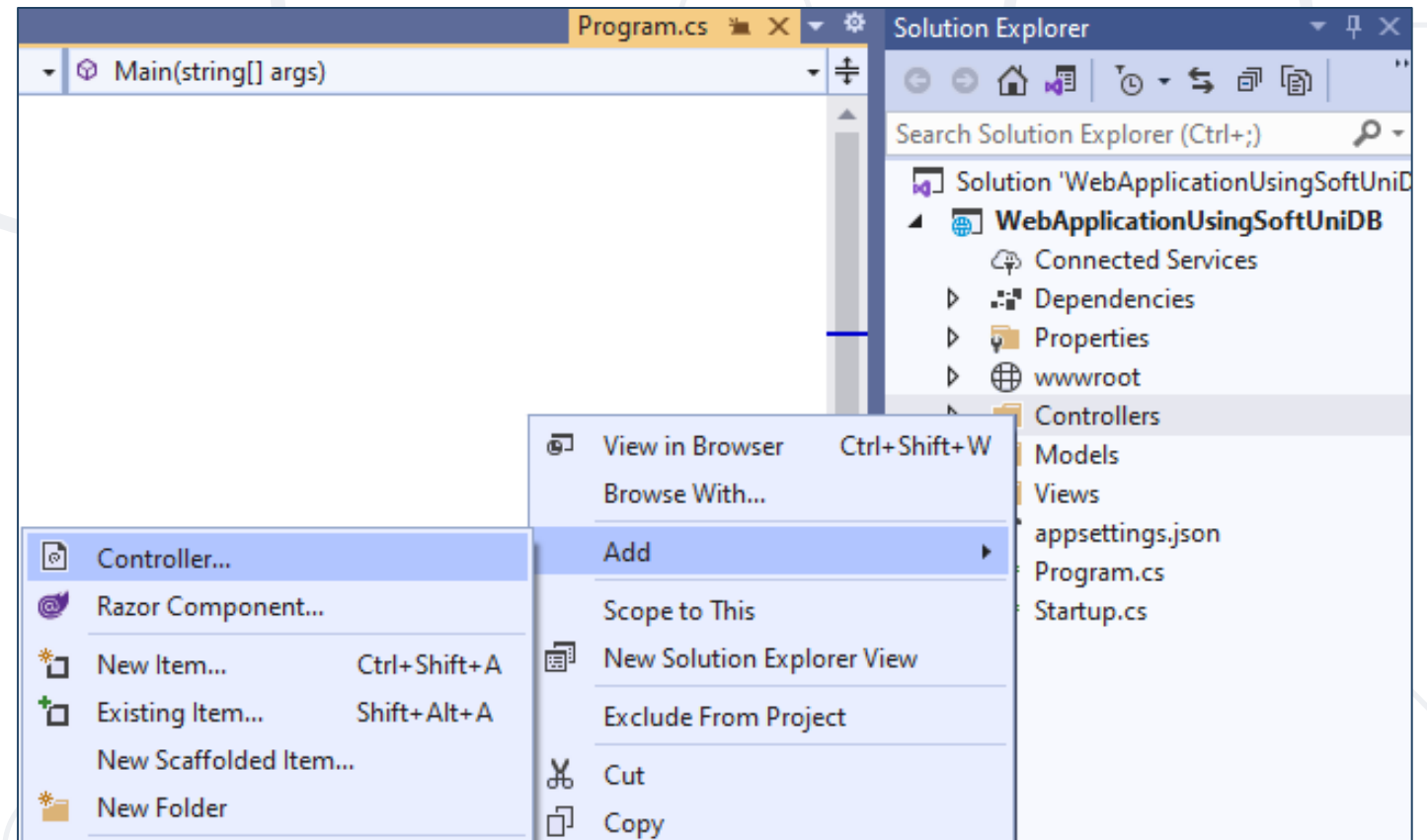
```
1 using System;
2 using Microsoft.EntityFrameworkCore;
3 using Microsoft.EntityFrameworkCore.Metadata;
4
5 #nullable disable
6
7 namespace WebApplicationUsingSoftUniDB.Models
8 {
9     // references
10     public partial class SoftUniContext : DbContext
11     {
12         // references
13     }
14 }
```
- Solution Explorer:** Shows the project structure for `WebApplicationUsingSoftUniDB`. The `Models` folder is expanded, listing files: `Address.cs`, `Department.cs`, `Employee.cs`, `EmployeesProject.cs`, `ErrorViewModel.cs`, `Project.cs`, `SoftUniContext.cs`, and `Town.cs`. A callout bubble labeled "EF models" points to this folder.
- Package Manager Console:** Shows the build output:

```
Package source: All
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it
out of source code. You can avoid scaffolding the connection string by using the Name=
syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148.
For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?
LinkId=723263.
```

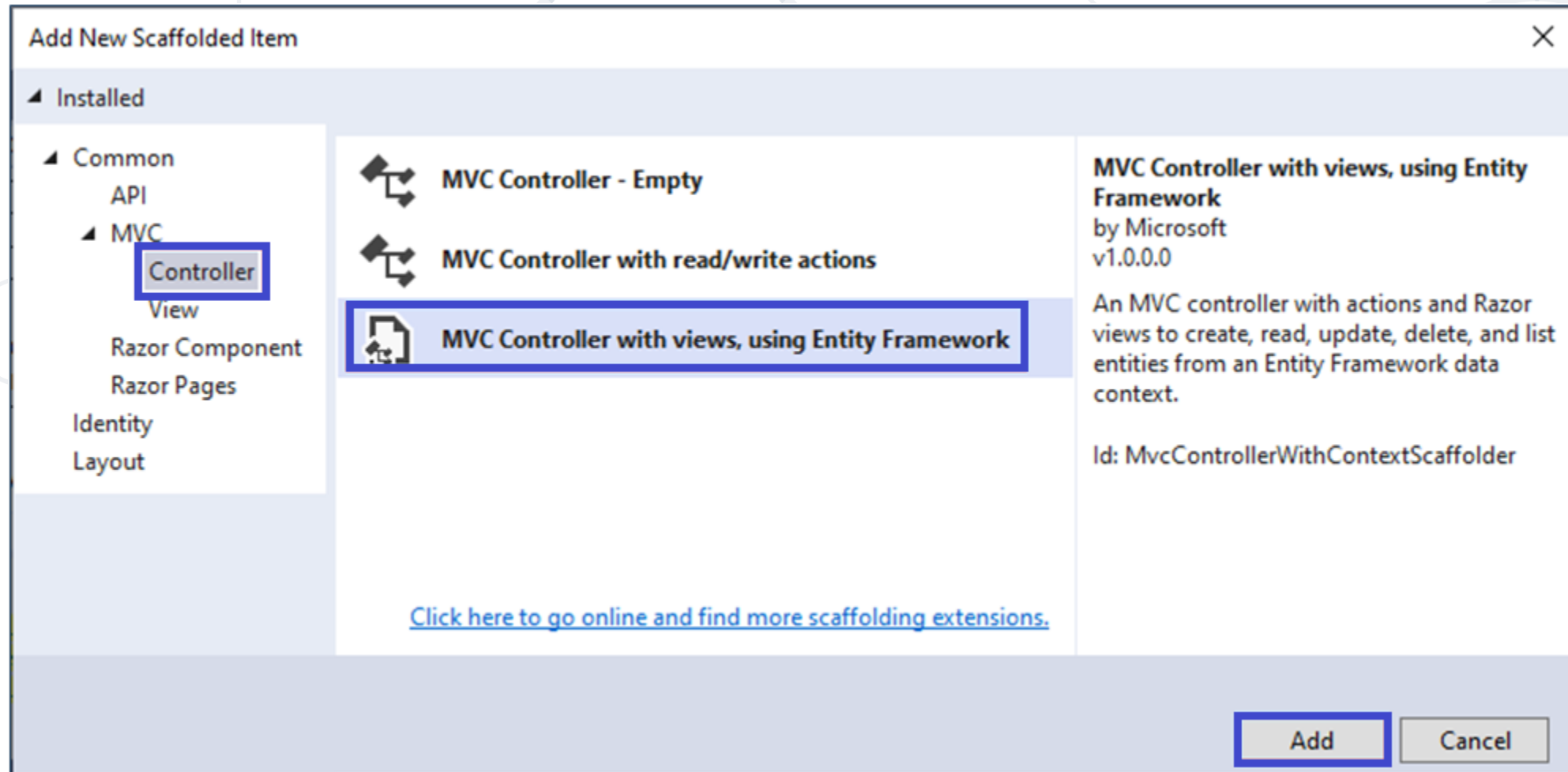
A callout bubble labeled "Build succeeded" points to the console output.
- Status Bar:** Shows "97 %", "No issues found", and "Ln: 1 Ch: 1 SPC CRLF".

Scaffolding Controller with Views

- **Create an ASP.NET Core MVC Controller** for handling data and performing basic CRUD operations
- Right-click on the **Controllers** folder in Solution Explorer
 - Select **[Add]** → **[Controller]**

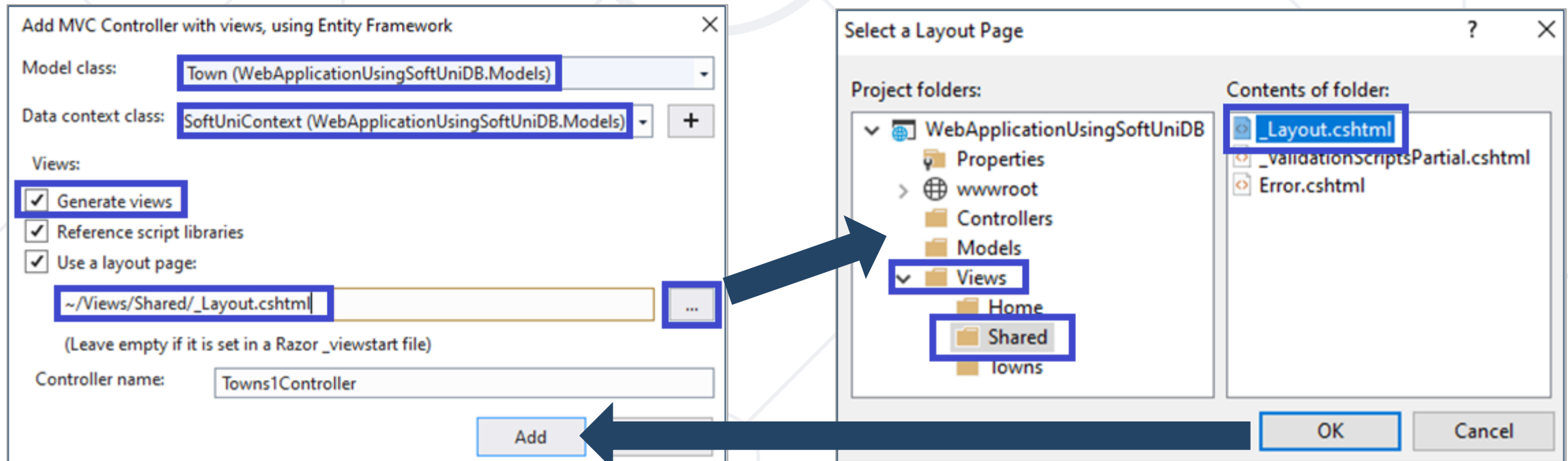


- Select **MVC Controller with Views using Entity Framework**



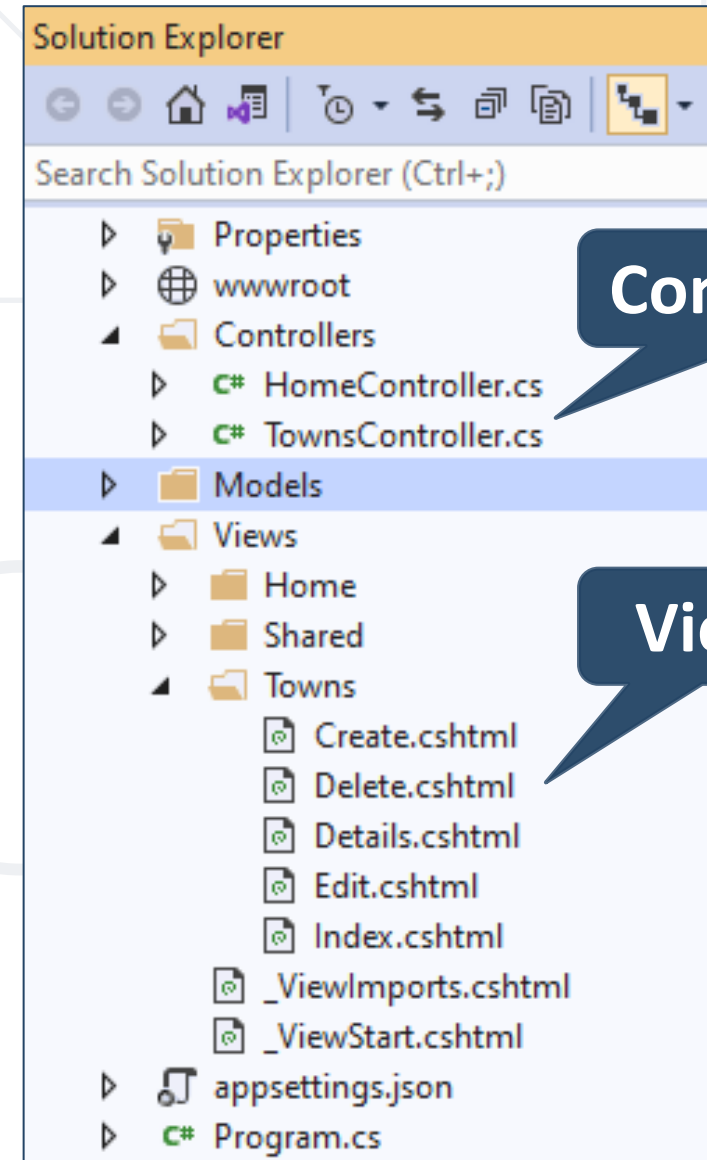
Controller with Views Using Entity Framework

- Select "**Town**" as **Model class**
- **SoftUniContext** as the **Data context class**
- Tick the **Generate views** option
- In the "**Select a Layout Page**": [Views] → [Shared] → **_Layout.cshtml**



Controller with Views

- A **view** is used to display data
 - The **Views** folder contains all the **.cshtml** view files in your app
- **Controllers** prepare data (model) for the view
 - **Views** display the data as HTML
- **Controllers** take the data from the DB using **Entity Framework**

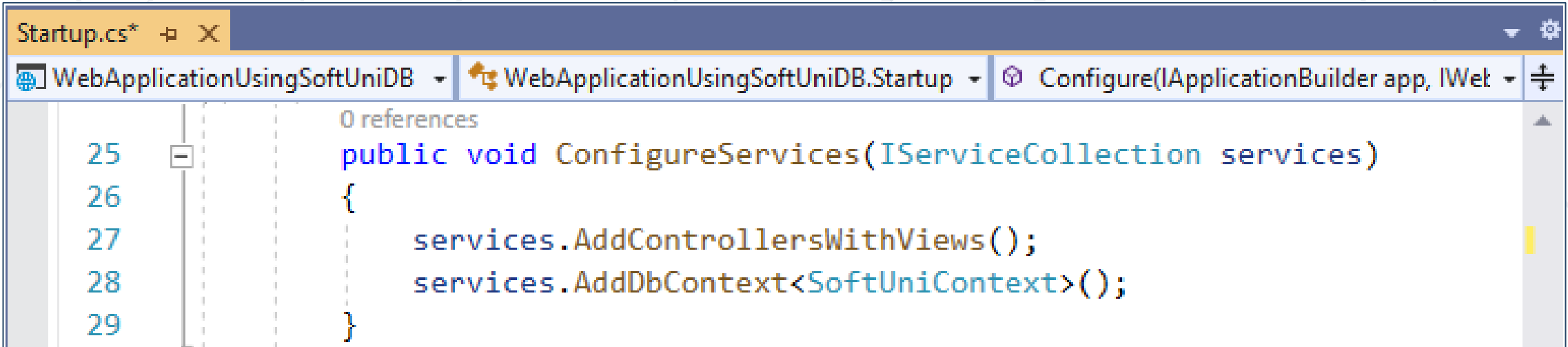


Register the DbContext in the App Startup

- **Register** your configuration class at the **ConfigureServices()** method in **Startup.cs**, by adding:

Startup.cs

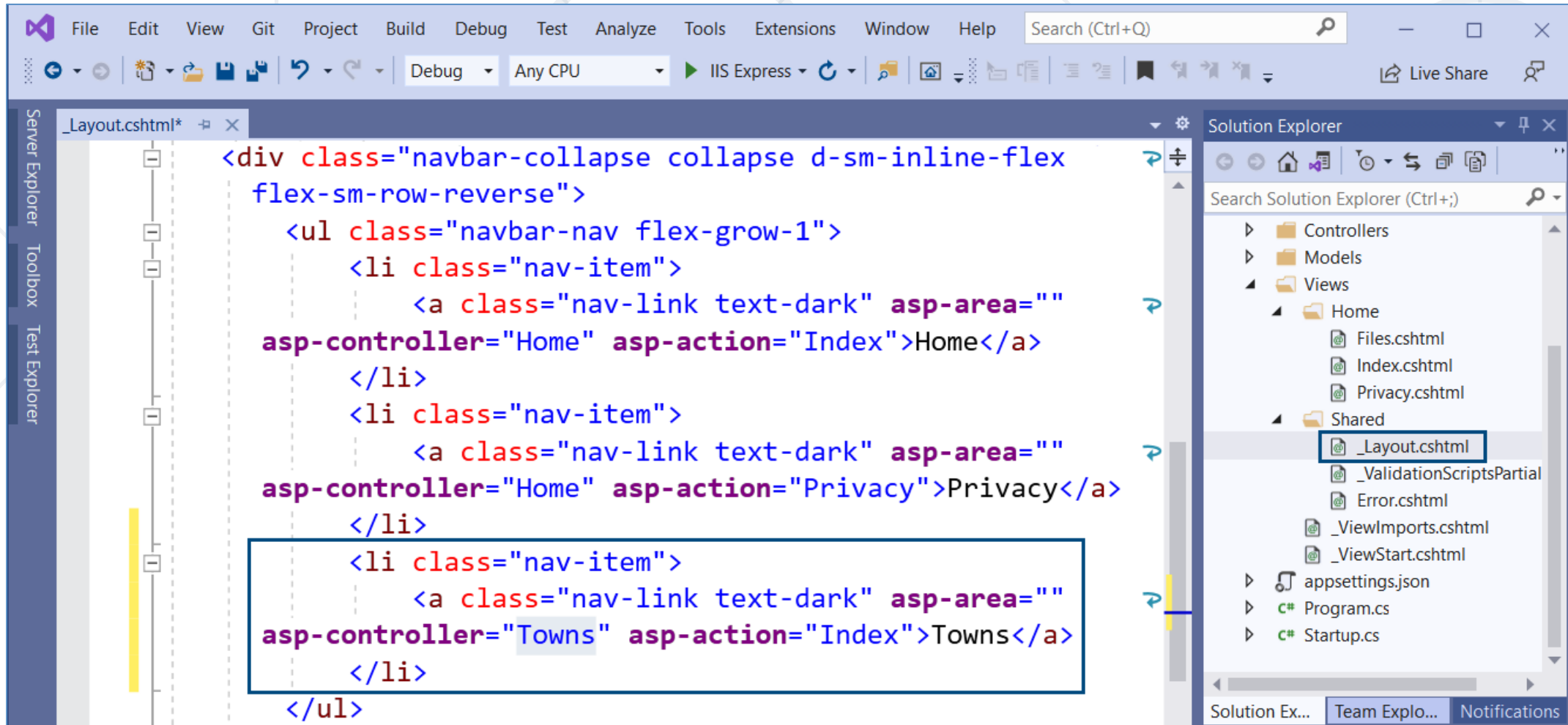
```
services.AddDbContext<SoftUniContext>();
```



```
Startup.cs*  +  X
WebApplicationUsingSoftUniDB  WebApplicationUsingSoftUniDB.Startup  ConfigureServices(IApplicationBuilder app, IWebHostEnvironment env)
0 references
25 public void ConfigureServices(IServiceCollection services)
26 {
27     services.AddControllersWithViews();
28     services.AddDbContext<SoftUniContext>();
29 }
```


Include the Towns Page in the Menu

- Edit **_Layout.cshtml** to include the "Towns" link in the app menus:

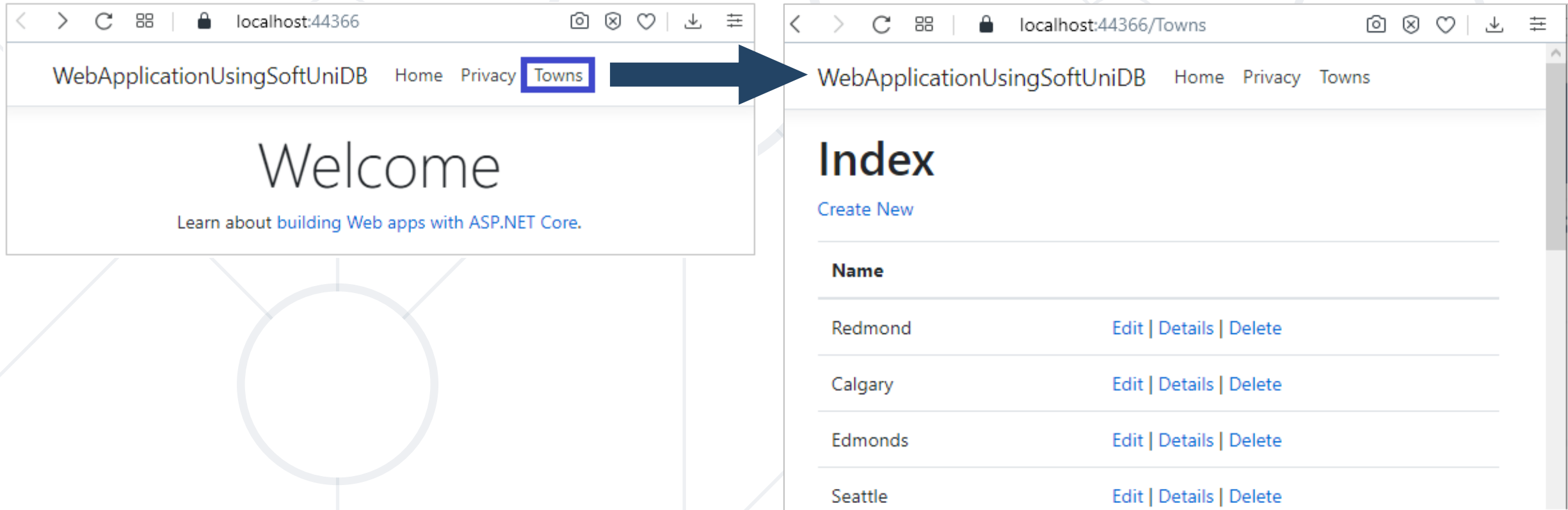


The screenshot shows the Visual Studio IDE with the `_Layout.cshtml` file open. The code defines a navigation bar with three items: Home, Privacy, and Towns. The 'Towns' item is highlighted with a blue box. The Solution Explorer on the right shows the project structure, with `_Layout.cshtml` selected under the 'Views' folder.

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <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="Towns" asp-action="Index">Towns</a>
    </li>
  </ul>
```

Testing the Application

- Now **build** and **run** the application using **[Ctrl] + [F5]**

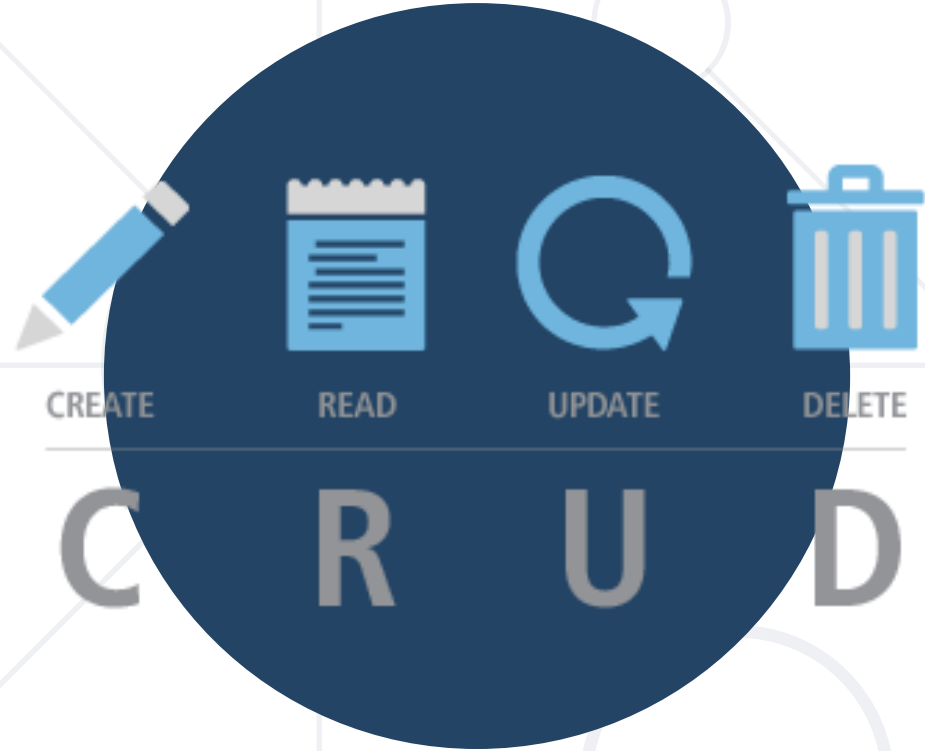


The screenshot displays two browser windows side-by-side, illustrating the application's state before and after a navigation action.

Left Window (localhost:44366): Shows the 'Welcome' page. The navigation bar includes 'WebApplicationUsingSoftUniDB', 'Home', 'Privacy', and 'Towns'. The 'Towns' link is highlighted with a blue box. Below the navigation bar, the text 'Welcome' is displayed, followed by 'Learn about building Web apps with ASP.NET Core.'

Right Window (localhost:44366/Towns): Shows the 'Index' page. The navigation bar is identical to the left window. Below the navigation bar, the text 'Index' is displayed, followed by a 'Create New' link. A table lists towns with their names and links to 'Edit', 'Details', and 'Delete'.

Name	
Redmond	Edit Details Delete
Calgary	Edit Details Delete
Edmonds	Edit Details Delete
Seattle	Edit Details Delete



CRUD in ASP.NET MVC with EF

Manipulating Databases in ASP.NET MVC

- We can retrieve an **entire collection**, using the **DbContext**:

```
// GET: /Towns/Index  
public IActionResult Index()  
{  
    return View(this._context.Towns.ToList());  
}
```

- Generated route: “**/Towns/Index**”

- **Asynchronous** version of the previous controller:

```
// GET: /Towns/Index  
public async Task<IActionResult> Index()  
{  
    return View(await _context.Towns.ToListAsync());  
}
```

- Asynchronous code is **more efficient** (load faster)

Generated CRUD Controllers (3)

- Other operations (see the generated code):

```
// GET: Towns/Details/5  
public async Task<IActionResult> Details(int? id) { ... }
```

```
// GET: Towns/Create  
public IActionResult Create() { ... }
```

```
// POST: Towns/Create  
[HttpPost] [ValidateAntiForgeryToken]  
public async Task<IActionResult> Create([Bind("TownId,  
Name")] Town town) { ... }
```

Generated CRUD Controllers (4)

- Other operations (see the generated code):

// GET: Towns/Edit/5

```
public async Task<IActionResult> Edit(int? id){...}
```

POST: Towns/Edit/5

```
[HttpPost][ValidateAntiForgeryToken]  
public async Task<IActionResult> Edit(int id,  
[Bind("TownId,Name")] Town town){...}
```

GET: Towns/Delete/5

```
public async Task<IActionResult> Delete(int? id){...}
```

- Other operations (see the generated code):

```
// POST: Towns/Delete/5  
[HttpPost, ActionName("Delete")] [ValidateAntiForgeryToken]  
public async Task<IActionResult> DeleteConfirmed(int id){...}
```


Index.cshtml

```
@foreach (var item in Model) {  
    <tr>  
        <td>@Html.DisplayFor(modelItem => item.Name)</td>  
        <td>  
            <a asp-action="Edit" asp-route-id="@item.TownId">Edit</a> |  
            <a asp-action="Details" asp-route-id="@item.TownId">Details</a> |  
            <a asp-action="Delete" asp-route-id="@item.TownId">Delete</a>  
        </td>  
    </tr>  
}
```

Generated CRUD Views (2)

Edit.cshtml

```
<form asp-action="Edit">
  <div asp-validation-summary="ModelOnly"></div>
  <input type="hidden" asp-for="TownId"/>
  <label asp-for="Name"></label>
  <input asp-for="Name"/>
  <span asp-validation-for="Name"></span>
  <div>
    <input type="submit" value="Save"/>
  </div>
</form>
```

Generated CRUD Views (3)

Details.cshtml

```
<dl class="row">
  <dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Name)
  </dt>
  <dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Name)
  </dd>
</dl>
<div>
  <a asp-action="Edit" asp-route-id="@Model.TownId">Edit</a> |
  <a asp-action="Index">Back to List</a>
</div>
```

Generated CRUD Views (4)

Delete.cshtml

```
<dl class="row">
  <dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Name)
  </dt>
  <dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Name)
  </dd>
</dl>
<form asp-action="Delete">
  <input type="hidden" asp-for="TownId" />
  <input type="submit" value="Delete" class="btn btn-danger" /> |
  <a asp-action="Index">Back to List</a>
</form>
```

Generated CRUD Views (5)

Create.cshtml

```
<form asp-action="Create">
  <div asp-validation-summary="ModelOnly"></div>
  <div>
    <label asp-for="Name"></label>
    <input asp-for="Name"/>
    <span asp-validation-for="Name"></span>
  </div>

  <div>
    <input type="submit" value="Create"/>
  </div>
</form>
```

- **ASP.NET MVC** is powerful Web development platform

- **Views** render HTML code

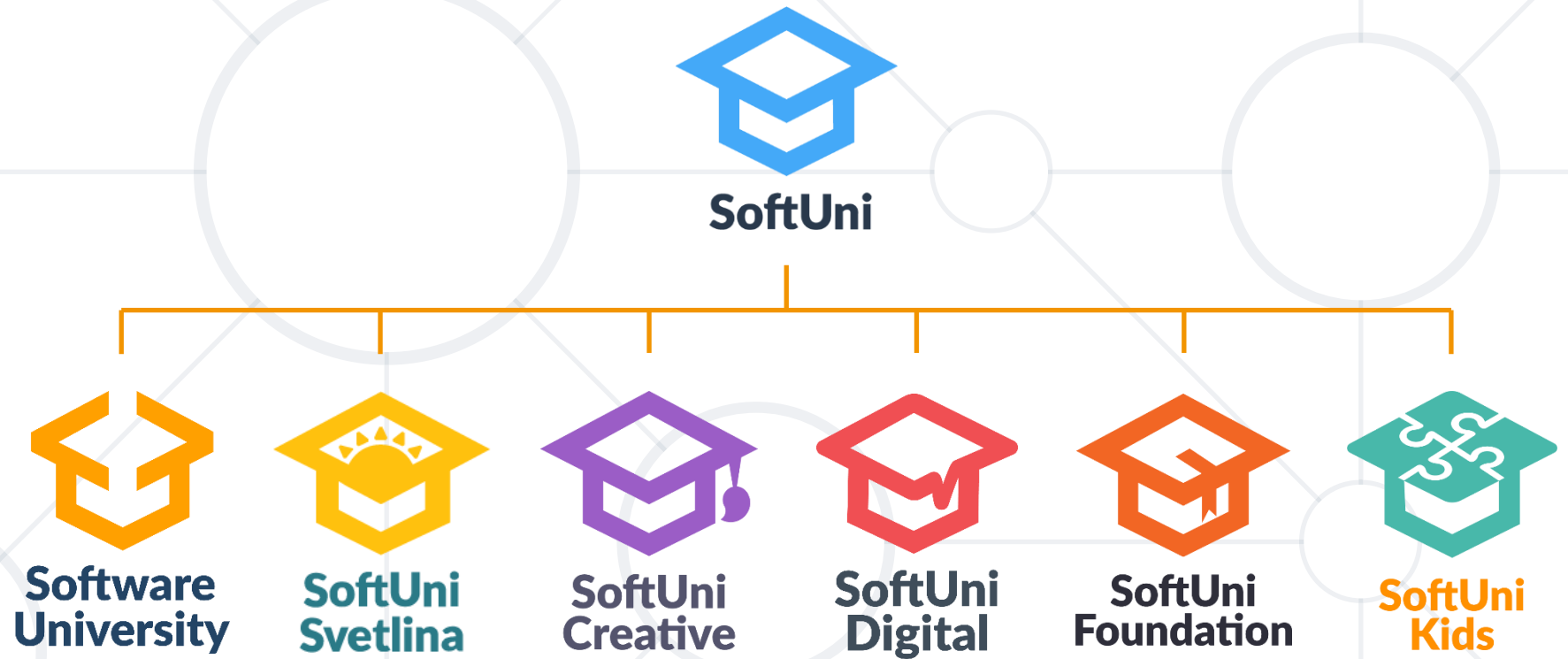
```
@foreach (var item in @Model)
{ <li>@item</li> }
```

- **Controllers** process HTTP GET / POST actions

```
public ActionResult Index()
{ return this.View(GetAllItems()); }
```

- Great integration with **databases** and **EF Core**
 - VS generates **CRUD** operations by existing model

Questions?



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, softuni.org

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg

