# Exercises: ASP.NET Core Introduction

Problems for exercises for the ["ASP.NET Core Fundamentals" course @ SoftUni](https://softuni.bg/trainings/4953/asp-net-fundamentals-may-2025)

A movie ticket and popcorn

Description automatically generated

## Final Project Demonstration

Before we start building the application, your lecturer will show you a quick demonstration of the final version of the **Cinema Web Application**.

This short preview will help you understand **what the complete project will look like** and what features you’ll be working on.

You don’t need to take notes during the demo — just **focus on getting a general idea** of the functionality.

Once the demonstration is over, we’ll begin step by step from scratch.

## Create a New MVC Project in an ASP.NET Core

Open **Visual Studio 2022**. You will see the start screen with various options to get started.

Select [Create a new project] from the list of options. This will initiate the process of creating a new project from scratch.  
On the next step, **choose** [ASP.NET Core Web App (Model-View-Controller)] as a **project template**. The steps are shown below:

**A screenshot of a computer

AI-generated content may be incorrect.**

Give a **name** to your project and solution:

A screenshot of a computer

AI-generated content may be incorrect.

On the next step you should **choose** your **target frameworok** and **authentication type.**

After that click on the [Create] **button**:

**A screenshot of a computer

AI-generated content may be incorrect.**

**Important:**We will **not use Identity at the beginning** of this course.  
You don't need to worry about login, registration, or authentication for now.  
However, it's **important to enable Identity from the start** to avoid complicated setup later.  
We will come back to Identity in a **dedicated section later in the course**.

A screenshot of a computer

AI-generated content may be incorrect.

Now your **app is created** and looks as shown below. Note that it has **folders** for **controllers**, **models** and **views** because of the template we chose:

A screenshot of a computer menu

AI-generated content may be incorrect.

If you **run the app**, you will see the **default** "Home" **page**, which is served by the HomeController in the app:

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### Understanding the Layout

#### What is \_Layout.cshtml?

In ASP.NET Core MVC, the file **\_Layout.cshtml** defines the **overall layout** (structure) of your website.  
It contains parts that are **shared across all pages**, like:

* The **<head>** section (with title, meta, CSS)
* The **navbar** at the top
* The **footer** at the bottom
* The **@RenderBody()** placeholder where each page loads its unique content

You can think of **\_Layout.cshtml** as a **master template**.

#### Where to find \_Layout.cshtml?

A screenshot of a computer

AI-generated content may be incorrect.

## Transition to the Modern Architecture Skeleton

#### From This Point Forward…

Until now, you’ve been creating your application step by step to understand how ASP.NET Core MVC works.

Starting now, we will work on a **professional template architecture** that is **pre-built** and **ready for development**. It will save you time, let you focus on building real features, and teach you how **grade applications** are structured.

#### What Is the Skeleton Project?

We’re switching to a **pre-created layered solution** that includes everything you need:

* A structured architecture
* All required Razor Views
* Identity integration
* Layer separation (Data, Services, Web, ViewModels, etc.)

#### Use It for Your Graduate Project

This architecture will be shared with you as a GitHub repository:

* <https://github.com/KrIsKa7a/ASP.NET-Core-Arch-Template/tree/main>

#### Please Take Time to Explore

Students, we strongly encourage you to:

* Open the repository and explore each project/folder
* Read the code, structure, and documentation
* Try to understand *why* it is built this way
* Follow how each layer communicates with others
* Think about how this will help your graduation project later

***This is more than just a coding exercise — it’s an opportunity to learn how professional applications are built***

#### What You’ll Do from Now On

You will no longer write views or layout code from scratch. Instead:

* Use the **predefined views and UI**
* Implement the **controllers and services**
* Bind logic to the views
* Learn advanced practices through real implementation

#### What We’ve Used So Far?

Until now, we worked in a **single-project solution** with folders like:

* 📁 Controllers/
* 📁 Models/
* 📁 Views/
* 📁 Data/
* 📄 Program.cs

This was useful for learning the basics — how MVC works, how views connect with controllers, how models are created, etc.

A screenshot of a computer

AI-generated content may be incorrect.

#### We Will Make Transition To

The solution you just opened may look big — and that’s on purpose! It’s designed to be scalable, testable and professional. But don’t worry: **you will not work on everything at once**.

The solution includes the following layers:

* CinemaApp.Data
* CinemaApp.Data.Common
* CinemaApp.Data.Models
* CinemaApp.Services.AutoMapping
* CinemaApp.Services.Common
* CinemaApp.Services.Core
* CinemaApp.IntegrationTests
* CinemaApp.Services.Core.Tests
* CinemaApp.Web.Tests
* CinemaApp.Web
* CinemaApp.Web.Infrastructure
* CinemaApp.Web.ViewModels
* CinemaApp.GCommon

#### Summary of Transition

|  |  |
| --- | --- |
| From | To (Modern Architecture) |
| One MVC project (CinemaApp) | **Multi-project solution (Data, Services, Web)** |
| Mixed folders | **Organized layers per concern** |
| Manual view creation | **Pre-built views, waiting for logic** |
| Local logic in controller | **Separated logic in services and ViewModels** |

#### Understanding the Role of the Software Architect

In every professional software project, there is a key person responsible for designing the big picture —   
the **Software Architect**

#### What Does a Software Architect Do?

A Software Architect defines **how the entire system is structured**. This includes:

* Deciding how the code is split into layers and modules
* Ensuring scalability, maintainability, and performance
* Designing how data flows between components
* Making decisions about patterns, conventions, and technology

Their work ensures that the developers can focus on implementing logic in a clean, consistent, and efficient way.

#### Why We Provide the Architecture in This Course?

**This course is not focused on teaching software architecture.**  
It is focused on helping you grow as a **developer** — someone who writes code, builds features, and implements functionality.

#### You Still Learn Architecture — by Using It

While the course does not train you to be an architect, you **gain valuable knowledge** by working inside a proper architecture.

## Starting the CinemaApp Project

In this phase, we will begin our CinemaApp project with a simple and focused approach. The goal is to help you understand the core components of an ASP.NET Core MVC application and how to implement basic functionality before we scale the project further.

#### Initial Functionality Overview

* Use the **IdentityUser** from the pre-configured **Individual Accounts** authentication
* Create the **Movie** model with basic properties
* Implement full **CRUD operations** (Create, Read, Update, Delete) for movies
* Introduce a simple **Watchlist** feature, where users can mark movies they want to watch later

#### Simplified Project Structure

To **keep the development process simple and focused**, we will begin by working only with the **essential layers** of the solution. While the original solution includes multiple projects and architectural components, we will **remove all unnecessary projects from the solution** for now. They are **not deleted, only removed** to reduce distractions during this early phase.

#### Active Projects in This Stage

* Data Layer
  + CinemaApp.Data
  + CinemaApp.Data.Models
  + CinemaApp.Data.Common
* Web Layer
  + CinemaApp.Web
  + CinemaApp.Web.ViewModels

#### Cleaning Up the Solution for Now

To keep our project focused and easy to manage in the early stages, we will **remove all the projects we don't currently need**. This will help us avoid confusion and keep the solution lightweight. Don't worry — we are **not deleting** anything from your computer, just removing the projects from the solution view in Visual Studio.

* **Open Visual Studio** and make sure your solution is loaded.A screenshot of a computer

  AI-generated content may be incorrect.
* In the **Solution Explorer**, right-click on each of the projects one by one
* Choose **"Remove"** from the context menu

A screenshot of a computer

AI-generated content may be incorrect.

* When prompted, **confirm** that you want to remove the project from the solution

#### Result

You should now see only the following projects remaining in your solution:

A screenshot of a computer

AI-generated content may be incorrect.

This is all we need to build our **first version of the application** — with **Movies**, **basic CRUD**, **authorization**, **authentication**, and **Watchlist**

#### Identity Setup – Ready but Not Used Yet

Since we selected **Individual Accounts** during project creation, ASP.NET Core automatically set up the default **Identity system** for us. This means that **registration and login functionalities are already available and fully functional**. You can run the project and try to register a new user or log in with existing credentials.

* However, for now, we **won’t use Identity features in the application logic**. Our focus at this stage is on building core functionality without any restrictions or authentication checks.

During the **ASP.NET Fundamentals** course, we will dive deeper into Identity. We’ll **scaffold and customize** the Identity views and implement proper **authentication** and **authorization** logic. Until then, just keep in mind that Identity is set up and working behind the scenes.

A screenshot of a website

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a login page

AI-generated content may be incorrect.

## Connecting the Project to SQL Server

Before we can work with data in our CinemaApp project, we need to **connect the application to a database**. This happens through a configuration file called appsettings.json

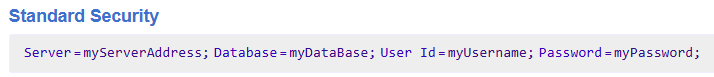
A screenshot of a computer

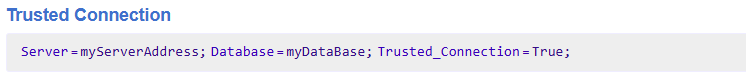
AI-generated content may be incorrect.

#### Connection String

* In the **CinemaApp.Web** project, locate the appsettings.json file. This is where we store settings like our **connection string** — which tells the application where to find the database
* Replace the default "DefaultConnection" value with your own SQL Server connection string
* If you're unsure how to format your connection string based on your SQL Server version, authentication type, or other settings, we recommend this helpful resource:

<https://www.connectionstrings.com/sql-server/>





#### Creating the Initial Migration

Now that we’ve connected the application to a database through appsettings.json,  
it’s time to create our **first migration.**

Even though we haven’t created any custom models yet, this step is still important because:

* Our project uses **Identity**
* The **Identity system** includes several built-in tables (like AspNetUsers, AspNetRoles, etc.) that need to be created in the database

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### Database is Successfully Created

You should be able to see a new database (e.g., CinemaApp-May-2025) in **SQL Server Management Studio (SSMS)**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Applying a Custom Appearance

When we create a new ASP.NET Core MVC project, the application includes only a basic appearance using the default Bootstrap layout and a site.css file.

In this section, we will improve the visual look of our CinemaApp by **creating and applying custom style files**. This will make our application look cleaner and more personalized from the beginning.

#### Add CSS Files

Create two new CSS files inside the wwwroot/css/ folder:

* cinema-styles.css
* user.css

You can create them directly from Visual Studio:

* Right-click on the **css folder** → **Add** → **New Item** → **Style Sheet**

A screenshot of a computer

AI-generated content may be incorrect.

#### Link the CSS Files in the \_Layout.cshtml

A screen shot of a computer code

AI-generated content may be incorrect.

#### Replacing the Default Layout with a Custom Cinema \_Layout.cshtml

In the previous section, we created and linked custom CSS files to define the look of our CinemaApp. Now we will complete the appearance upgrade by replacing the default layout:

* A **custom navigation bar** with icons and role-specific links
* A **footer** with a dynamic year
* All the necessary **scripts and stylesheets** to support Bootstrap and interactivity



#### Updating the \_LoginPartial View

The \_LoginPartial.cshtml file controls the right-hand side of the navigation bar. It displays either:

* A **Register** and **Login** button for anonymous users
* A **Logout** button and user info when logged in

We’ll now replace the default version with a **cleaner version styled with Bootstrap**,   
so it matches our CinemaApp layout and icon set.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



## Simplifying Password Requirements

To make **development and testing easier**, we will **reduce the default password complexity rules**. This will allow us to register users with simpler passwords, such as 123456.

**By default**, ASP.NET Core Identity **enforces strong password rules** like requiring:

* Uppercase letters, Lowercase letters, Digits, Non-alphanumeric characters, Minimum length of 6

#### Open Program.cs

A screenshot of a computer program

AI-generated content may be incorrect.

#### Update Identity Configuration

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

## Visual Feedback After Login – Navbar Changes

After registering and logging in successfully, users will see immediate changes in the **navigation bar**, thanks to the role-aware layout and \_LoginPartial.cshtml

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Viewing Registered Users in SQL Server

Now that **registration is functional**, it’s important to understand **where user data is stored and how to view it**.

ASP.NET Core Identity stores user accounts in a table called AspNetUsers in your database.

* Open **SQL Server Management Studio** (SSMS)
* Connect to your local SQL Server instance (e.g., localhost)

A screenshot of a computer

AI-generated content may be incorrect.

## Customizing the Identity Views (Login & Register)

By default, the **Login** and **Register** pages are **not visible** in your project — they are embedded in the Identity library.

In this section, we will **scaffold the Identity UI** to make these views editable, and then **simplify** them by removing unnecessary fields and elements.

#### Add the Identity UI Scaffold

**Right-click** on the Areas folder in your **CinemaApp.Web** project

A screenshot of a computer

AI-generated content may be incorrect.

Choose **Identity** -> **Add**

A screenshot of a computer

AI-generated content may be incorrect.

In the dialod, **check Register** and **check Login**. Choose your existing ApplicationDbContext.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Open each of the scaffolded .cshtml files and remove unwanted sections:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Creating a Custom Home Page

The default home page of a new ASP.NET Core MVC project is generic and not suitable for our CinemaApp. In this section, we will replace it with a **custom welcoming home page** and apply a **background image** to give the app a strong visual identity right from the start.

#### Add a Background Image

Create a new folder in wwwroot

A screenshot of a computer

AI-generated content may be incorrect.

Add your chosen background image (e.g., some-pic-bg.jpg) into the images folde

A screenshot of a computer

AI-generated content may be incorrect.



#### Create Custom Home View

**Replace the content of:**

**Views** → **Home** → Index.cshtml

with the following example code:



#### Result

A screenshot of a movie theater

AI-generated content may be incorrect.

## Creating the Movie Model

Now that our layout, Identity system, and home page are in place, we can begin developing application features. We’ll start by creating the first model: **Movie**

This model will store information about each movie in our cinema system and will be used to perform CRUD operations (**Create**, **Read**, **Update**, **Delete**) later.

#### Create the Model Class

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### Define the Properties

A screenshot of a computer program

AI-generated content may be incorrect.



## Model Validation

Now that we’ve defined the Movie model and its properties, it’s important to ensure that the data users submit is valid before it's saved to the database.

#### Defining Validation Constants

A screenshot of a computer program

AI-generated content may be incorrect.

Before applying validation to our model properties, we’ve created a special **helper class** called EntityConstants. This class is located in the CinemaApp.Data.Common project and is used to **hold all the length limits and value constraints** for our entities — starting with Movie.

#### Applying Validation with Data Annotations

ASP.NET Core provides built-in **Data Annotations** to help us validate our models.   
These annotations ensure, for example:

* required fields are not left empty
* string lengths are within range
* values like ratings are within valid numeric ranges

A screen shot of a computer code

AI-generated content may be incorrect.

#### Fluent API Configuration

While we’ve already added validation using **Data Annotations**, ASP.NET Core and EF Core also allow us to configure the model using the **Fluent API**. This approach is useful when you want to keep your entity classes clean or need more advanced configuration

We'll create a separate configuration class to apply Fluent API rules:

* Create MovieConfiguration
* Implement the IEntityTypeConfiguration**<Movie>** interface
* Register Configuration in OnModelCreating

A computer screen shot of a program

AI-generated content may be incorrect.

#### Do You Need Both Data Annotations and Fluent API?

In our project, we demonstrated both **Data Annotations** and **Fluent API** to show the two most common ways to apply validation.

However, you do **not need to use both at the same time** for the same property.

* **Data Annotations** if you prefer adding rules **directly on the model**
* **Fluent API** if you want to **keep models clean** and apply configuration separately

A screenshot of a black screen

AI-generated content may be incorrect.

## Connecting the Movie Model to the Database

Now that **we’ve created the Movie model**, we need to **tell Entity Framework Core to include it in the database**. We do this by registering the model in the **application’s DbContext** and creating a new **migration.**

#### Create a DbSet<Movie> Movies

A screen shot of a computer program

AI-generated content may be incorrect.

#### Register Configuration in DbContext - Applying All Configurations Automatically

In our CinemaAppDbContext class, we’ve added this line inside the OnModelCreating method:

A screen shot of a computer program

AI-generated content may be incorrect.

This line tells Entity Framework Core to automatically apply **all configuration classes** that implement **IEntityTypeConfiguration<T>** — like our **MovieConfiguration**

#### Create a New Migration

* Open the **Package Manager Console** in Visual Studio
  + **Tools** → **NuGet Package Manager** → **Package Manager Console**
* Make sure the **CinemaApp.Data** project is selected as the default project, then run:
  + Add-Migration AddMovieTable

A screenshot of a computer program

AI-generated content may be incorrect.

#### Apply the New Migration

* Still in the **Package Manager Console**, run:
  + Update-Database

A screenshot of a computer program

AI-generated content may be incorrect.

#### Verifying the Movie Table in SQL Server

After applying the migration, **Entity Framework Core creates the necessary table**(s) in the connected **SQL Server** database. In this step, we will **verify that the Movies table exists** and contains the correct columns.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Seeding the Database with Movies



Before we build the UI to manage movies, let’s add some **initial movie records** to the database. This helps us test the layout and functionality with real data.

#### Seeding Movies Using MoviesConfiguration

* Create the Configuration File

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.



#### Create and Apply the Migration

A screenshot of a computer program

AI-generated content may be incorrect.

#### Verify the Data

A screenshot of a computer

AI-generated content may be incorrect.

## Creating the MovieController and Index View

We’re now ready to **display the list of movies stored in the database**. In this section, we’ll **create a controller** and implement the **Index action** to **fetch all movies** and show them in a simple table.

#### Create the Controller

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### Add Dependencies and Constructor

A screen shot of a computer

AI-generated content may be incorrect.

#### Creating the Movie View Folder and Index View

ASP.NET Core MVC follows **naming conventions** to connect controllers and views. If a controller is named MovieController and you have an action named Index(), the framework will look for a Razor View at:

* **Views** → **Movie** → Index.cshtml

A screen shot of a computer

AI-generated content may be incorrect.

#### Add the Index View

A computer screen with a red box

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



#### Using a ViewModel for the Movie Index Page

To display only the required movie information in the view (e.g., title, genre, director, release date), we will create a special class called a **ViewModel**.

This separates our **presentation layer** (what the view needs) from the **data layer** (the full Movie entity)

* Create the **ViewModel**

**A screenshot of a computer program

AI-generated content may be incorrect.**

****

#### Adjust the Index Action in MovieController

Update the Index action to project data into the view model:

A screen shot of a computer program

AI-generated content may be incorrect.

## Final Result

You should now see a fully styled and responsive movie list page that displays all movies from your database.

A screenshot of a computer

AI-generated content may be incorrect.

## Workshop I – Version and Repository

At this stage of the workshop, we have completed the **initial functional version** of our CinemaApp project. This version contains the essential setup and foundational features we will build upon in the next sessions.

#### What’s Included in This Version

* **Registration, Login, and Logout** functionality using ASP.NET Core Identity
* **Seeded Movie Data** — a few movies are already added to the database
* **Movies Index View** — accessible to everyone, whether logged in or not

At this point, **no user can modify the data** — the movies are visible to all users, but CRUD operations and user-specific features like Watchlist will be added in the next workshop.

#### GitHub Repository – Workshop-I Branch

We have saved this version of the project so you can always refer to it if something breaks or if you want to compare your code with the working solution.

* **Branch Name:** Workshop-I
* **Repository:**   
  <https://github.com/KTsaneff/ASP.NET-Core-SoftUni-CSharpWeb-May-2025-CinemaApp/tree/Workshop-I>

#### To Clone This Version

You can use this command in your terminal:  
git clone -b Workshop-I https://github.com/KTsaneff/ASP.NET-Core-SoftUni-CSharpWeb-May-2025-CinemaApp.git