# Exercises: ASP.NET Core

# Separation of Concerns, Expanding Database, Extending Models Part II

Problems for exercises for the ["ASP.NET Core Advanced" course @ SoftUni](https://softuni.bg/trainings/4708/asp-net-advanced-october-2024)

A popcorn and film reels and a movie ticket

Description automatically generated with medium confidence

## Creating a Welcoming Home Page

### Set Up the Basic Structure

* In the **Home** folder under Views, open the **Index.cshtml** file
* Replace the current content with the following layout structure to **start creating a more welcoming  
   home page**A computer code on a white background

  Description automatically generated

### Add a Hero Section with Call-to-Action Buttons

* Below the introductory text, add a welcome **section with buttons to direct users to the list of   
  cinemas and movies**A screenshot of a computer

  Description automatically generated

### Add a Featured Section for Top Movies and Cinemas

* Below the hero section, you can **add a featured section that highlights popular or recommended   
  movies and cinemas**A screenshot of a computer code

  Description automatically generated

### Add a Footer Section with Additional Information

* Finally, **add a footer with a brief description** and links to social media (optional)  
  A screen shot of a computer code

  Description automatically generated

### Summary of Complete Code for the Home Page



### The Home Page

A screenshot of a computer

Description automatically generated

## Setting Up the Manager Navigation for CRUD Operations

### Introduction

The **"Manager" navigation item** will serve as a centralized portal for **CRUD operations** within our CinemaWebApp. This setup will lay the foundation for easily accessing management functionalities like adding, updating, and deleting entries. **Later**, we will enhance the structure by **dividing the application into different *areas*** for better organization.

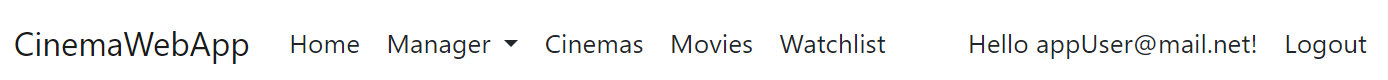
### Introducing the Manager Dropdown

* **Open \_Layout.cshtml**  
  A screenshot of a computer code

  Description automatically generated
* **Add the Manager Dropdown to the Navigation Bar**Inside the **<ul class="navbar-nav flex-grow-1">** section, add the following code snippet to create the **Manager** dropdown menu:A screenshot of a computer code

  Description automatically generated
* Ensure the **"Manager" dropdown only appears for registered** (authenticated) **users**A screen shot of a computer code

  Description automatically generated



## Setting Up the "Manage Cinemas" View for Full CRUD Control

* In the **Views/Cinema folder**, add a new **Razor view** named **Manage.cshtml**
* Set up this view to **display each cinema in a table with action buttons for CRUD operations**A screen shot of a computer code

  Description automatically generated

### Code File for the Cinema Manage View



### Add the Manage Action in CinemaController

A computer code with text

Description automatically generated with medium confidence

### Connect the Manage Button in \_Layout.cshtml

* Locate the **Manager** dropdown menu where the **"Manage Cinemas"** link **currently points to Cinema/Index**
* Change the **asp-action** to **Manage** to **point it to the new view**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

## Setting Up the "Manage Movies" View for Full CRUD Control

* In the **Views/Movie folder**, add a new **Razor view** named **Manage.cshtml**
* Set up this view to **display each movie in a table with action buttons for CRUD operations**A screenshot of a computer

  Description automatically generated

### Code File for the Movie Manage View

****

### Add the Manage Action in MovieController

A screen shot of a computer

Description automatically generated

### Connect the Manage Button in \_Layout.cshtml

* Locate the **Manager** dropdown menu where the **"Manage Movies"** link **currently points to Movie/Index**
* Change the **asp-action** to **Manage** to **point it to the new view**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

## Expanding Your Cinema Controller for Full CRUD Functionality

* **If you have worked with the lecturer so far**, your CinemaController should be well-structured, covering most essential actions
* To **complete full CRUD** (Create, Read, Update, Delete) functionality, you’ll need to add **Edit** and **Delete** actions to each controller

### Adding Edit Action in CinemaController

* **EditCinemaFormModel**
  + To create an **EditCinemaFormModel** **similar to AddCinemaFormModel**, you’ll need a model that can **accept the ID of the cinema being edited** (since the ID will be used to identify the record for updating in the database). This **model will also include the validation attributes** for Name and Location, as in AddCinemaFormModel.  
    A screenshot of a computer program

    Description automatically generated
* **Add GetCinemaEditModelByIdAsync Method in the Service Layer**
  + In the service layer, **we need a method to retrieve the cinema details by ID** and **populate the EditCinemaFormModel**. This method **will be called from the controller to load the existing data** when opening the edit form

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

* **Add UpdateCinemaAsync Method in the Service Layer**
  + The **UpdateCinemaAsync** method **will take the cinema ID and an instance of EditCinemaFormModel** **with updated data**. It **returns a bool** to indicate whether the   
    update was successful

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* **Edit Action** - will allow users to update the details of an existing cinema
  + **Edit (GET)**: Retrieves the cinema by ID and populates the form with its data

A computer code on a white background

Description automatically generated

* + **Edit (POST)**: Updates the cinema details after validation

A screenshot of a computer program

Description automatically generated

### Adding the Edit View in the Views/Cinema

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

### Code File for the Cinema Edit View



**Anti-Forgery Token**: The **@Html.AntiForgeryToken()** line adds a token to **protect against CSRF attacks**

### Update Cinema Model to Support Soft Deletion

* Add a **bool property named IsDeleted** in the Cinema model
* This property **will be set to true when a cinema is soft-deleted**

A computer code with colorful text

Description automatically generated

* Add **SoftDeleteCinemaAsync** Method to the **ICinemaService** Interface

A screenshot of a computer program

Description automatically generated

* Implement **SoftDeleteCinemaAsync** in **CinemaService**
  + In CinemaService, **implement SoftDeleteCinemaAsync to mark the cinema as deleted**. Additionally, **add a check to ensure the cinema can be safely deleted** (e.g., no active movies associated with it)

A screen shot of a computer program

Description automatically generated

* **Add Soft Delete Actions** in CinemaController
  + **GET** action for soft deleting a cinema

A screen shot of a computer program

Description automatically generated

* + **POST** SoftDeleteConfirmed Action - This action **attempts to soft delete the cinema**.   
    If **restrictions are violated**, it **redirects back to the view** with an error message

A computer screen shot of text

Description automatically generated

* **Update CinemaDetailsViewModel**
  + Add an Id property to it, so the soft delete view can access the cinema's identifier.

A screenshot of a computer program

Description automatically generated

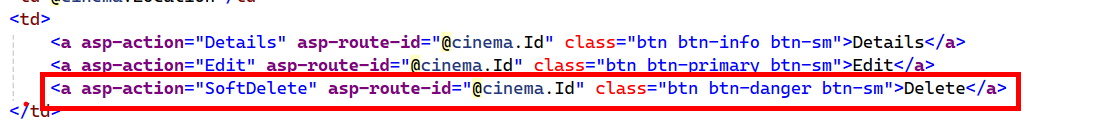
* Update **GetCinemaDetailsByIdAsync** in the **Service**
  + Ensure that the **Id is correctly populated when fetching the data** for CinemaDetailsViewModel

A screen shot of a computer code

Description automatically generated

* **Updated Manage Cinemas View**

To make the **Delete** button in your **Manage Cinemas** view redirect to the **SoftDelete** confirmation page (instead of directly deleting the cinema), we’ll update the button to use the SoftDelete action rather than a direct Delete action. This will guide users to a confirmation view where they can confirm the deletion.



* Create **SoftDelete.cshtml** Confirmation View
  + In **Views/Cinema**, **create a SoftDelete.cshtml** view to confirm deletion. This view will display the cinema’s details and ask for confirmation

A screenshot of a computer program

Description automatically generated

### Code File for the Cinema SoftDelete View

### CinemaMovieConfiguration Using Cascade Delete for the Cinema Relationship

* This setup will **automatically delete associated CinemaMovie records if a Cinema is deleted**, helping avoid database conflicts by ensuring that related mappings are also removed

A screen shot of a computer program

Description automatically generated

### Add and Apply Migration

**Create a new migration file in your project that adds the IsDeleted column to the Cinemas table.**

**A black text on a white background

Description automatically generated** **A close up of black text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

## Expanding Your Movie Controller for Full CRUD Functionality

* **If you have worked with the lecturer so far**, your **MovieController** should be well-structured, covering most essential actions
* To **complete full CRUD** (Create, Read, Update, Delete) functionality, you’ll need to add **Edit** and **Delete** actions to each controller

### Adding Edit Actions in MovieController

* **EditMovieFormModel**
  + How to Use **AddMovieInputModel** as a Basis for **EditMovieFormModel**
  + Since **EditMovieFormModel** will be similar, we can modify **AddMovieInputModel** slightly to make it fit for the Edit functionality. Here are the key differences to note:
    - Include an Id Property
    - Remove Default Date Assignment
    - Mapping Configuration
  + Final EditMovieFormModel Implementation:  
     
* Implement the **GetMovieEditModelByIdAsync** and **UpdateMovieAsync** methods in **MovieService**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* **Add Edit Actions in MovieController**

A computer code with text

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

### Create the Edit View

Now, let’s create the **Edit.cshtml** view in the **Views/Movie folder**. This view will be **similar to the Add view** but with fields pre-filled with the movie’s current details

A screenshot of a computer program

Description automatically generated

### Code File for the Movie Edit View



### Update Movie Model to Support Soft Deletion

* Add a **bool property named IsDeleted** in the **Movie** model
* This property **will be set to true when a movie is soft-deleted**

A screenshot of a computer program

Description automatically generated

* Add **SoftDeleteMovieAsync** Method to the **IMovieService** Interface

A screenshot of a computer program

Description automatically generated

* Implement **SoftDeleteMovieAsync** in **MovieService**
  + In MovieService, **implement SoftDeleteMovieAsync to mark the movie as deleted**. Additionally, **add a check to ensure the movie can be safely deleted** (no cinema programs associated with it)

A screen shot of a computer program

Description automatically generated

* **Add Soft Delete Actions** in MovieController
  + **GET** action for soft deleting a movie

A computer screen shot of a program code

Description automatically generated

* + **POST** SoftDeleteConfirmed Action - This action **attempts to soft delete the movie**.   
    If **restrictions are violated**, it **redirects back to the view** with an error message

A screen shot of a computer program

Description automatically generated

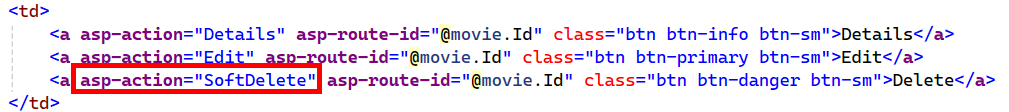
* **Update MovieDetailsViewModel**
  + Add an Id property to it, so the soft delete view can access the cinema's identifier.

A screenshot of a computer program

Description automatically generated

* **Updated Manage Movies View**

To make the **Delete** button in your **Manage Movie** view redirect to the **SoftDelete** confirmation page (instead of directly deleting the movie), we’ll update the button to use the SoftDelete action rather than a direct Delete action. This will guide users to a confirmation view where they can confirm the deletion.



* Create **SoftDelete.cshtml** Confirmation View
  + In **Views/Movie**, **create a SoftDelete.cshtml** view to confirm deletion. This view will display the movies’s details and ask for confirmation

A computer screen shot of a program

Description automatically generated

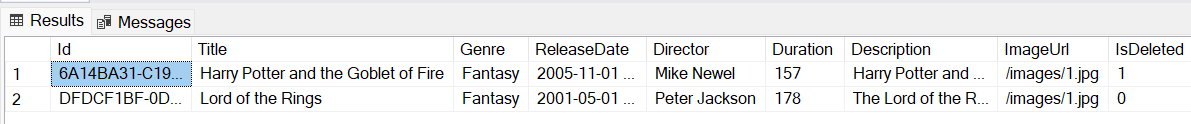
### Code File for the Movie SoftDelete View

### Add and Apply Migration

* **Create a new migration file in your project that adds the IsDeleted column to the Movies table.**

**A screenshot of a computer

Description automatically generated**

****

### Ensure that soft-deleted movies do not appear on the Movies page

* **You need to add a filter to exclude movies where IsDeleted is set to true. This can be done in your GetAllMoviesAsync method in the MovieService**

**A computer screen shot of a computer code

Description automatically generated**

* **You can also apply a global query filter in the entity configuration to exclude soft-deleted entries by default.**

**A computer code with text

Description automatically generated with medium confidence**

## Watchlist Functionality Separation

By **isolating the Watchlist functionality**, each feature (Movies, Cinemas, Watchlist) can evolve independently, making the **codebase easier to extend and maintain**.

### Creating the IWatchService Interface

* Тhe **IWatchlistService** interface will **define the methods required** for managing the Watchlist

**A screenshot of a computer program

Description automatically generated**

### Injecting the WatchlistService into the WatchlistController

After extracting the logic into the WatchlistService, we now need to use this service within our WatchListController. **Dependency injection** (DI) is the method we use to provide the controller with the necessary services it depends on.

A screen shot of a computer

Description automatically generated

### Implementing the WatchlistService

* The WatchlistService is a **dedicated service layer** that handles the logic for managing users' movie watchlists. By creating this service, we **separate the watchlist-related logic from the controller**, achieving a cleaner structure and better maintainability. This separation allows us to apply business logic within the service, ensuring consistency and reducing code duplication.

A screenshot of a computer

Description automatically generated

* GetUserWatchlistByUserIdAsync
  + Responsible for **retrieving all the movies a user has added to their watchlist**. It queries the database to fetch the relevant movies and **returns them as a list** of ApplicationUserWatchlistViewModel objects.

A screenshot of a computer program

Description automatically generated

### Refactoring the Controller

* **Move the query that retrieves the watchlist for the user from the controller to the** WatchlistService. This will make the Index action simpler and offload the data-fetching responsibility to the service.



### Updating DI in Program.cs

### Bootstrap Error Message Alert