# Exercises: ASP.NET Core

# Implementing Functionalities Part III

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

## Bonus: Using Icons with Bootstrap

In this project, we enhance the interface by **adding icons to buttons**, making the user experience clearer and more visually appealing. We use **Bootstrap Icons**, a free icon library that integrates seamlessly with Bootstrap.

### Import Bootstrap Icons

* To use Bootstrap Icons, we need to add the **Bootstrap Icons CSS file**. Open the \_Layout.cshtml file and add the following line inside the <head> section:

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css">

A screen shot of a computer code

Description automatically generated

## Ticket feature in the application

### Creating ViewModels for the Ticket Feature

In order to implement the ticket functionality, you’ll need to create some ViewModels: BuyTicketModel, UserTicketViewModel, **SetAvailableTicketsViewModel**

* The BuyTicketViewModel will be used when a user wants to purchase tickets for a movie in a cinema.
* The UserTicketViewModel is for displaying purchased tickets to the user in a structured format.
* The SetAvailableTicketsViewModel to handle the data required for setting available tickets.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

### Code Files for the View Models

### Create a Service Layer

* Implement ITicketService and TicketService to handle the core logic of **creating**, **retrieving**, and **deleting** tickets

A screenshot of a computer program

Description automatically generated

### Overview of the Service Layer

* **User Operations:**
  + BuyTicketsAsync - A method for users to purchase tickets, selecting the number of tickets they want for a specific movie in a particular cinema
  + GetUserTicketsAsync: A method to retrieve a user’s purchased tickets for display in their ticket section
* **Manager Operations:**
  + SetAvailableTicketsAsync**:** Allows the manager to set or update the number of available tickets for a specific movie in a specific cinema
* **Ticket Availability:**
  + DecreaseAvailableTicketsAsync: A method to decrease the number of available tickets for a movie in a cinema when a user purchases tickets

### Temporary TicketService Implementation



In this initial setup, we’re providing a **basic implementation** of the TicketService to manage tickets. The methods in **this service currently only return bool values**, which serve as indicators of whether the operations were successful. However, this is a **temporary solution**, and we will introduce a more robust approach to track ticket availability.

### Using the CinemaMovie Mapping Tabel to Track Tickets Availability

Rather than creating a separate CinemaMovieTicketAvailability table, we can add an AvailableTickets field to the **existing mapping table**. This will avoid the need to maintain an additional table and reduce data redundancy.

* To **avoid conflicts when updating the database**, you should **delete or temporarily remove any existing entries in the** CinemaMovie **table** that could interfere with this new structure.

A screenshot of a computer program

Description automatically generated

* **Modify the CinemaMovie Class**

**A screen shot of a computer program

Description automatically generated**

* **Update the Database with a Migration**

** **

**A screenshot of a program

Description automatically generated**

* **Update to** AddMovieToCinemasAsync **Method**
  + When creating a new CinemaMovie entity, set the AvailableTickets to zero

A screen shot of a computer program

Description automatically generated

* + When a new CinemaMovie entity is created, we initialize AvailableTickets to **zero**, as required
  + This ensures that a **newly added movie to a cinema program has zero available tickets** until the **manager explicitly sets a value for** AvailableTickets

### Include a Generic Method for Complex Queries

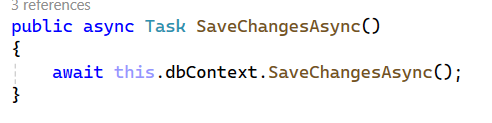
* You can **add a method to IRepository that allows complex queries**, which could be useful for scenarios like filtering by both cinemaId and movieId in CinemaMovie



* The BaseRepository currently **does not have a method for saving changes independently**, as it relies on operations that automatically save changes after actions like Add, AddRange, Delete, and Update. However, **if you need to save changes directly in your service (e.g., after modifying a property without triggering SaveChanges)**, it would be helpful to **add** a SaveChangesAsync method to the BaseRepository

A black and brown text

Description automatically generated



* Then, **implement** this in BaseRepository as follows:

A computer code with text

Description automatically generated with medium confidence

* Then, in TicketService, you can use FindByConditionAsync to **get a specific CinemaMovie**:

### Enable the TicketService Methods to Adjust the Ticket Availability

A screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

### Set Up the TicketController

To proceed with the TicketsController, let’s **outline the necessary actions** based on the functionalities:

* **BuyTicket (GET and POST):** This allows the user to purchase a ticket, redirecting them to a view where they can **select the number of tickets** and **confirm the purchase**
* **MyTickets (GET):** A view where users can see the **tickets they've purchased**
* **SetAvailableTickets (GET and Post):** For managers, this displays a list of cinemas, where they can view and update **available tickets for each movie in the cinema’s program**

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer code with text

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

## Function of the Manager in CinemaApp

The Manager role in CinemaApp has specific responsibilities focused on overseeing the application’s content rather than using its regular user features. The manager can:

* **Manage Cinemas**: Add, edit, and delete cinemas available in the application
* **Manage Movies**: Add, edit, and delete movies in the system
* **Manage Tickets**: (To be implemented later) Oversee ticket sales and manage ticket-related features

Since the **manager's focus is on administration**, they **do not need access to features meant for regular users**, like the Watchlist. Therefore, we will **remove My Watchlist from the navbar.**

A screenshot of a computer

Description automatically generated

### Remove Access to "My Watchlist" for the Manager

* In your project, navigate to the shared \_Layout view file
* Locate the Watchlist link
* Add a condition to **hide** it for **managers** and **not-logged-users**

A screen shot of a computer

Description automatically generated

**With this approach, the navigation bar will adapt based on the user’s authorization, keeping it consistent with the intended functionality of the CinemaApp**

**Navigation for Regular User (Non-Manager):**

**A close up of a logo

Description automatically generated**

**Navigation for Manager:**

**A screen shot of a computer

Description automatically generated**

### Removing the Details Button from Manage Cinemas

**In this step, you’ll edit the Manage Cinemas view so that only the Edit and Delete buttons are displayed. The Details button is not needed for the manager user**

A screen shot of a computer

Description automatically generated

The **Manage Cinemas** view provides the manager with an organized interface to oversee cinema entries. This view displays each cinema’s name and location in a structured table format, along with action buttons for **Edit** and **Delete**. The table includes a convenient "Add New Cinema" button that allows the manager to quickly add new cinemas to the list. With the help of Bootstrap styling and icons, the view is intuitive and user-friendly, ensuring that the manager can efficiently manage cinema information. The buttons also feature icons for easy recognition, following a consistent design pattern across the application.

### Code File for the Manage Cinema View

### Add the Manage Action in MovieController

* **Add a new Manage action** method in MovieController.
* This action will be responsible for **retrieving the list of movies** and passing them to the  
   Manage.cshtml view.
* **Only manager users** should be able to **access** this action.

A screenshot of a computer program

Description automatically generated

### Creating the Manage Movies View

A screenshot of a computer

Description automatically generated

In this section, you will create a dedicated **Manage Movies** view for the manager, similar to the **Manage Cinemas** view. This view will allow the manager to **Add to Program**, **Edit**, and **Delete** movies, following a consistent pattern in the application.

* **Create the Manage View**
  + Navigate to the **Movie Views Folder**
  + Right-click on the **Movie** folder, select **Add** > **New Item**…, and choose **Razor View**
  + Name the new view Manage.cshtml and click **Add**

**A screenshot of a computer

Description automatically generated** **A screenshot of a computer menu

Description automatically generated**

**Manage Movies Link**

Make sure, that in your dropdown menu for the Manager section, it the Manage Movies link is pointing to the Manage action. To direct it to the new Manage action in the MovieController, you should update the asp-action attribute.

**A screenshot of a computer program

Description automatically generated**

**Code File for the Manage Movie View**

## Application User, Logged-In-User-Not-Manager

A regular user in our CinemaApp can:

* **View All Cinemas**: See a list of all cinemas with basic information and a **"Program"** button to view the movies currently being shown and buy tickets
* **View Movie List:** Browse movies and see detailed information about each movie, add movies to a personal watchlist
* **Manage Personal Watchlist**: Users can add movies to a watchlist to keep track of films they want to watch
* **View Purchased Tickets**: Access a section where they can see all the tickets they’ve bought

### Update NavBar

* Remove the "**Home**" button, as the "**CinemaWebApp**" logo serves the same purpose
* Rename "**My Watchlist**" to "**Watchlist**"
* Add a "**Tickets**" navigation item for logged-in users who are not managers
* Ensure that **managers** will see **only** the "**Manager**" dropdown menu

**Navigation for Not Logged-In User:**

A close up of a logo

Description automatically generated

**Navigation for Manager:**

A close up of a logo

Description automatically generated

**Navigation for Regular Logged-In User:**

A close up of a sign

Description automatically generated**Code File for the \_Layout View**

**Cinema Index View (Logged-In-User)**

A screen shot of a computer

Description automatically generated

### Code File for the Cinema Index View



### Cinema Program for Logged-In-Not-Manager-User

To create the view for displaying all movies in a specific cinema's program, we can structure it to include each movie's details, along with options to view more details, buy tickets, or return to the list of cinemas.

* **MovieInCinemaViewModel**

**A screen shot of a computer code

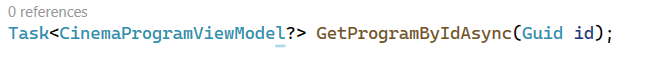
Description automatically generated**

* **CinemaProgramViewModel**

A screenshot of a computer program

Description automatically generated

* To support the ViewProgram functionality, we need to **add a method to retrieve the cinema program**



A screen shot of a computer program

Description automatically generated

* CinemaController - Add ViewProgram Action

A screen shot of a computer program

Description automatically generated

## Cinema ViewProgram for Logged-In-Not-Manager-User

A screenshot of a computer

Description automatically generated

### Code File for the Cinema Program View



## Manage Tickets Functionality

### Adding Manage Tickets Option to the Manager Dropdown

* To add the "Manage Tickets" option to the Manager dropdown, you can modify the dropdown menu within the layout code to include a link to the ManageTickets action in the TicketsController.

A screenshot of a computer code

Description automatically generated

A close up of a sign

Description automatically generated

A screen shot of a ticket

Description automatically generated

## API Approach for Managing Tickets Availability

### Create an API Controller

* **Add a new controller** to handle **ticket management via API**, such as TicketsApiController
* This controller will use [ApiController] and [Route("api/[controller]")] attributes for routing and RESTful API conventions
* We’ll use the [Authorize] attribute to require that **only logged-in users can access this controller**

A screenshot of a computer program

Description automatically generated

### Define API Endpoints

* **Get Movies by Cinema ID**: An endpoint to **retrieve the movies associated with a particular cinema**

A screen shot of a computer program

Description automatically generated

* **Update Available Tickets**: An endpoint to **update the available ticket count** for a specific movie in a cinema

A screenshot of a computer program

Description automatically generated

### Creating the Frontend Interface to Interact with the API

To **allow managers to interact with this API** and update ticket availability, we'll need to:

* Use JavaScript (or a frontend framework/library like jQuery) to make **asynchronous requests to the API**
* Display a modal or popup when the manager clicks the **Manage Tickets** button for a specific cinema, listing **all movies for that cinema with a field to set available tickets**
* **Send the CinemaId, MovieId, and AvailableTickets data** from the frontend **to the UpdateAvailableTickets endpoint in the API**

### Setting Up the Frontend UI for Ticket Management

* **Create a Partial View**
  + Add a new partial view in the **Ticke** folder and name it \_ManageTicketsModal.cshtml

A screenshot of a computer code

Description automatically generated



#### Include the Partial View in the Main View

Now, go back to your main Manage.cshtml view and **render this partial view**. You can add a hidden placeholder that will load the partial modal when required



A screen shot of a computer code

Description automatically generated

#### JavaScript Organization and Functionality Explanation

* **File Structure**
  + We've organized our **JavaScript code for managing tickets into a dedicated JavaScript file** named manageTickets.js. This file is stored in the wwwroot/js/ folder, where we keep all JavaScript files related to the project. By keeping our JavaScript code in a separate file, we improve the structure of our project, making the code easier to maintain and reuse

A screenshot of a computer

Description automatically generated

#### JavaScript Functions in manageTickets.js

* Function: openManageTicketsModal(cinemaId)
  + This function opens the "Manage Tickets" modal and loads the movies for a specific cinema

A screen shot of a computer code

Description automatically generated



* Function: renderMoviesInModal(movies)
  + This function generates and renders the HTML content inside the "Manage Tickets" modal using the movies data

A screenshot of a computer program

Description automatically generated



* Function: updateAvailableTickets(movieId, cinemaId)
  + This function updates the number of available tickets for a specific movie in a cinema

A screen shot of a computer program

Description automatically generated



#### Add the following JavaScript to the Main View

A screen shot of a computer

Description automatically generated

#### Required Models and Updating the ViewModels Project

In this section, we will outline the models needed for the "Manage Tickets" functionality in the API controller and guide you on how to update the ViewModels project to support these requirements.

* CinemaProgramViewModel
  + Represents the overall cinema program for a specific cinema, including its details and a list of movies available in that cinema



* MovieInCinemaViewModel
  + Represents the details of a movie in a cinema, including ticket availability



* SetAvailableTicketsViewModel
  + Used to update the available ticket count for a specific movie in a specific cinema



With these models in place, the API controller can use them to send and receive data.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a ticket

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a ticket

Description automatically generated

A screenshot of a ticket

Description automatically generated