# Exercises: ASP.NET Core - Part V

# Searching, Pagination, Unit Tests

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

## Import More Movies From Json

### Introduction

In this section, you will learn how to **expand the database by importing additional movie records** **from a JSON file**. This step will help **populate your application with more data**, providing a richer user experience and enabling thorough testing of features like filtering, sorting, and displaying movies with images.

The JSON file contains a collection of movies, complete with essential properties like title, genre, release date, duration, description, and image URLs. This data will be added to your existing database to enhance the variety of movies displayed in your application.

By the end of this section, you will:

* Understand how to **work with JSON files** in ASP.NET Core
* **Load and parse data** from a JSON file
* **Insert the parsed data** into the database

### Update the AllMoviesIndexViewModel

* Add a new property for the **ImageUrl** to the AllMoviesIndexViewModel

A screenshot of a computer program

Description automatically generated

### Update the 'Movie Index' View

* Modify the `Movie Index` view, to support image displaying

A screen shot of a computer code

Description automatically generated

* `Movie Index` view code **.zip**



### Clearing Test Movies Data

Before moving forward, it's essential to **clear all the test movies you may have added during development**. This step ensures that your database only contains the movies imported from the JSON file, providing a clean slate for testing and showcasing the updated functionality of your application.

A screenshot of a computer message

Description automatically generated

* **Steps to Clear Test Data**
  + **Option 1: Using Entity Framework**

**A screen shot of a computer program

Description automatically generated**

* + **Option 2: Using SQL Commands**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**The result should be empty movies table in the SSMS:**

**A screenshot of a computer

Description automatically generated**

### Importing Movies from JSON

**Best Practices for Importing Data**:

Before diving into the steps for importing movies, it's important to understand the **best practices** for handling external data in your application.

* **Validate** the Data
* Use a **Separate Service** or **Layer**
* **Error Handling**
  + Implement error handling to **catch and log issues**, such as missing files, malformed JSON, or database errors
  + Provide **meaningful error messages** to aid in debugging
* **Avoid Duplicate** Entries
* Use **Transactions**
  + If inserting **multiple records**, use a transaction to ensure that all records are inserted successfully
  + This **avoids partial updates**

**Place the JSON File:**

* Save your JSON file (e.g., movies.json) in a **dedicated folder** within your project, such as SeedingData

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Adding Movie Import Functionality to the DatabaseSeeder Class:**

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

**Where Else Can Seeding Logic Go?**

* Inside the OnModelCreating Method
* **Custom** Seeder Classes
* During **Application Initialization**

To call your ImportMovies method, you should integrate it into your application startup process in the Program.cs file:

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

### Test the Seeding Logic

Now that you’ve implemented the movie seeding logic in the Program.cs, it's time to test whether everything works correctly. Follow these steps to guide the students through the testing process:

* Verify Database State
* Run the Application
* Verify Seeding in the Database
* Verify in the Application

### Importing Movies: Success!

Congratulations! 🎉 You have successfully completed the process of importing movies into your database using the JSON file. This is a major milestone in your project and a testament to the importance of correctly handling data seeding in applications.

A screen shot of a computer

Description automatically generated

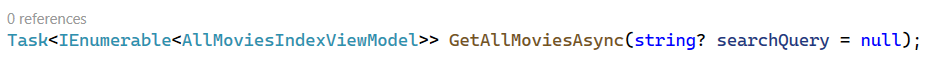
## Implementing Search Functionality in the Movies Page

Now that we have successfully visualized all seeded movies in the Movies page, it's time to enhance the user experience by adding a search functionality.

### Search byTitle

#### Modify the MovieService to Include Search Logic

**Interface Update (IMovieService)**



**Implementation in MovieService**

**A computer screen shot of a code

Description automatically generated**

**Update the MovieController to Handle Search**

* Modify the Index action in your MovieController to **accept a search query as a parameter** and pass it to the MovieService

A computer screen shot of text

Description automatically generated

**Updating the View:**

* In your Index view, **add a search form** at the top to **allow users to input their search query**

A computer code with text

Description automatically generated with medium confidence

* `Movie Index` view code **.zip**



Congratulations! 🎉 We have successfully implemented a search functionality in the Movies page of our CinemaApp. Users can now search for movies by their titles or parts of their titles, making the application more user-friendly and accessible

A screenshot of a computer

Description automatically generated

## Implementing Filtering in the Movies Page

In this section, we will enhance the Movies page by adding **filtering functionality**. Filtering allows users to refine their movie results based on specific criteria, such as **genre** or **release year**.

### Update the Service Layer

* Enhanced GetAllMoviesAsync Method

A computer code with text

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

By reusing and enhancing the existing method, you maintain a single source of truth for your filtering logic, making the service more robust, scalable, and easier to maintain. You don't need a separate method for filtering, as the enhanced version can handle both searching and filtering seamlessly.

### Review and Update the Controller

* Add **Filtering Parameters** to the Index action

A computer code with many colored text

Description automatically generated with medium confidence

### Add a Filtering Form to the View

The form will **allow users to specify the title (search query), genre, and release year for filtering**. Add the form at the top of the view, before displaying the list of movies

A screen shot of a computer program

Description automatically generated

* **Filtering form** code **.zip**



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white rectangular object with text

Description automatically generated

A screenshot of a computer

Description automatically generated

Congratulations! 🎉 In this section, we successfully extended the **Movies** page by implementing a robust filtering feature.

## Implementing Pagination

### Extend the Service Layer

* IMovieService interface should be updated to **include the new method signature for pagination**

A computer screen shot of a computer code

Description automatically generated with medium confidence

* We need to update the GetAllMoviesAsync method to **support pagination**:

A screen shot of a computer code

Description automatically generated

A computer code with text

Description automatically generated

A screen shot of a computer code

Description automatically generated

### Update the Controller Method

* Modify the Index action to **handle the new tuple structure** returned by the GetAllMoviesAsync method. Update the code to **manage both the movies and pagination data**.

**A computer code with text

Description automatically generated with medium confidence**

### Updated View with Pagination

**A computer code with many text

Description automatically generated with medium confidence**

* **`Movie Index`** view updated with pagination, code **.zip**

****

**Congratulations! 🎉 You have successfully implemented pagination in your Movies page. With this feature, users can now navigate through the movie list in an organized way, viewing only a limited number of movies per page. We also ensured that the pagination works seamlessly with the search and filtering functionality, making it a fully dynamic and user-friendly experience.**

**This concludes the implementation of the pagination feature, showcasing the power of combining ASP.NET Core MVC functionalities with user-centric features like filtering, searching, and paginating data. This approach improves usability and lays the groundwork for scaling your application further.**

**A screenshot of a calendar

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

## Unit Testing Services

Unit testing is crucial for **ensuring the reliability and correctness** of your application.

### Choose the Right Testing Framework

* **NUnit** is widely used and offers excellent functionality for test-driven development
* **xUnit** is also preferred for its approach, extensibility, and community support

### Use a Mocking Framework

**Mocking** is essential for **isolating the service being tested**. Use libraries like:

* **Moq**: Lightweight and widely used for creating mock dependencies
* **FakeItEasy**: Another user-friendly mocking library

### Setup Your Unit Testing Project

* Create a **new test project** in your solution to keep tests organized:

A screenshot of a computer

Description automatically generated

A screenshot of a computer test

Description automatically generated

* Add a **References to Your Main Project**

A screenshot of a computer

Description automatically generated

* Install Necessary **NuGet Packages**

A close-up of a message

Description automatically generated

* **Verify** the Project is **Setup** for Testing
  + **Create and Run a Dummy Test**

A close-up of text

Description automatically generatedA screenshot of a computer

Description automatically generated

### Plan Your First Unit Test

In this section, we will create a solid foundation for **testing** the MovieService in our application. **Testing** is a crucial **part of development** that **ensures the methods in your application work as expected** and can handle various scenarios.

Unit testing involves writing **small**, **focused tests for individual pieces of your application**, such as **methods** in a service class.

#### Create a Test Class for MovieService

* Add a new class in the CinemaApp.Tests project

A screenshot of a computer program

Description automatically generated

* **Identify** the **First Test**
  + We will test the GetAllMoviesAsync method from the MovieService. This method retrieves movies from the database, **optionally applying filters** like search query, genre, and release year.
* **Plan Scenarios** for Testing:
  + **Scenario 1**: No filters applied, should return all movies
  + **Scenario 2**: A search query is provided, should return movies matching the query
  + **Scenario 3**: A genre filter is applied, should return only movies in the specified genre
  + **Scenario 4**: A release year filter is applied, should return movies from the given year
  + **Scenario 5**: Filters result in no matches, should return an empty list