# Exercises: ASP.NET Core - Custom Middleware

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

## Restricting Access to Manager Routes

### Goal of This Section

Create a custom middleware that intercepts all HTTP requests and checks:

* If the path starts with /Manager
* If the user is **not in the "Manager" role**, they get redirected to an AccessDenied page

This way, we **secure sensitive pages** (like Manager Dashboard, Manage Movies, etc.) without needing Areas or [Authorize] attributes.

### Create a Middlewares Folder

In CinemaApp.Web project:

* **Right-click** → **Add** → **New Folder** → name it **Middlewares**

A screenshot of a computer

AI-generated content may be incorrect.

### Add the Middleware Class

A computer screen shot of a program code

AI-generated content may be incorrect.

### How It Works – Line by Line

* Constructor
  + It stores the next piece of middleware (or request handler) in the pipeline
  + Every middleware must call \_next(context) at the end to continue request execution
* InvokeAsync Method
  + Gets the **URL path** of the incoming request
  + Converts it to lowercase for consistent comparison (e.g., **/Manager** → **/manager**)
* If the request targets a “/manager” route
  + **Checks** whether the **path** **begins with /manager**
  + This is how we **detect that the user is trying to access a manager-only page**
* Role and Authentication Check
  + First, it checks if the **user is not logged in**
  + Then it checks if the **user is not in the "Manager" role**
  + If either of these **conditions are true → access is denied**
* Redirect to Access Denied
  + The user is redirected to a simple error page that says they are **not allowed to view this content**
  + After redirecting, we **return immediately and stop further processing**
* Allow the Request to Proceed
  + If the **request does not go to a manager route**, **or the user is a valid manager**, the middleware **passes the request to the next component in the pipeline** (such as the controller).

### Register the Middleware in Program.cs

Open your Program.cs file (in the CinemaApp.Web project) and **register the middleware in the correct position** in the **request pipeline**.

Add this line **after app.UseAuthentication();** and **before app.UseAuthorization();**

**A black background with green and white text

AI-generated content may be incorrect.**

## Seeding Roles and a Default Manager User

In this section, **we will add the "Manager" and "User" roles to our application**, and create a default Manager account.

This step is **necessary to make our custom middleware work** — the middleware **checks if a user is in the "Manager" role before allowing access to manager-only pages**.

### Add Role Support to Identity

Open your Program.cs file and locate the Identity configuration:

* Replace this line:

A screen shot of a computer code

AI-generated content may be incorrect.

* With this version that **enables role support**:

A screen shot of a computer program

AI-generated content may be incorrect.

### Creating a RoleSeeder Class

To make role-based access control work in our application, we need to **create and seed roles** in the database. Additionally, we will create a **default Manager account** for testing.

This logic will be placed in a new helper class called RoleSeeder

* Create the RoleSeeder.cs Class

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

### Change Main Method to Async

A black background with white text

AI-generated content may be incorrect.

### Call the Seeder in Program.cs

Scroll to the very bottom of your Program.cs, just **before app.Run();**, and insert:

A screen shot of a computer program

AI-generated content may be incorrect.

### Do You Need a Migration?

**No — you do not need a migration right now if:**

* You are using the built-in IdentityUser class (which you are)
* You haven’t added custom properties to IdentityUser

The default Identity tables (AspNetUsers, AspNetRoles, etc.) were already created during your **initial Identity setup and migration**.  
You can proceed **without running Add-Migration** again.

### Do You Need to Extend IdentityUser?

**No — not yet, unless:**

* You want to store extra fields like FullName, IsDeleted, or ProfileImageUrl

Since your current goal is to:

* Use role-based authorization
* Test middleware
* Seed a test manager

**The default IdentityUser is sufficient.**

## Creating Manager Views and Navigation

Now that we've added **role-based middleware and seeded a default Manager user**, it's time to create actual views that are only **accessible to users in the "Manager" role**. These **views will allow managers to configure cinemas**, **set up movie programs**, and **manage showtimes**.

We will place everything in a regular **ManagerController**, and all views will go under the **Views/Manager/** folder.

### Create the ManagerController

### Create the Manager Views Folder

### Add Manager Navigation to the Layout

## Where to Find the Views

All the views mentioned in this section (ConfigureCinemas, ProgramSetup, and ShowtimeSetup) are **already created and included for you.**

You can find them inside the [.zip file](http://svn.softuni.org/admin/svn/csharp-web/trunk/May-2025/ASP.NET-Advanced/05-Workshop/Views.zip) provided in the **Workshop Lesson** page of the course platform.

**Make sure to:**

1. **Download the archive** from the workshop resources
2. **Extract the contents**
3. Copy the entire Manager views folder into your project’s Views directory
4. Adjust your **ViewModels** and controller actions to match the inputs and display fields from each view

This will save you time and ensure consistency across everyone’s projects.