# Exercises: ASP.NET Core Filters

Problems for exercises and homework for the [“C# MVC Frameworks - ASP.NET” course @ SoftUni](https://softuni.bg/courses/asp-net-mvc).

Let’s add some filters to improve the application handling and lifecycle.

## Logging

Add a global logging filter. **It should work for both MVC actions and pages.**

The filter should use the ASP.NET logger via the console. Look up how to enable the logging to the app console.

Upon entering each action / page handler, the filter should display two consecutive messages:

* info: Executing <HTTP method> <controller name>.<action name>
* info: Model state: [**valid** / **invalid**, depending on the **ModelState.IsValid** value]

Upon exiting each action / page handler, the filter should display one message:

* info: Executed <HTTP method> <controller name>.<action name> in <time elapsed, seconds> s.

For pages, replace <controller name> and <action name> with <page name> and <handler name>.

Time elapsed is measured from the end of the …Executing filter to the beginning of the …Executed filter.

**Hint:** Implement both **IActionFilter** and **IPageFilter**. Reuse the common logic by extracting it in separate methods.

## Global Exception Logging

Add a global **ExceptionFilter** which writes information about the exception (type, stack trace) to the console. It should log with the “Error” level.

## Logging to a File

Write a custom logger that writes the result to a file, or use one of the libraries which do exactly that. I recommend **Serilog**. You can see [the documentation on GitHub](https://github.com/serilog/serilog-aspnetcore).

## Simple Authentication

Add a table for Users. A user has a username and password. Reminder: Don’t store the passwords as plain text.

Don’t write functionality for registering at the moment. “Register” users in the database.

Each registered user can add borrowers, books, authors, movies, and directors. Each non-registered user can access the other functionality (seeing a list of all books and their locations, searching, etc.). This is equivalent to having “library administrators” who can lend books and movies; and the public having access to all information, but being unable to edit anything. We’re going to implement that in the next two problems.

## Users Controller

Create a controller with methods for login and logout. Provide links for login / logout in the top menu, next to the Search button.

You may need to configure the application cookies first. Search how to do that :).

**Upon successful login, save the user in the Session dictionary. Upon logout, remove the entry.**

## Authorization

Create a custom action / page filter (like in Problem 1, better create one class which does both) which takes a look at the **Session** and searches for a logged in user. If there is no logged in user, redirect them to the login page (**/Users/Login**) and show an error message. Use **TempData** for the messages.

Decorate all “add” forms with this attribute.

Later, we’ll see how the ASP.NET Core framework handles all authorization.

## \* Tag Helper: Messages

Create a tag helper which displays messages, in a similar way to Problem 6. Messages have a type (e.g. success, warning, danger, etc.) and a content (either plain text or HTML – as you wish). They are stored in TempData and can be accessed via the Layout. This ensures that if there’s a message present, it will be shown to the user, regardless of the current page.