Project 4 – Movie App

The aim of this project is for you to get more experience working with the MVC project structure and get more comfortable with using Linq.

All the views in this project should use the same LayoutView. You should use a **single file** that states that every View should use the same LayoutView. This was covered in the course slides.

The LayoutView should contain a navigation bar which should have 4 links

- **Home**, this link should navigate to **Home/Index**
- **Movies**, this should navigate to **Movies/Index**
- **Top 5 movies**, this should navigate to a page which should display the 5 highest rated movies
- **Actors**, this page should navigate to **Actors/Index**

The layout View should also contain a footer which says: **© 2018 - MovieApp**

In this assignment you should structure the Models folder as follows:

- A folder for view models
- A folder for entity models
- A folder for input models

Your application should support the following routes:

- **/**
- **/Home**
- **/Home/Index**
    - All the above routes should lead to the same page.
    - That page should have a h1 that says: **Welcome to the Movie App.** It should also show some lines of text(some lorem ipsum text is enough).

- **/Movies/Index**
    - Should show a page that displays a list of all movies
        - Should have a h1 that says: **Movies**
        - It should have a search bar and a button that says: Search!
            - Once clicked, a page showing a list of movies that have a title that **contains** the search term should be displayed.
        - Each item in the list should show the title of the movie, genre and rating.
        - Each movie title should be clickable and when clicked it should direct to **/Movies/Details/id** where id is the id of the movie that was clicked.
        - There should be a button on this page that says: **Add Movie!** Once it is clicked it should direct to **/Movies/Create**

- **/Movies/Details/id**
  - Should show a page that shows the details about the movie with the given id or show an error page if a movie with the given id does not exist in the database
    - The following properties should be shown in this Details view:
      - The title of the movie
      - The genre of the movie
      - The release year
      - The rating
      - The runtime of the movie
      - The image of the movie
      - A list that contains the names of the actors that are starred in the movie, each name should be clickable and should lead to **/Actors/Details/id** where id is the id of the actor that was clicked

- **/Movies/Create**
  - Should show a page that lets the user input a Title of a movie, the genre of the movie, the release year and its runtime in minutes. It should also contain a button that says: **Submit Movie.** Once clicked, it should submit the movie to the server.
  - The id of the new Movie should be the highest id of the movies incremented by one. I.E if the highest id is 12 the new movie should get the id 13. Once the movie has been added to the fake database the action should redirect to Movies/Index where the user should see the new movie at the end of the movie list.
  - The image can be an empty string, the rating can be zero and you dont need to register any actors for the new movie.
  - You should use **data annotations** to validate the input for the new movie, all input fields should be **required** and if they are omitted an error message should be displayed by the corresponding input field.

- **/Actors/Index**
  - Should show a list of the name of all actors
    The name of each actor should be clickable
    - Once it is clicked it should lead to **/Actors/Details/id** where id is the id of the actor that was clicked.

- **/Actors/Details/id**
    - Should show a page that shows the details about the actor with the given id or show an error page if there is no actor with the given id in the database
        - The following properties should be shown in this Details view:
            - The name of the actor
            - The age of the actor
            - A list of movies that the actor has starred in
                - Each movie should be clickable and should lead to **/Movies/Details/id** where id is the id of the movie that was clicked.

In order to implement this you will need to persist some data, since we have not yet discussed databases you will be provided with a fake database in this project. This fake database is just a static class that has a static list called Movies, another one called Actors and a list ActorMovieConnections that connects movies to actors. To use a static class in your controller you **don't need to create** an instance of the class. You can store this file at the root of the project (at the same level as the Startup.cs file)

For example: You can access the Lists of the FakeDatabase in a controller like this:

```
FakeDatabase.Movies;
```

Remember to use the Models, Views, Controllers folder structure.

Do note that you should use 3 controller classes (**HomeController, MoviesController and ActorsController**) in this project.

All data retrieval from the fake database should be implemented using **Linq.**

You should examine the fakeDatabase.cs file to see what properties the entity classes should have.

All views should use the same LayoutView and must therefore only contain the HTML they should display. I.e they should not be full blown HTML pages.

For all links you should use **tag-helpers**

You should also use **tag-helpers** in the form that creates a new Movie

Make sure that you add the changes to Startup.cs (see lecture from 27.Feb)

The project must have the name **MovieApp** because the given fake database expects the project to have that name!

Handin

You can handin in groups of two. The group member that hands in the project should state the full name and RU username of the other group member in a comment.

You are to handin a .zip file containing your entire project