Master's Degree in Agile Software Development for the Web

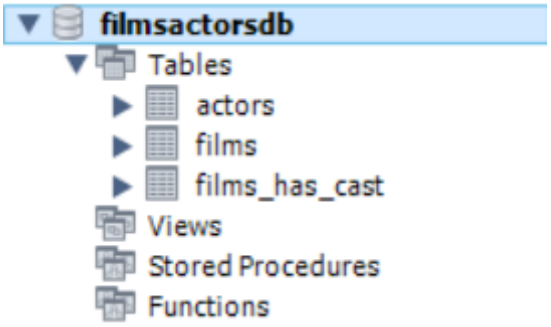
Initial Work Report:
Frameworks Backend y Microservicios

Movie Management Web Application

Tomasz Sojka

1. Task analysis and database design.

First, the task was analyzed and the database was designed. The SQL file was prepared to generate database schema and tables to store films and actors data. There were 3 tables created, one for films details, one for actors and one to join the films with the cast, by storing their ids. Then, the SQL file to initiate some data into tables was created. Both files were run in MySQLWorkbench successfully.



Created database schema.

Afterwards, to test if everything worked well and tables contain initial data, query SELECT * was used.

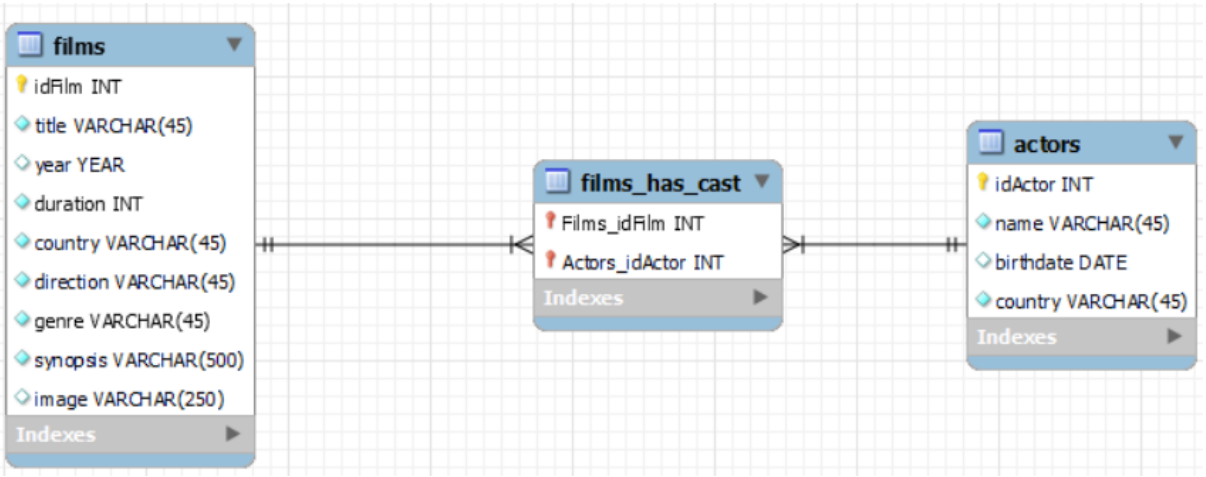
idActor	name	birthdate	country
1	Brad Pitt	1963-12-18	USA
2	Cate Blanchett	1969-05-14	Australia
3	Adriana Barraza	1956-03-05	Mexico
4	Gael García Bernal	1978-11-30	Mexico
5	Morgan Freeman	1937-06-01	USA
6	Kevin Spacey	1959-07-26	USA
7	Tim Robbins	1958-10-16	USA
NULL	NULL	NULL	NULL

Films_idFilm	Actors_idActor
1	1
2	1
1	2
1	3
1	4
2	5
3	5
2	6
3	7
NULL	NULL

idFilm	title	year	duration	country	direction	genre	synopsis	image
1	Babel	2006	144	France / Mexico / USA	Alejandro Gonzalez Inarritu	drama	Tragedy strikes a married couple on vacation in ...	
2	Se7en	1995	127	USA	David Fincher	crime	Two detectives, a rookie and a veteran, hunt a ...	
3	The Shawshank Redemption	1994	142	USA	Frank Darabont	drama	Two imprisoned men bond over a number of ye...	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

All of the initial data selected from the tables. As we can see, everything was inserted properly.

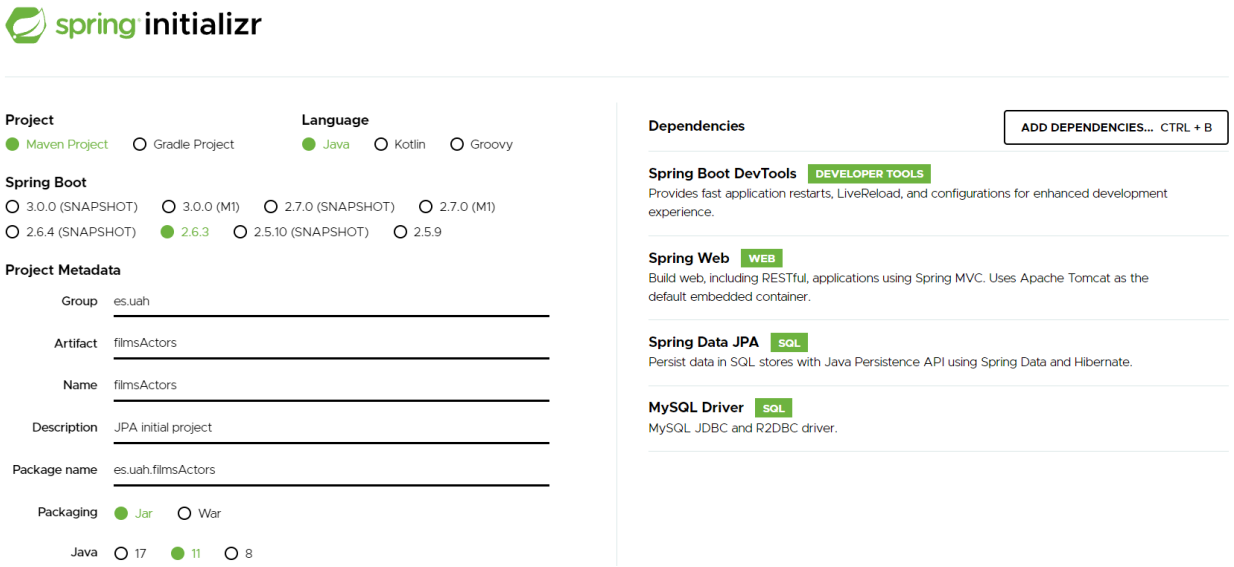
In the end, the EER Model was generated from the created database to visualize the relation between tables.



EER Diagram generated from ‘filmsactorsdb’ database.

2. Backend microservice implementation.

The base of the project was generated using Spring Initializr.



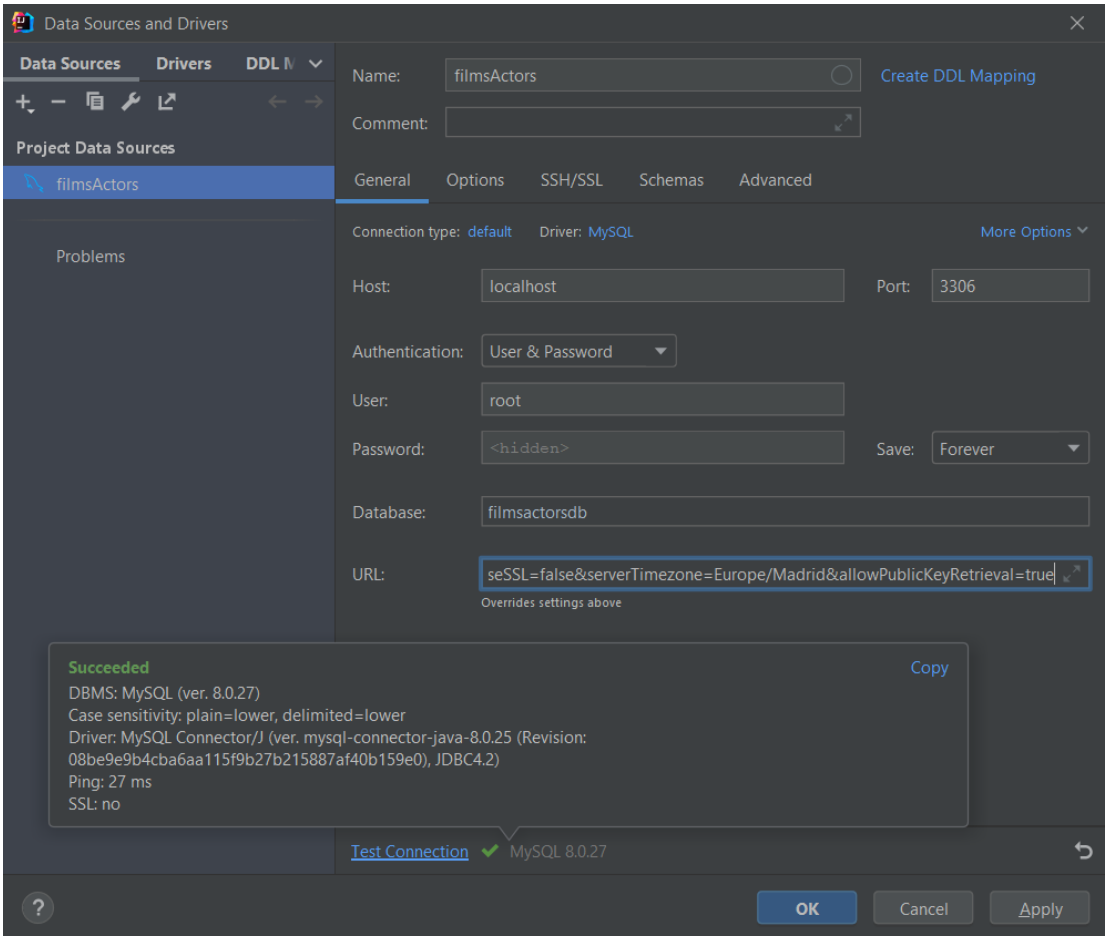
As it can be seen above, java 11 is used and the modules that were included are the following:

- Spring Boot DevTools,
- Spring Web,
- Spring Data JPA,
- MySQL Driver.

Next, the project folder was opened in Intellij Idea IDE. The application.properties file had to be edited, so the access to the database is configured.

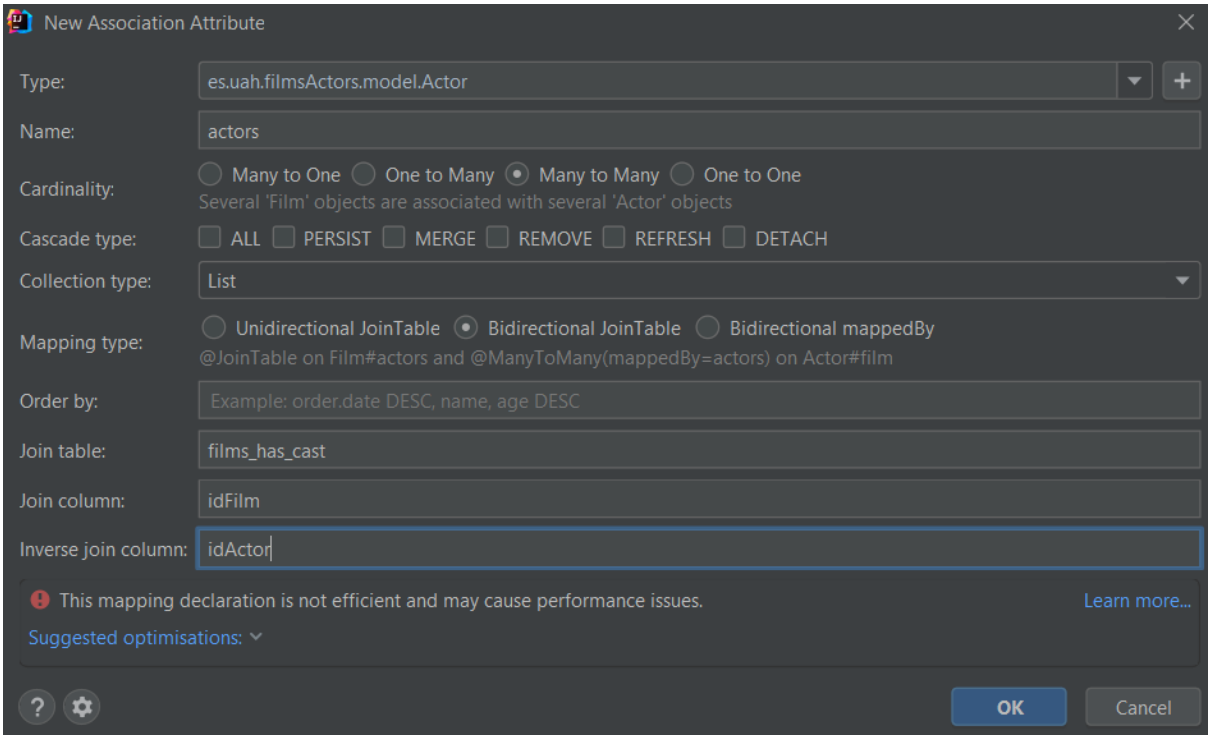
```
application.properties
# DATASOURCE (MYSQL 8.0)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/filmsactorsdb?useSSL=false&serverTimezone=Europe/Madrid&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=rootAdmin1
#JPA
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
# Table names physically
spring.jpa.hibernate.naming.physicalstrategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
# Port and name
server.port=8001
spring.application.name=service-films-actors
```

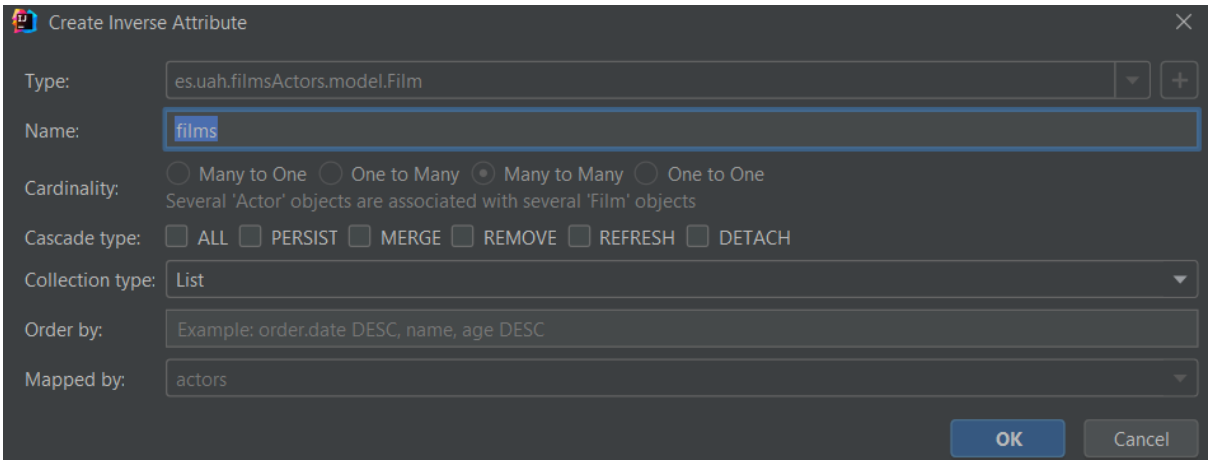
To allow the management of the database directly, a datasource was created and the connection was tested.



The details of the data source creation and the result of the connection tests. Everything went ok.

Then, using the persistence tool window data source was assigned to the entity manager factory and using JPA Buddy plugin entities from the database were created. Afterwards, JPA Palette was used to create a new ManyToMany relationship between Actor and Film entities.





The code of the generated entities was slightly changed.

Afterwards, all of the DAO, Service and Controller classes had to be added and connected. All methods were implemented to enable CRUD operations on films and actors and to enable more specific search of data.

All CRUD operations and searching methods were tested by browser and POSTMAN application.

GET localhost:8001/films/actor/Morg Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results Status: 200 OK Time: 9 ms Size: 580 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "idFilm": 2,
        "title": "Se7en",
        "year": 1995,
        "duration": 127,
        "country": "USA",
        "direction": "David Fincher",
        "genre": "crime",
        "synopsis": "Two detectives, a rookie and a veteran, hunt a serial killer who uses the seven deadly sins as his motives.",
        "image": "",
        "actors": [
            {
                "idActor": 5,
                "name": "Morgan Freeman",
                "birthdate": "1937-06-01",
                "country": "USA"
            },
            {
                "idActor": 6,
                "name": "Kevin Spacey",
                "birthdate": "1959-07-26",
                "country": "USA"
            }
```


Example request sent to find all of the films, whose cast include an actor, whose name includes “Morg” (could be “Morrison”, “Mort” etc.). In this case, all films with Morgan Freeman are listed, because he’s the only actor matching in the database.

Next, the Client microservice had to be implemented.

3. Client microservice implementation.

The base of the project was generated using Spring Initializr.

spring initializr

Project
Maven Project Gradle Project

Language
Java Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M1) 2.7.0 (SNAPSHOT) 2.7.0 (M1)
2.6.4 (SNAPSHOT) 2.6.3 2.5.10 (SNAPSHOT) 2.5.9

Project Metadata

Group	com.example
Artifact	clientFilmsActors
Name	clientFilmsActors
Description	Client initial project
Package name	com.example.clientFilmsActors
Packaging	Jar War
Java	17 11 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

As it can be seen above, the modules that were included are the following:

- Spring Boot DevTools,
- Spring Web,
- Thymeleaf,
- Spring Data JPA.

Next, the project folder was opened in Intellij Idea IDE.

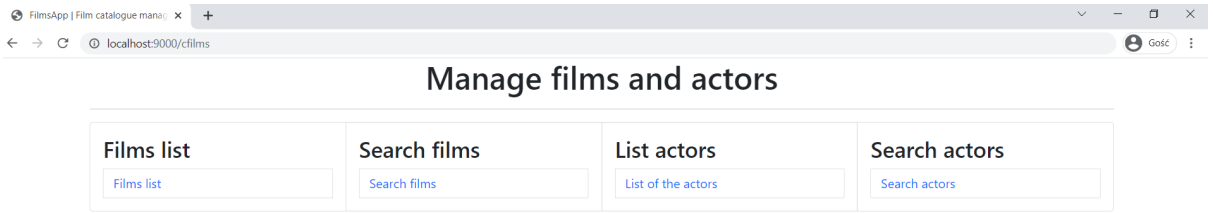
Bean classes that represent model entities from the backend JPA microservice were created. Then, RestTemplate Bean was introduced into the application to enable communication between microservices and application. To divide a list into a series of pages with a certain number of rows paginator classes were implemented. In the end service and controller classes were implemented. Presentation layer was prepared using Thymeleaf and by creating html files. To make the style of the HTML pages more professional, ready-to-use compiled bootstrap code was downloaded and used in HTML files.

4. Manual

To use the application, first start the backend JPA microservice, then start the client microservice. When both are started go to:

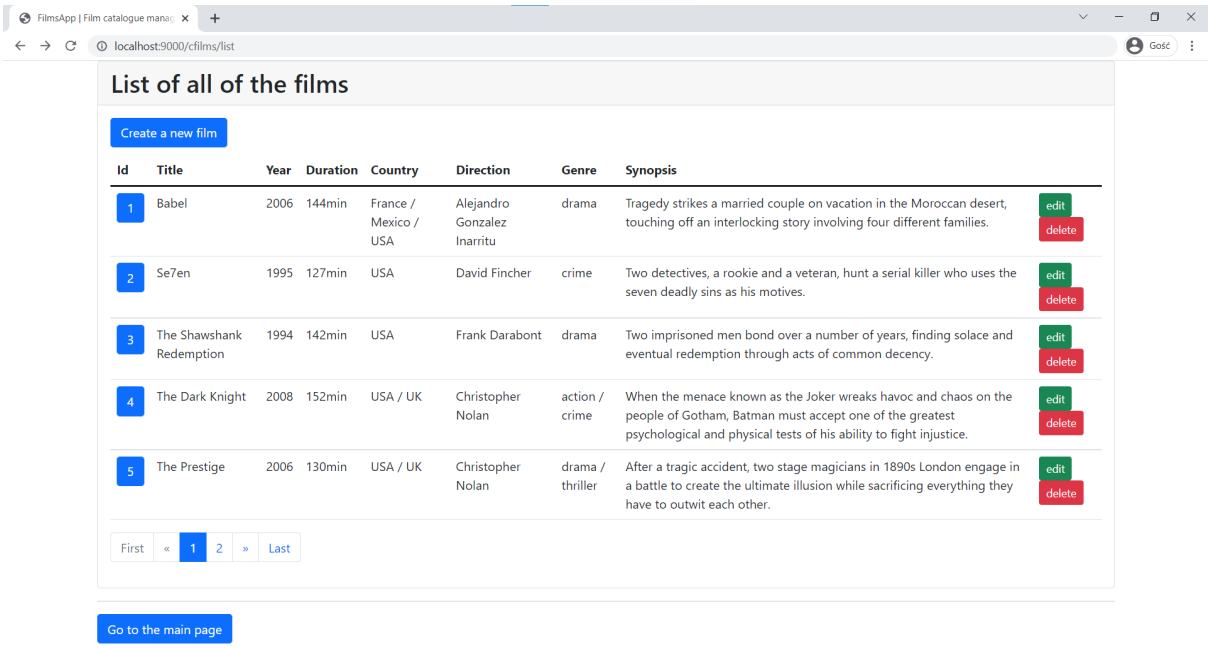
http://localhost:9000/cfilms

where you will find a home page.



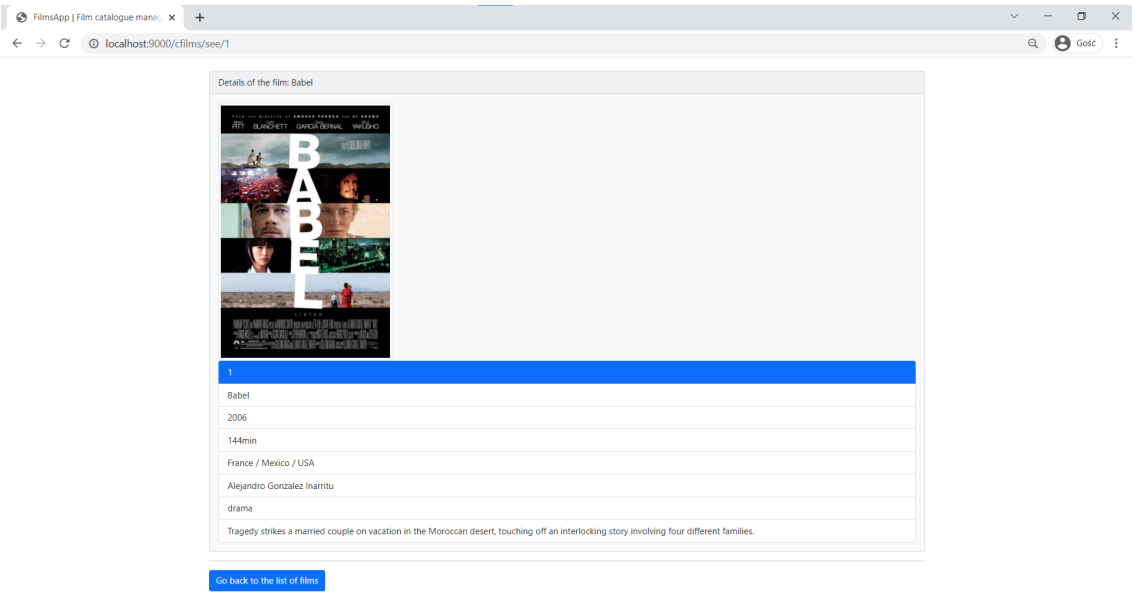
Homepage with the functionalities for administrator user. Administrator can open the page where all of the films are listed, search films by properties or do the similar operations on actors.

If the first option is chosen, the film list page is opened:



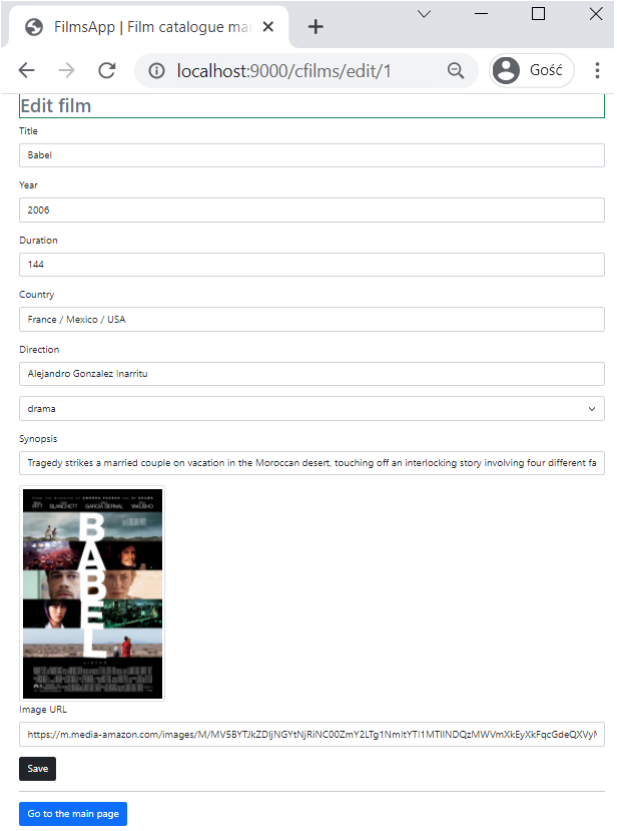
On top of the page there is a button to create a new film. Below, the films are shown sorted by id. Thanks to the pagination classes only 5 films are shown on one page. The pages can be changed by the buttons below the table with films. Each film can be edited or deleted. Each film can be also shown with details by pressing the id button. Down below there is a button to go back to the main page.

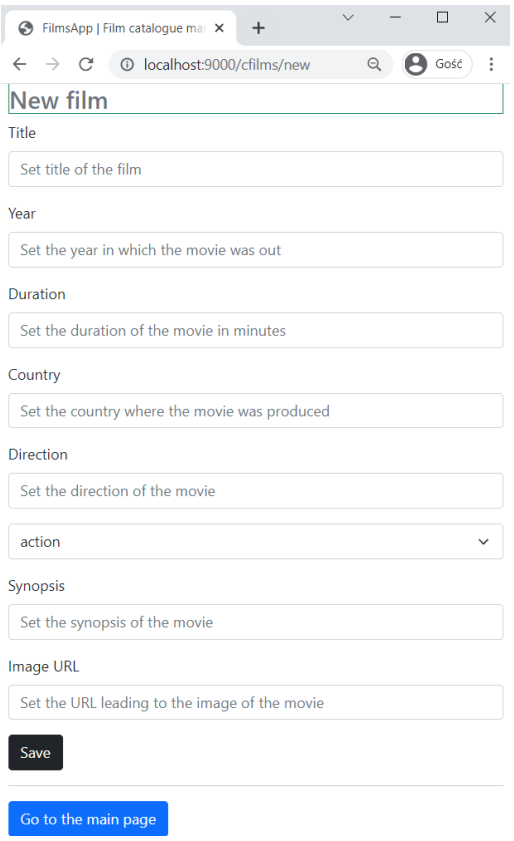
If the id button of the movie is pressed the page with details of the film is opened.



The details of the film of id = 1. The page was a bit zoomed out to see all the presented details. On the bottom there is a button to go back to the list of films.

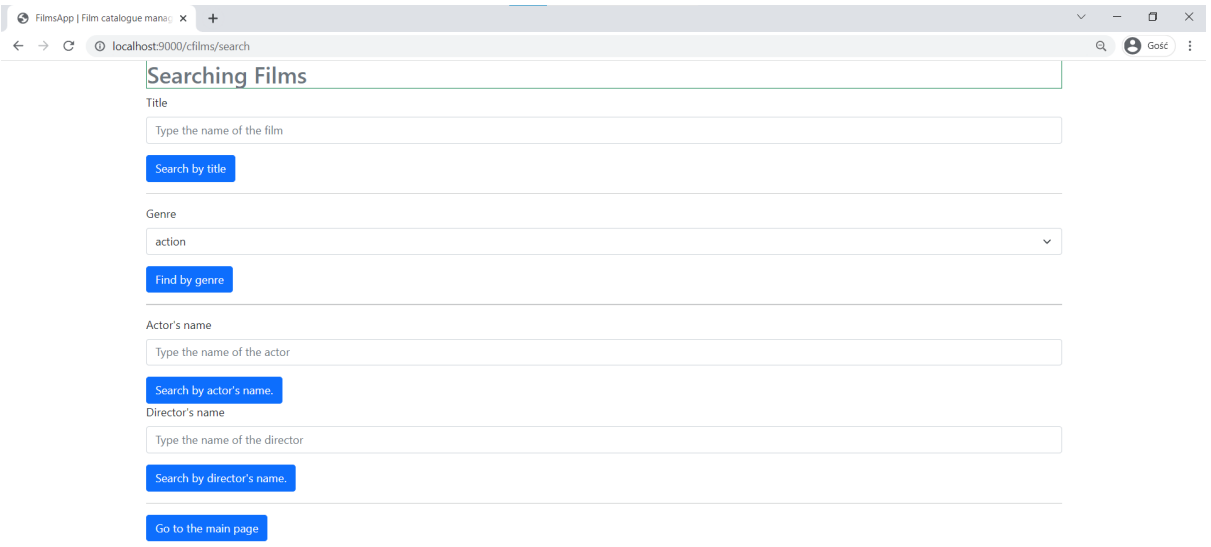
If you go back and press the “edit ”or “create new film” the form for saving the data of the film is opened.





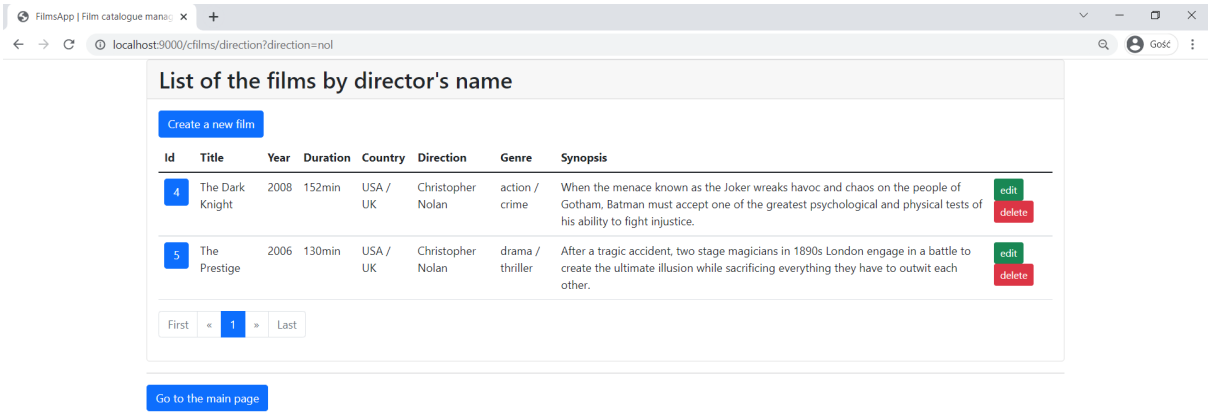
The forms to edit the existing film or to create a new film. The only difference is that for editing, the already saved film data is shown. After pressing the “save” button the data from the form is saved, only if all of the data fields were filled with proper values.

Next, going back to the main page and choosing the second option, the film search page is opened.



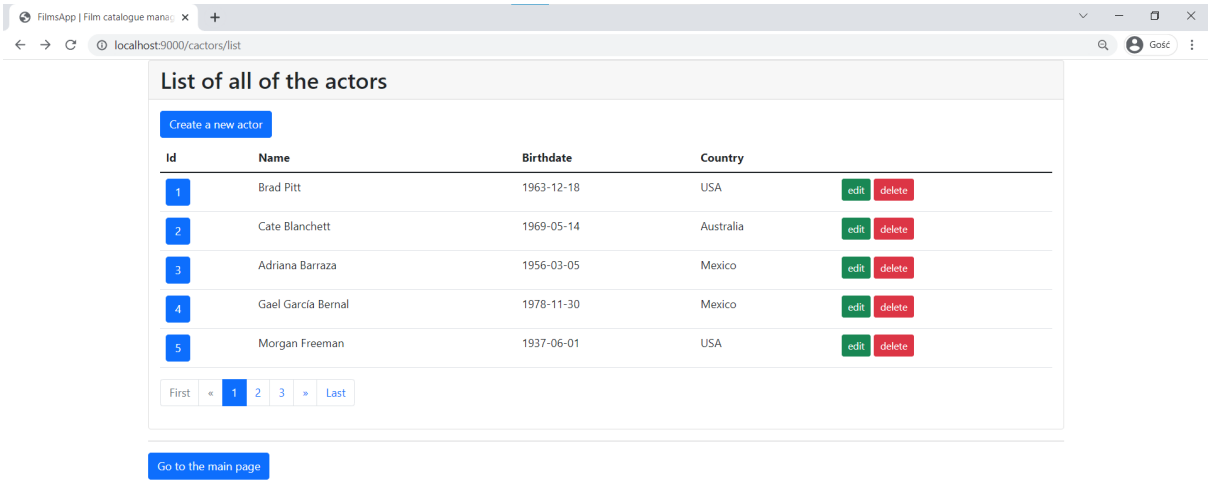
The films can be searched by title, genre, cast member’s name and director’s name. For the genre search, user needs to choose a genre from the list. For the rest of the searching fields user needs to pass a word. The word does not need to be exactly matching the searched result (e.g. to find a movie with morgan freeman it is enough to type “mor” in search by actor’s name).

After submitting the search, the list of films is shown:

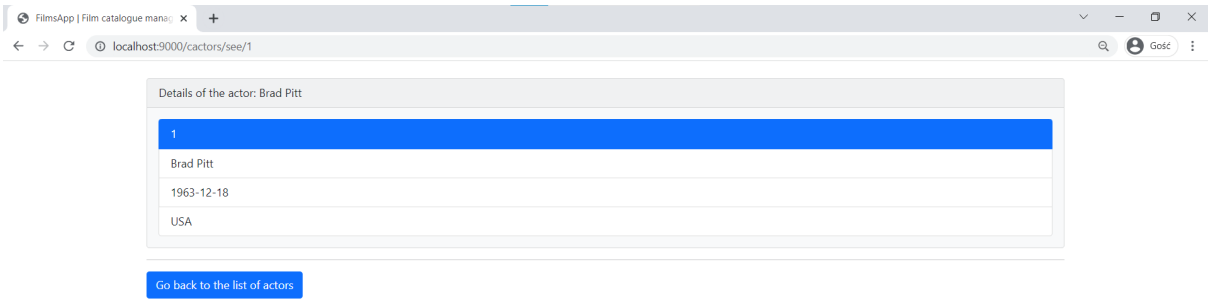


The resulting list of the films directed by a director, whose name includes “nol”.

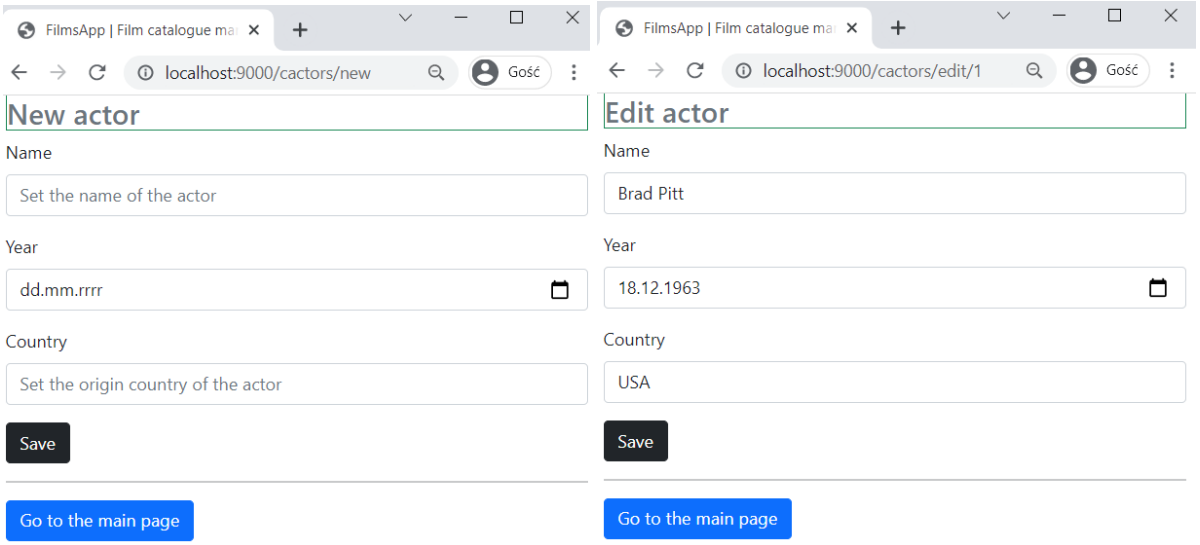
Going back to the main page, there are 2 remaining options, both for actors. The functionalities are similar to ones of the films.



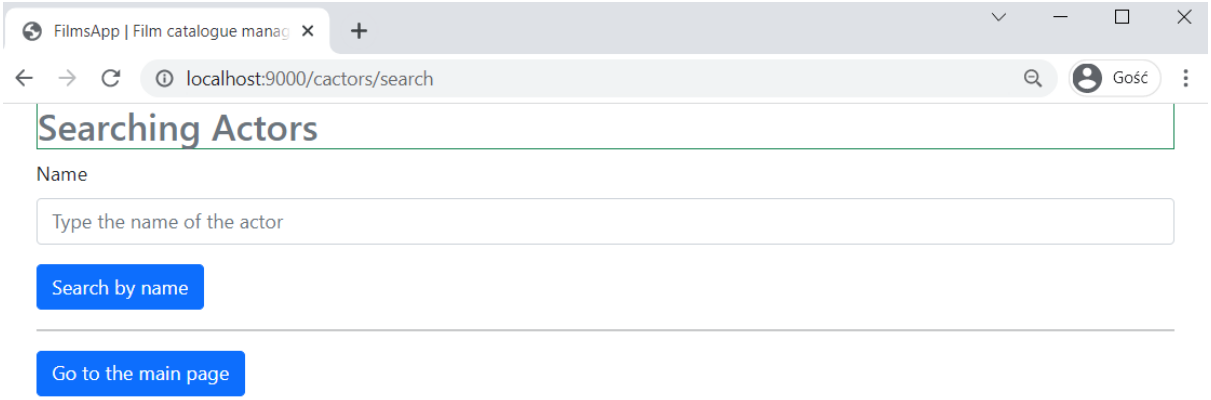
The list of all of the actors.



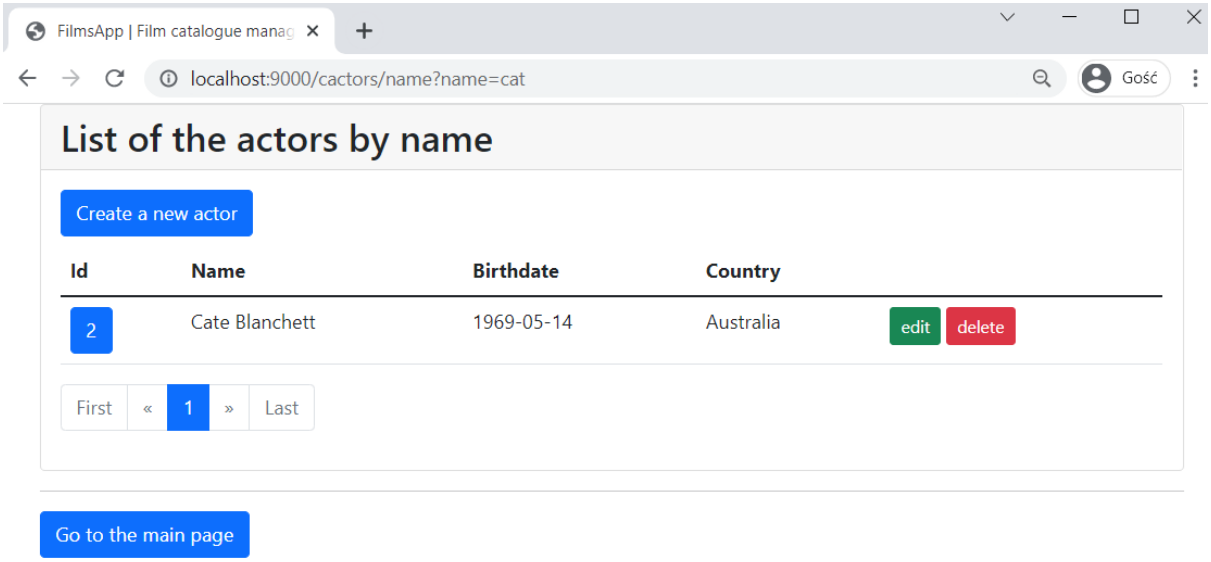
The details of the actor with id = 1.



The forms to add a new actor and to edit an existing actor (in this case the actor with id = 1).



The page to search for an actor by his name.



The result of the search for the actor, after passing “cat” into the search field.