КУРСОВА РАБОТА

БАЗИ ОТ ДАННИ

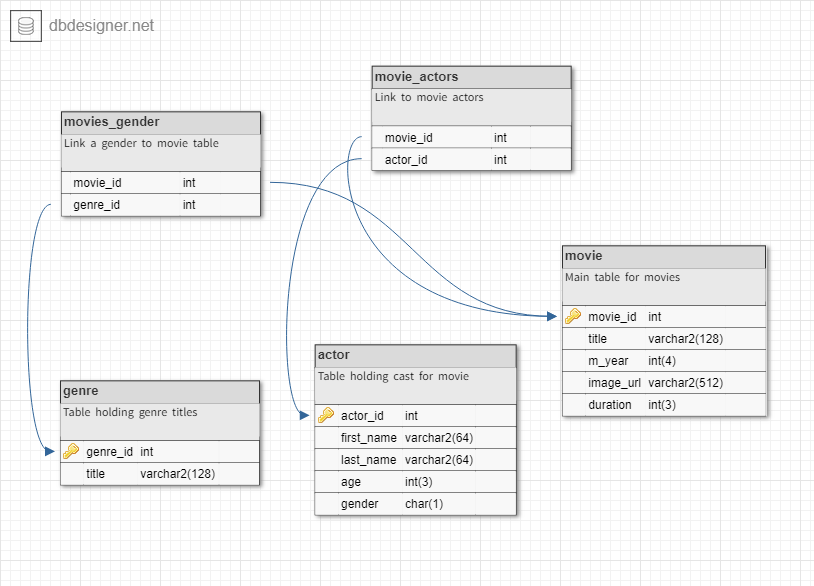
1. Въведение

За изпълнение на курсовата работа е направено уеб приложение от структура на база данни в Oracle. За целта са използвани следните технологии и инструменти:

* Oracle Database (за БД сървър)
* VirtualBox (виртуализация на сървъра за БД)
* Spring framework (за MVC модел)
* Thymeleaf (template engine)
* Bootstrap (визуални елементи)

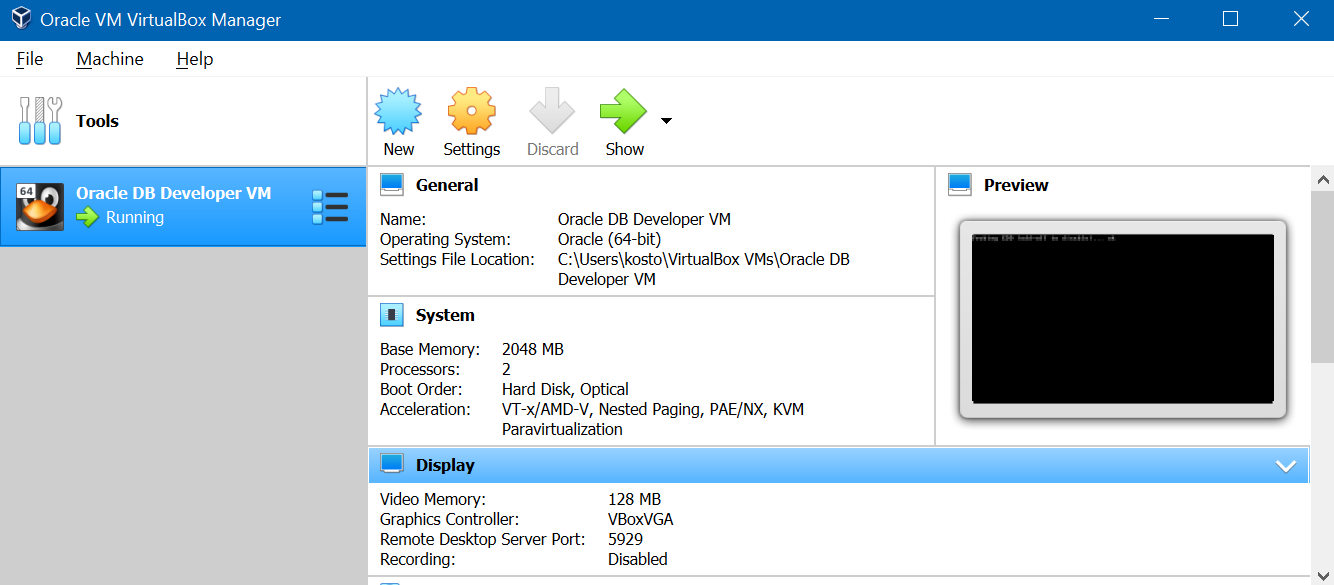
Приложението представлява базисни CRUD операции (**C**reate, **R**ead, **U**pdate, **D**elete) за управление на база данни от филми. Могат да се добавят филми от различни жанрове, заедно с актьорите в него.

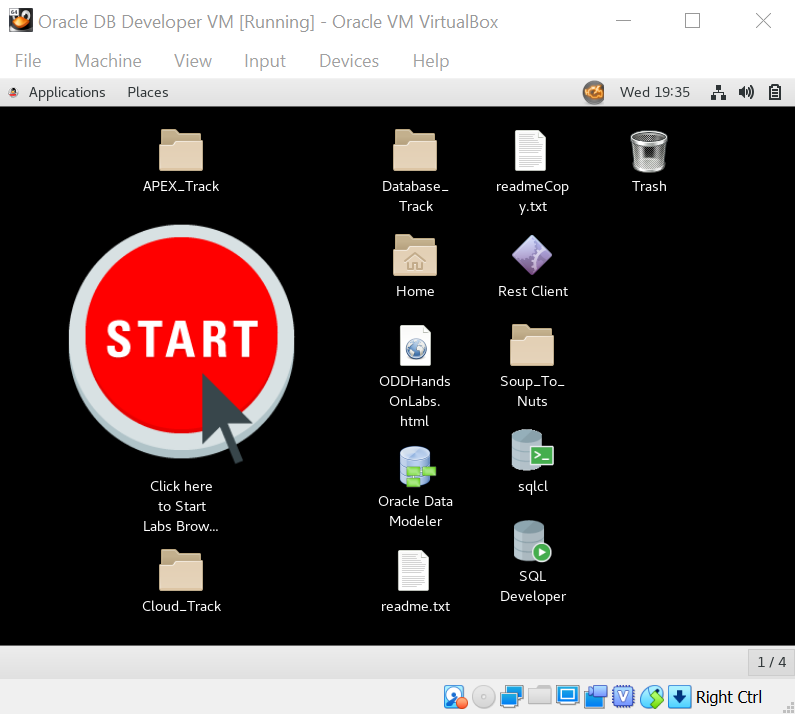
1. База данни - Oracle DB



За изпълнението на задачата е избрана Oracle База данни. Базата данни се казва **movies**. Тя съдържа 5 таблици: **ACTOR**, **GENRE**, **MOVIE**, **MOVIE\_GENRES**, **MOVIE\_ACTORS**. Като последните две са от тип „много към много“ (many-to-many). Тъй като се използва [Spring Framework](https://spring.io/projects/spring-framework), автоматично се добавят и допълнителни атрибути към таблиците.

За инсталиране на БД се използва [готово изображение на виртуална машина](https://www.oracle.com/technetwork/database/enterprise-edition/databaseappdev-vm-161299.html), която върви във [VirtualBox](https://www.virtualbox.org/wiki/Downloads).





След стартиране използваме инструмента на Oracle – [SQLdeveloper](https://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html) за да създадем таблиците, но първо БД:

|  |  |
| --- | --- |
| 5 | ALTER SESSION SET "\_ORACLE\_SCRIPT"=true; |
| 6 | CREATE USER movies IDENTIFIED BY "987321456"; |
| 7 | GRANT CONNECT TO movies; |
| 8 | GRANT CONNECT, RESOURCE, DBA TO movies; |
| 9 | GRANT UNLIMITED TABLESPACE TO movies; |
| 10 | GRANT SELECT, INSERT, UPDATE, DELETE ON schema.movies TO movies; |

Изпълняваме SQL командите за всички таблици:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 12 | -------------------------------------------------------- | | 13 | --  DDL for Table ACTOR | | 14 | -------------------------------------------------------- | | 15 |  | | 16 | CREATE TABLE "MOVIES"."ACTOR" | | 17 | (  "ACTOR\_ID" *NUMBER*(19,0), | | 18 | "AGE" *NUMBER*(10,0), | | 19 | "FIRST\_NAME" *VARCHAR2*(255 *CHAR*), | | 20 | "GENDER" *VARCHAR2*(255 *CHAR*), | | 21 | "LAST\_NAME" *VARCHAR2*(255 *CHAR*) | | 22 | ); | | 23 |  | | 24 | -------------------------------------------------------- | | 25 | --  DDL for Table GENRE | | 26 | -------------------------------------------------------- | | 27 |  | | 28 | CREATE TABLE "MOVIES"."GENRE" | | 29 | (  "GENRE\_ID" *NUMBER*(19,0), | | 30 | "NAME" *VARCHAR2*(255 *CHAR*) | | 31 | ); | | 32 | -------------------------------------------------------- | | 33 | --  DDL for Table MOVIE | | 34 | -------------------------------------------------------- | | 35 |  | | 36 | CREATE TABLE "MOVIES"."MOVIE" | | 37 | (  "MOVIE\_ID" *NUMBER*(19,0), | | 38 | "DURATION" *NUMBER*(10,0), | | 39 | "IMG\_URL" *VARCHAR2*(255 *CHAR*), | | 40 | "TITLE" *VARCHAR2*(255 *CHAR*), | | 41 | "YEAR" *NUMBER*(10,0) | | 42 | ); | | 43 | -------------------------------------------------------- | | 44 | --  DDL for Table MOVIE\_ACTORS | | 45 | -------------------------------------------------------- | | 46 |  | | 47 | CREATE TABLE "MOVIES"."MOVIE\_ACTORS" | | 48 | (  "MOVIE\_ID" *NUMBER*(19,0), | | 49 | "ACTOR\_ID" *NUMBER*(19,0) | | 50 | ); | | 51 | -------------------------------------------------------- | | 52 | --  DDL for Table MOVIE\_GENRES | | 53 | -------------------------------------------------------- | | 54 |  | | 55 | CREATE TABLE "MOVIES"."MOVIE\_GENRES" | | 56 | (  "MOVIE\_ID" *NUMBER*(19,0), | | 57 | "GENRE\_ID" *NUMBER*(19,0) | | 58 | ); | |

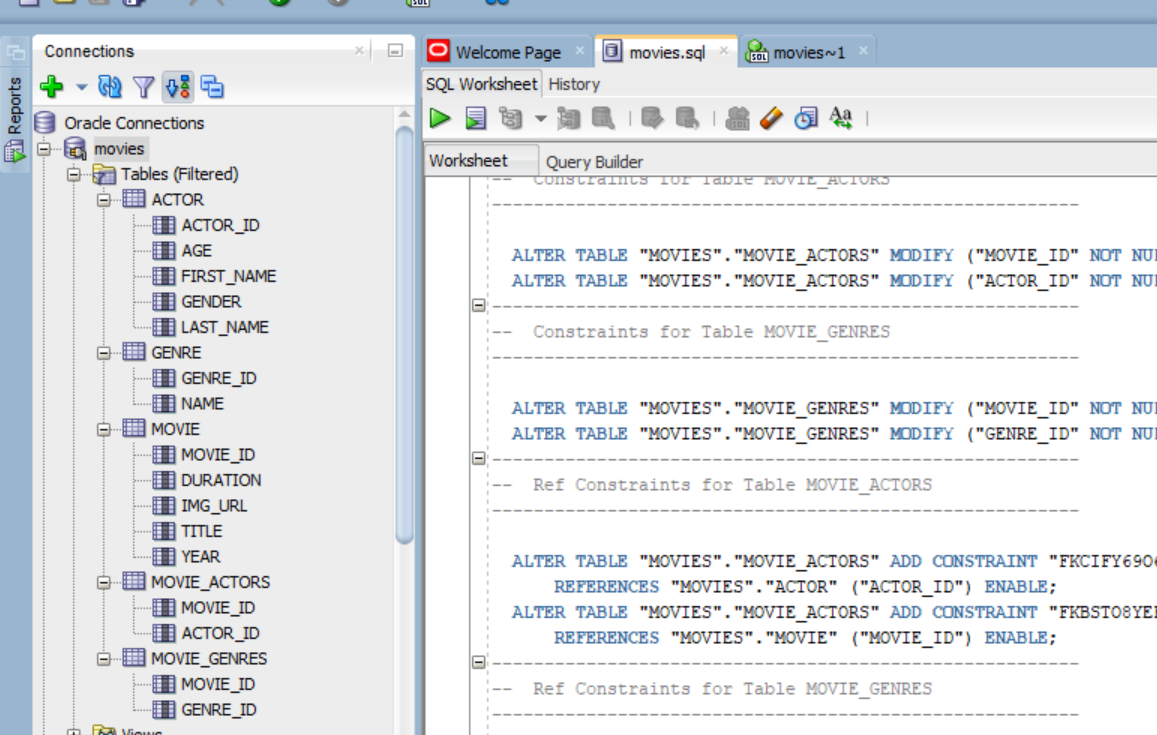
Създават се всички индекси:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 60 | -------------------------------------------------------- | | 61 | --  DDL for Index SYS\_C008785 | | 62 | -------------------------------------------------------- | | 63 |  | | 64 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008785" ON "MOVIES"."ACTOR" ("ACTOR\_ID"); | | 65 | -------------------------------------------------------- | | 66 | --  DDL for Index SYS\_C008787 | | 67 | -------------------------------------------------------- | | 68 |  | | 69 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008787" ON "MOVIES"."GENRE" ("GENRE\_ID"); | | 70 | -------------------------------------------------------- | | 71 | --  DDL for Index UK\_CTFFRBU4484FT8DLSA5VMQDKA | | 72 | -------------------------------------------------------- | | 73 |  | | 74 | CREATE UNIQUE INDEX "MOVIES"."UK\_CTFFRBU4484FT8DLSA5VMQDKA" ON "MOVIES"."GENRE" ("NAME"); | | 75 | -------------------------------------------------------- | | 76 | --  DDL for Index SYS\_C008789 | | 77 | -------------------------------------------------------- | | 78 |  | | 79 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008789" ON "MOVIES"."MOVIE" ("MOVIE\_ID"); | |

И всички релации на таблиците:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 60 | -------------------------------------------------------- | | 61 | --  DDL for Index SYS\_C008785 | | 62 | -------------------------------------------------------- | | 63 |  | | 64 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008785" ON "MOVIES"."ACTOR" ("ACTOR\_ID"); | | 65 | -------------------------------------------------------- | | 66 | --  DDL for Index SYS\_C008787 | | 67 | -------------------------------------------------------- | | 68 |  | | 69 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008787" ON "MOVIES"."GENRE" ("GENRE\_ID"); | | 70 | -------------------------------------------------------- | | 71 | --  DDL for Index UK\_CTFFRBU4484FT8DLSA5VMQDKA | | 72 | -------------------------------------------------------- | | 73 |  | | 74 | CREATE UNIQUE INDEX "MOVIES"."UK\_CTFFRBU4484FT8DLSA5VMQDKA" ON "MOVIES"."GENRE" ("NAME"); | | 75 | -------------------------------------------------------- | | 76 | --  DDL for Index SYS\_C008789 | | 77 | -------------------------------------------------------- | | 78 |  | | 79 | CREATE UNIQUE INDEX "MOVIES"."SYS\_C008789" ON "MOVIES"."MOVIE" ("MOVIE\_ID"); | |

Крайният резултат в SQLDeveloper:



1. Back-end

За разработване на back-end частта на приложението е използвано [IntelliJ IDEA](https://www.jetbrains.com/idea/download/#section=windows). Той разполага с инструмент за генериране на проекти със Spring Framework и чрез него [генерираме структурата на приложението](https://www.javadevjournal.com/spring-boot/spring-boot-application-intellij/).

[Spring](https://spring.io/) е съвременен [framework](https://en.wikipedia.org/wiki/Software_framework) който представлява контейнер за [инверсия на контрола](https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html) за платформата Java, алтернатива на [Java EE (**E**nterprise **E**dition)](https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html)

При генерирането маркираме за добавяне JPA. **[J](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F)**[ava](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F) **[P](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F)**[ersistence](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F) **[A](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F)**[PI](https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F) ще се грижи за връзката между БД и [Entity](https://www.objectdb.com/java/jpa/start/entity) класовете. **Entity** клас е [POJO](https://en.wikipedia.org/wiki/Plain_old_Java_object) (**P**lain **O**ld **J**ava **O**bject) т.е. обикновен Java клас, който е маркиран ([анотиран](https://en.wikipedia.org/wiki/Java_annotation) с @Entity) като имащ за възможност да представя обекти в базата данни. Пример:

**@Entity**  
**public class** Genre {  
  
 @Id  
 **private** Long **genreId**;  
   
 @Column(name = **"name"**, unique = **true**)  
 **private** String **name**;

//… constructors/getters/setters

}

1. Front-end

Избираме за добавяне и **Thymeleaf**. [Thymeleaf](https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html) е модерен [template енджин](https://hackernoon.com/java-template-engines-ef84cb1025a4) за сървъри за уеб приложения, базирани на Java. Представлява алтернатива на JSF ([**J**ava **S**erver **F**aces](https://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html)). Това е стандартен **HTML** синтаксис с допълнителни **th:** тагове. Пример за неговия синтаксис:

<div class="starter-template">

<h1>Spring Boot Web Thymeleaf Example</h1>

<h2>

<span **th:text**="'Message: ' + ${message}"></span>

</h2>

</div>

**th:text** дава връзката между back-end и front-end, а ${message} принтира на екрана съдържанието на променливата message. Декларира се чрез поставянето на **xml** **n**ame**s**pace (xmlns) в елемента html.

<html xmlns:th="http://www.thymeleaf.org">

За визуализиране на компоненти в thymeleaf използваме [Bootstrap](https://getbootstrap.com/). Той е най-известният framework за използване на готови компоненти. В този проект ще се използват предимно [forms и input](https://getbootstrap.com/docs/4.0/components/forms/) полета, [таблици](https://getbootstrap.com/docs/4.0/content/tables/#examples) и [grid система](https://getbootstrap.com/docs/4.2/layout/grid/). Компонентите му се използват като се добавят допълнително класове в таговете на елементите. Нужно е да се добави:

<head>

<!-- other head elements -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

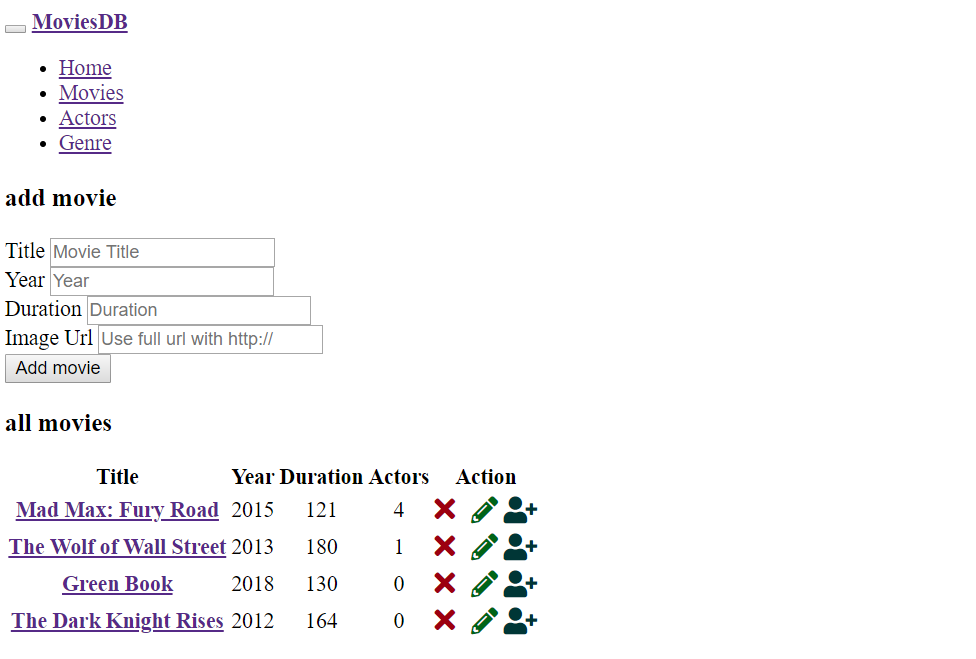
<!-- other head elements -->

</head>

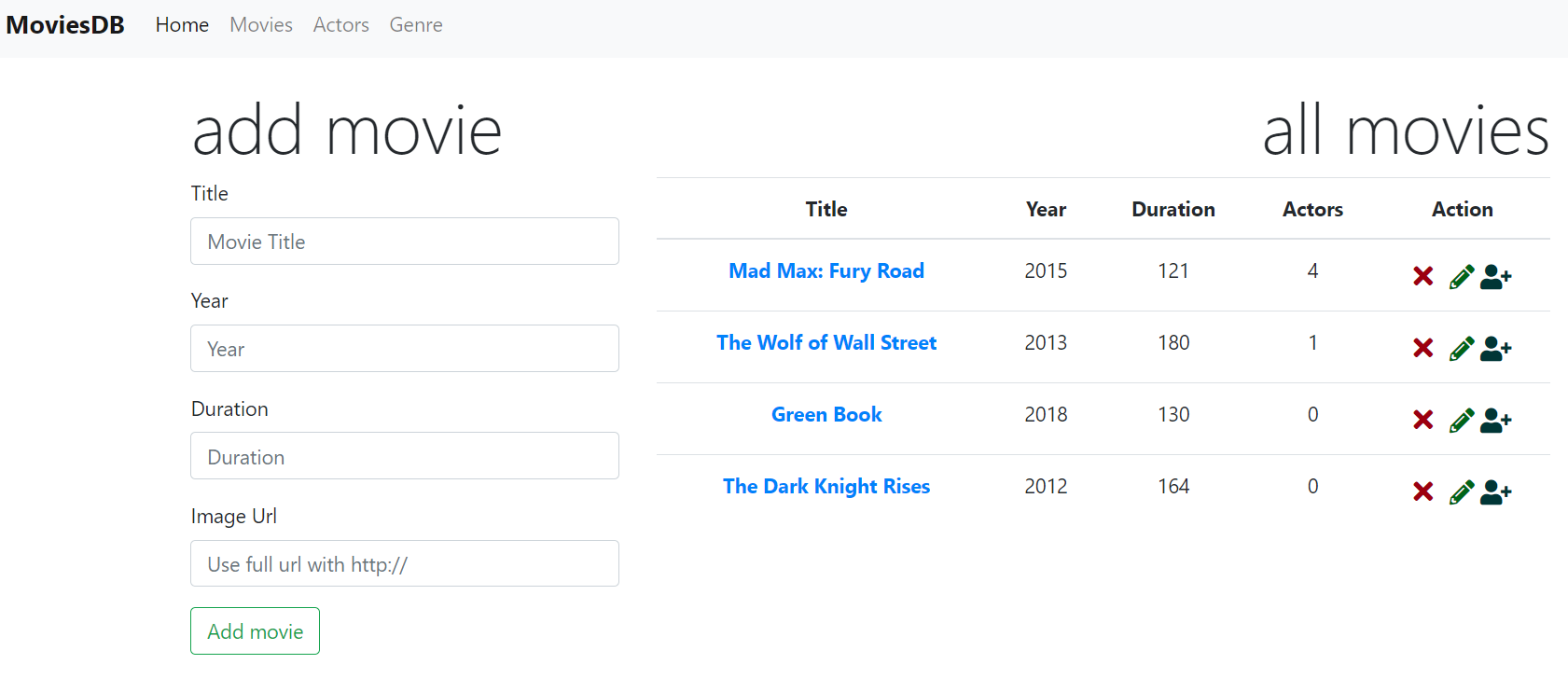
След което се добавят класовете към таговете. Пример с **input** и **form-control**:

<input type="email" **class="form-control"** placeholder="Enter email">

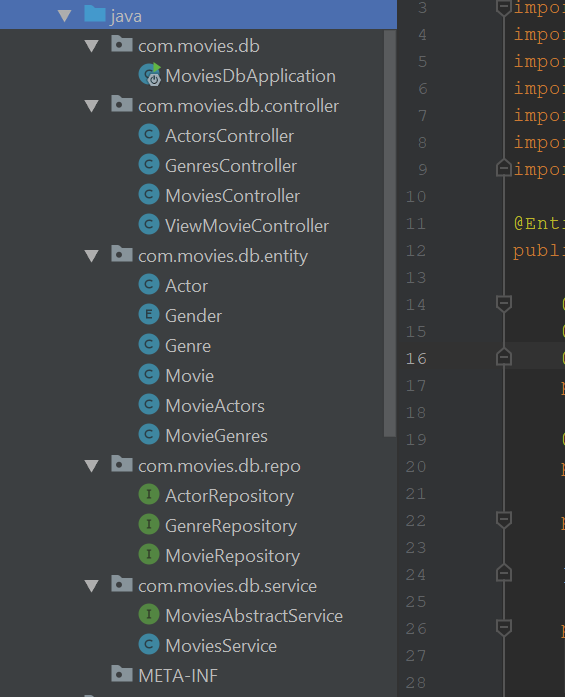
С една и съща структура на страницата **без bootstrap** би изглеждало така:



След добавяне и подреждане на елементите и с компоненти от grid системата



За допълнение са добавени и икони от [Font Awesome](https://origin.fontawesome.com/icons)

Структурата на приложението по пакети е:

**com.movies.db:**

MoviesDbApplication клас в който се намира main метода и е анотиран с

@SpringBootApplication

указващ, че приложението ще стартира от този клас

**com.movies.db.entity:**

Този пакет съдържа всички класове (Entity-та) с техните getters & setters

**com.movies.db.repo:**

Съдържа [интерфейси](http://tutorials.jenkov.com/java/interfaces.html), които наследяват (extends) интерфейса CrudRepository<T, ID> . Той разширява интерфейса на Spring Data Repository. CrudRepository осигурява обща CRUD операции в БД за определен тип **<T>**. Той има общи методи за работа с CRUD операции като:

<S extends T> S save(S entity): Запазва или актуализира обект (entity) в БД  
Optional<T> findById(ID primaryKey):Намира обект по ID от БД  
Iterable<T> findAll(): Връща всички обекти/записите от таблицата  
long count(): Връща броя на записите в БД  
void delete(T entity): Изтрива обекта от таблицата.   
boolean existsById(ID primaryKey): Проверява дали обекта съществува в БД

Практически тези методи са достатъчни, но интерфейса позволява да се добавят още. Пример за Actor Repository използваме [native query](https://www.baeldung.com/spring-data-jpa-query) (обикновенна заявка):

**public interface** ActorRepository **extends** CrudRepository<Actor, Long> {  
  
 @Query(value = **"SELECT movie\_id FROM movie\_actors WHERE actor\_id = ?1"**, nativeQuery = **true**)  
 List<Movies> findAllMoviesByActor(Long actorId);  
}

Този метод се декларира без body ( без { .. } ), защото имплементацията става по време на изпълнение (runtime), автоматично от [Spring Data](https://spring.io/projects/spring-data)

Заявката в анотацията намира всички филми свързани с избрания актьор (actorId). Останалите Repository интерфейси са аналогични.

**com.movies.db.controller:**

В този пакет се намират всички класове които са контролери на html страниците. Затова се анотират с @Controller анотация. В тях се добавят с анотация @Autowired всички repository интерфейси:

…

@Autowired  
ActorRepository **actorRepo**;

…

Така разполагаме с връзката към БД. Всички методи в този клас са свързани с [HTTP заявка](https://www.tutorialspoint.com/http/http_methods.htm) предимно [GET и POST](https://www.w3schools.com/tags/ref_httpmethods.asp). При заявка от клиента на страницата **movies.html** се изпълнява следният метод:

@RequestMapping(**value**=**"/movies"**, **method**=RequestMethod.***GET***)  
 **public** String greeting(Model model) {  
 model.addAttribute(**"movies"**, fetchAllMovies());  
 model.addAttribute(**"movie"**, freshMovie());  
 **return "movies"**;  
 }

Анотацията RequestMapping указва на JVM [(**J**ava **V**irtual **M**achine)](https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html) при търсене на страницата описана в **value** при HTTP заявка от тип описана в **method** да изпълнява метода **greeting()** В аргумента си метода приема **Model model** обект. Този обект ни позволява [да добавяме атрибути под формата на обекти](https://coderanch.com/t/581902/frameworks/Model-classes) към рендирането на html страницата, за да ги използваме за визуализиране на информация. Пример:

model.addAttribute(**"movies"**, fetchAllMovies());

Добавяме всички филми като **List<Movies>** в променлива под името **movies**. След което можем да визуализираме всички филми във front-end частта:

<**tbody**>  
 <**tr th:each="movie:${movies}"**>  
 <**td**><**span th:text="${movie.title}"**> Title </**span**></**b**></**td**>  
 <**td**><**span th:text="${movie.year}"**> Year </**span**></**td**>  
 <**td**><**span th:text="${movie.duration}"**> Year </**span**></**td**>  
 </**tr**>  
</**tbody**>

Чрез **th:each** ще се покажат всички филми, като тагът <tr>…</tr> се повтори толкова пъти, колкото филми съдържа променливата **movies**. Също, ще създава нова променлива на всяка итерация (**movie**) с текущия филм в итерацията. Това е еквивалент на [foreach](https://www.baeldung.com/thymeleaf-iteration) в java (и други езици). От променливата **movies** ще имаме достъп до атрибутите на обекта (title, year, duration). Чрез **th:text** принтираме стойността на променливите. Текста който се намира между <span th:text…> **text** </span> не се визуализира освен ако променливата в **th:text** не е празна Горните редове код ще се компилират от JVM и крайният резултат ще е:

<**tbody**>

<**tr**>  
 <**td**><**span**>Mad Max: Fury Road</**span**></**b**></**td**>  
 <**td**><**span**>2015</**span**></**td**>  
 <**td**><**span**>121</**span**></**td**>  
</**tr**>  
<**tr**>  
 <**td**><**span**>The Wolf of Wall Street</**span**></**td**>  
 <**td**><**span**>2013</**span**></**td**>  
 <**td**><**span**>180</**span**></**td**>  
</**tr**>

. . . . . . .

<**tbody**>

Допълнителните файлове към проекта са:

**Application.properties:** В него се указват всички параметри за Spring приложението:

**spring.jpa.hibernate.ddl-auto**=**none**

JPA има функция за автоматично създаване на таблици и релации само чрез информацията от Entity класовете. В нашия случай го изключваме като указваме стойност none

**spring.datasource.url**=**jdbc:oracle:thin:@localhost:1521:orclcdb  
spring.datasource.username**=**movies  
spring.datasource.password**=**password**

С тези три параметъра указваме връзката с приложението и БД,

**spring.datasource.driver-class-name**=**oracle.jdbc.driver.OracleDriver**

С този параметър указваме кой драйвер ще ползваме. Текущият е за Oracle

1. Демонстрация

