Cinema

Spring MVC & Hibernate

Summary: Let's expand your knowledge of web development using the Spring stack and master Hibernate framework

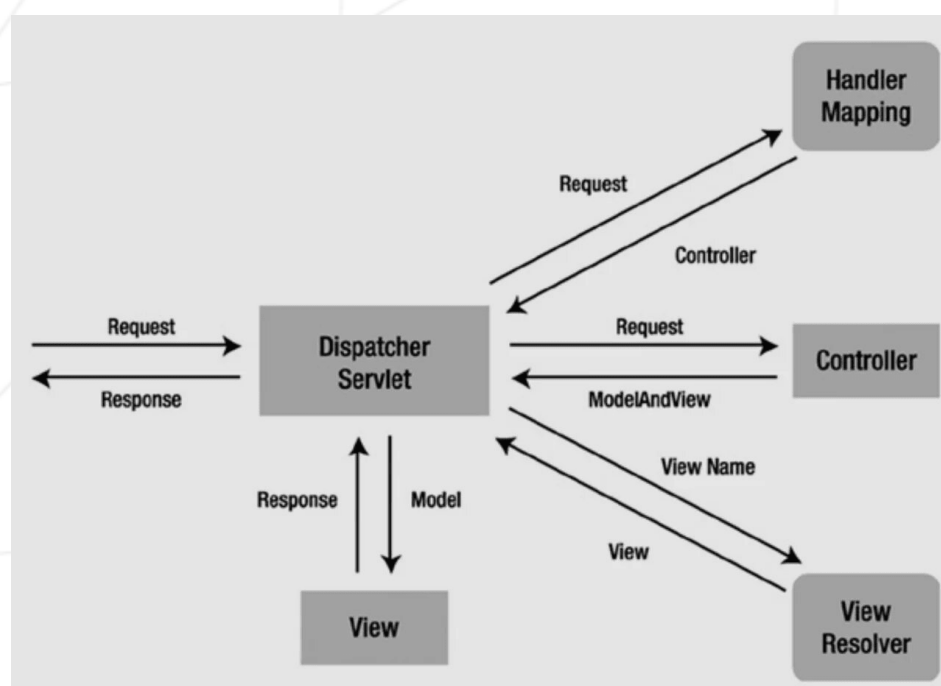*Version: 1*

Contents

Chapter I

Preamble

MVC (Model, View, Controller) is a classic web application design model. This model involves dividing common logic into three components:

- View is what a user (client) sees.

- Model defines the data to be included in View.

- Controller is a request handler that generates a view with required data.

In the context of Spring, MVC model is implemented using classes from an entire set of classes in the spring-webmvc library.

Initially, a request goes to DispatcherServlet. HandlerMapping component is then used to define a Controller that should handle a request. The controller, in turn, generates a ModelAndView object. The latter contains the name of the required view and data that should be used in this View. ViewResolver component will then select a required file with the view (JSP or Freemarker page) and return it to DispatcherServlet. Based on received data, the servlet generates a response for a user.

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.

- Now there is only one Java version for you, 1.8. Make sure that compiler and interpreter of this version are installed on your machine.

- You can use IDE to write and debug the source code.

- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each task, make sure you follow the generally accepted Oracle standards:

- Comments are not allowed in the source code of your solution. They make it difficult to read the code.

- Pay attention to the permissions of your files and directories.

- To be assessed, your solution must be in your GIT repository.

- Your solutions will be evaluated by your piscine mates.

- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.

- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.

- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.

- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.

- Read the examples carefully. They may require things that are not otherwise specified in the subject.

- Use "System.out" for output.

- And may the Force be with you!

- Never leave that till tomorrow which you can do today ;)

Chapter III

Rules of the project

- The project you are implementing now can use a database from the previous task.

- This project does not imply the use of authorization, registration, etc. and shall be implemented independently of the previous task.

- Spring Boot is prohibited in this project.

- For each task, you will need to create a README.txt file with instructions on how to deploy and use your application.

- For each task, you shall attach schema.sql and data.sql files where you describe a schema of a database being created and test data, respectively.

- You can add custom classes and files to each of the projects without breaking the overall suggested structure:

```
Cinema
└── src
    └── main
        ├── java
        │   └── edu.school.21.cinema
        │       ├── config
        │       ├── services
        │       ├── models
        │       ├── repositories
        │       ├── servlets
        │       ├── listeners
        │       └── filters
        ├── resources
        │   └── sql
        │       ├── schema.sql
        │       └── data.sql
        └── webapp
            └── WEB-INF
                ├── application.propreties
                ├── web.xml
                ├── html
                └── jsp
├── pom.xml
└── README.txt
```

Chapter IV

Exercice 00 : Welcome To Controllers & Hibernate

	Exercise 00
	Welcome To Controllers & Hibernate
Turn-in directory : *ex00/*	
Files to turn in : `Cinema-folder`	
Allowed functions : `n/a`	

Now you need to implement a movie theater administrator functionality using Spring MVC mechanisms.
Below are URLs for each of the pages, along with a description of required functionality.

- /admin/panel/halls

The page for working with movie halls contains a list of all movie halls created by an administrator. The administrator can create a movie hall with a certain configuration. To each movie hall, a serial number and number of seats are assigned.
- /admin/panel/films

A movie page contains a list of all movies created by an administrator. An administrator can add a movie. For each movie, the title, year of release, age restrictions, and a description are specified. It is also possible to upload a poster for a movie.
- /admin/panel/sessions

A page for working with movie shows. An administrator can create a session for a certain movie in a certain movie hall at a required time. An administrator should be able to indicate a ticket cost. You should implement loading of all movie and movie hall data as attributes onto the page for subsequent selection by an administrator.

This time, repository layer will be implemented using Hibernate framework in conjunction with JPA. Thus, each of the models shall be annotated with @Entity. An example of a Hibernate/JPA-based repository in Spring context is provided below:

```java
    @Repository
public class MessagesRepositoryEntityManagerImpl implements MessagesRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public List<Message> findAll() {
        return entityManager
.createQuery("from Message", Message.class).getResultList();
    }

    @Override
    @Transactional
    public void save(Message entity) {
        entityManager.persist(entity);
    }
}
```

Technical requirements:

- Each page shall be a Freemarker template.

- Spring controllers should be used instead of servlets.

- Provide for the WebApplicationInitializer implementation and exclude the use of web.xml.
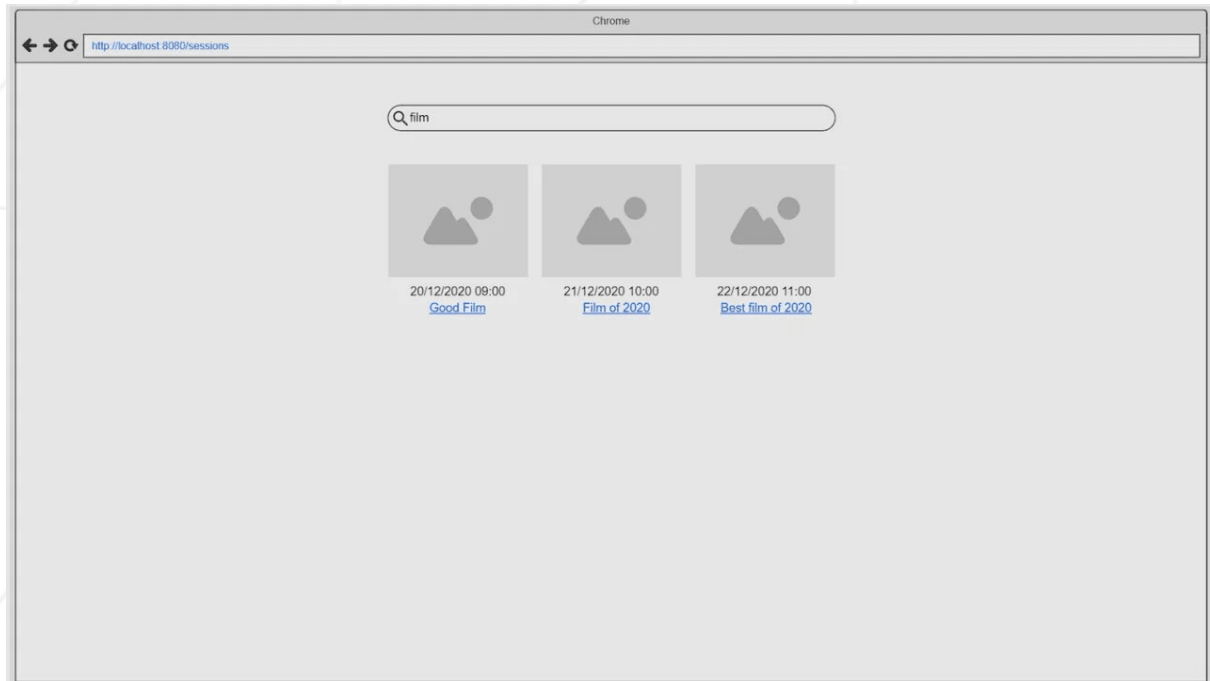
Chapter V

Exercice 01 : Live Search

	Exercise 01	
	Live Search	
Turn-in directory : *ex*01/		
Files to turn in : `Cinema-folder`		
Allowed functions : `n/a`		

You will now learn the new JQuery frontend technology. You will use it to implement
an AJAX request to get a list of all shows according to a search query.

Thus, in response to a /sessions/search?filmName=<movie title> GET request,
the server shall return a list of shows with movie titles matching a search
query. This list is a JSON object in the following form:

```
{
    "sessions":[
      {
        "id":2,
        "dateTime":"20/12/2020 09:00",
        "film":{
          "name":"Good film",
          "posterUrl":"images/874e9d50-5cc8-41b3-90e6-9666ccc80ef1"
        }
      }
    ]
  }
```

On the basis of this data, a list of movie shows is compiled on a user's page
in real time without completely reloading the page during user input. An example
of how the search page works:

- When clicking on a movie title, a user shall follow /sessions/session-id link. On this page, a user will see a full description of the movie, along with information about the movie hall assigned to this show.

Chapter VI

Exercice 02 : WebSockets

	Exercise 02
	WebSockets
Turn-in directory : *ex02/*	
Files to turn in : `Cinema-folder`	
Allowed functions : `n/a`	

For each movie, we will create a chat room with a discussion. Any user who
follows /films/{film-id}/chat link will see a page with a movie discussion
in real time. This page is accessible from the movie show page.

Each chat user is assigned a unique identifier, which is stored in browser's
cookies. Thus, one user will correspond to one browser.
Requirements:

- When this page is opened, the last 20 chat messages shall be loaded.

- Messages shall be available after restarting the application, refreshing pages in
  browsers, etc.

- Messaging shall be implemented using Websockets-based STOMP protocol (see
  Spring Websockets STOMP). Hence, every page subscribes to /films/film-id/chat/messages
  and also sends messages to a specified URL.Each message shall be sent in JSON
  format.

Information about the date / time / IP address of all user authentications
as a list shall also be displayed on this page.

In addition, the page shall have a user's "avatar" loading functionality. To
implement that, you shall provide for processing a POST request to /images
URL. The uploaded image shall be saved to disk. Since users can upload images
in identical files, you shall ensure the uniqueness of file names on the disk.

All uploaded images with their original names shall be available as a list
of links. When a user clicks on the link, the image shall be opened in a new
tab. An example of a profile page interface is shown below:

Example of the interface :
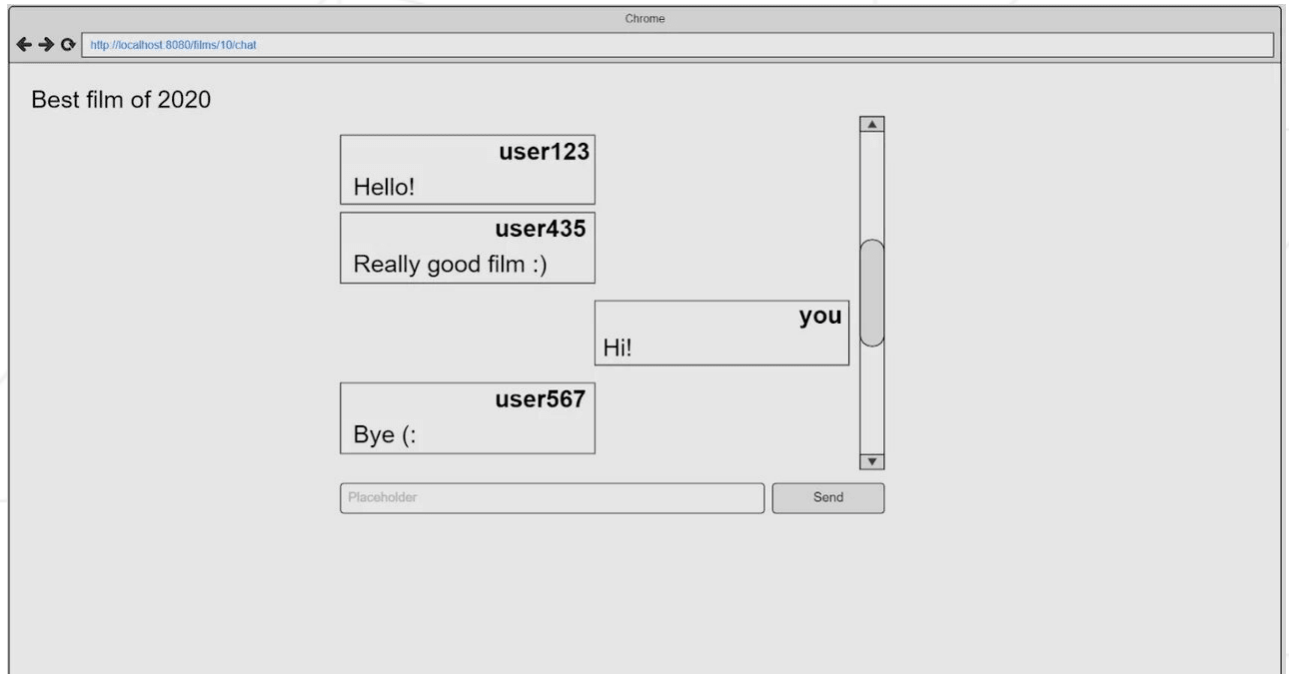
Note:

- It is recommended to use current versions of a few browsers to test this application.