THE SPRING MVC APPLICATION
“VideoKlubProjekat”

1. INTRODUCTION

The application “**VideoKlubProjekat**” has film renting and tracking of various resources as a main goal. This application is dedicated both to the administrator, whose role is to control the whole application, and also to the user for film/media renting.

This document contains a short description of coding technologies used in application design, application rules, configuration and design details, and also guidelines for further application improvements.

This is the English version of the final project design named “***Izrada aplikacije za video klub korišćenjem JAVA programskog jezika i NetBeans platforme***“ (trans. „ Application Design for Video Club using JAVA programming language and NetBeans platform“) within the course „*JAVA Web Programming*“ organized by **Università eCAMPUS**, Novedrate (CO), Italy, in cooperation with Training Center **ESENCA Software,** Belgrade, Serbia (january-december 2016).

2. LIST OF TECHNOLOGIES

“**VideoKlubProjekat**” is the web application which demonstrates a usage of the following technologies:
- Spring MVC framework (MVC pattern) – business layer
- Hibernate (ORM)
- MySQL – RDBMS data layer
- JSP + JSTL + different TAG libraries: presentational (View) layer
- Bootstrap + Jquery: front-end framework (css + js)

A standard JavaEE Servlet container is used as server platform.
Communication protocol is “pure” HTTP without AJAX.
Every web page generated by server has its own URL (Permalink).

For application design the NetBeans IDE has been used.

(tutorials and information about these technologies [1]..[13]).

Java code is jdk 1.7 – compliant.

3. APPLICATION'S FUNCTIONALITY AND RULES

There are two roles in this app.:

a. Administrator role:
- Has the complete view of all films
- Rents available films
- Returns rented films
- Performs CRUD operations on films
- Has the complete view of all users (users with/without rented films)
- Performs CRUD operations on users
- Has the statistics view of users and rented films

b. User role:
- Has the complete view of his/her own rented films
- Can rent available films

NOTE:
- User rents films by application and after that he physically takes over the film.
- (S)He cannot return film(s) by application. He returns films physically to administrator, then, administrator returns film(s) by application.

Application Controls:
- The renting time (for example 7 days, but for demonstration purposes here it has been downsized to 10h)
- Max number of renting films per user (max 5 films per user at the same time).
- Login data
- User page access
- Simultaneous work admin/user
- Simultaneous work user/user

4. DESIGN OF JAVA WEB APPLICATION "VideoKlubProjekat"

Application has several parts, or tiers, that are inter-connected:

- First step was to create a (MySQL) database "**dbvideo_club**".
- Then, "**VideoKlubProjekat**" as a Java Web application was designed. The integration of Hibernate and Spring framework was performed.
- For web pages views, jsp files were used (with HTML code); CSS resources and Bootstrap front-end framework was used alongside.

4.1. DESIGN OF DATABASE

"**dbvideo_club**" database schema is given in the next figure:

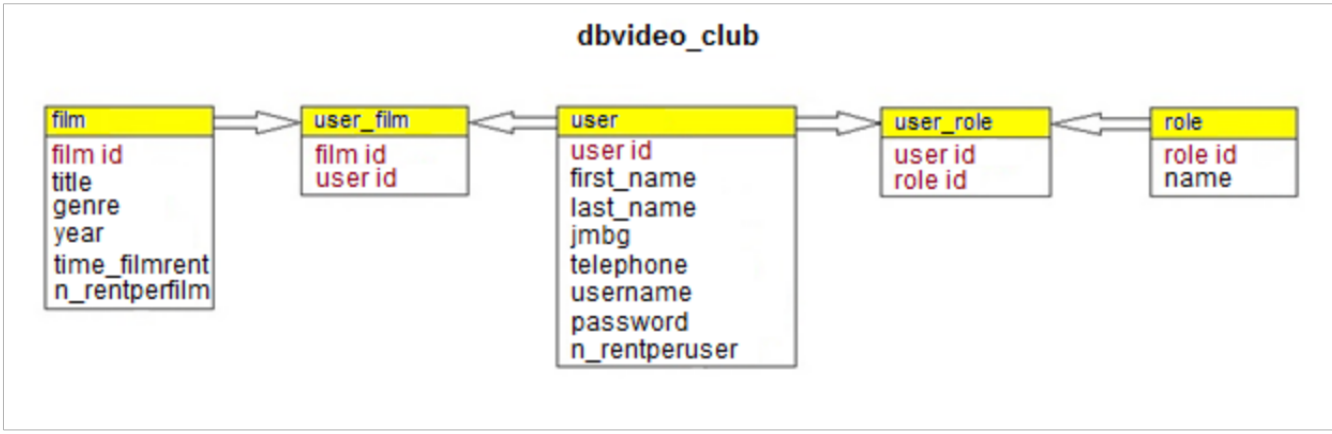

*Figure 1 - **"dbvideo_club"** database schema*

For tables '**film**' and '**user**', many-to-many mapping is used [14]. For this application real case is one-to-many (one user – more films), but concept many-to-many is used in order to expand the basic functionality (such as renting multimedia sources).

The same concept is applied for '**user**' and '**role**' tables. (For improved functionality of user role and access control you can use another Spring module – Spring Security [15]).

There are two join tables: '**user_film**' and '**user_role**'.

SQL script file **dbvideo_club.sql** is given in folder **sql**.

1.For application start we need to populate table '**role**':
INSERT INTO 'dbvideo_club'. 'role' ('id', 'name') VALUES ('1', 'ADMIN')
INSERT INTO 'dbvideo_club'. 'role' ('id', 'name') VALUES ('2', 'USER')

2. In table '**user**' we populate data for the admin:
INSERT INTO 'dbvideo_club'. 'user' ('id', 'username', 'password', 'first_name', 'last_name', 'jmbg', 'telephone') VALUES ('1', 'sneza', 'snezana', 'Snežana', 'Snežić', '1111111111', '123-1234')

jmbg – unique personal ID number (in Serbia - 13 digits, different for other countries). For demonstration purposes this number has 10 digits examples.

3. In table 'user_role' we insert id of admin and id of his role (ADMIN):
INSERT INTO 'dbvideo_club'. 'user_role' ('usr_id', 'rol_id') VALUES ('1', '1')

4.2. PROJECT CREATIONS AND XML FILES CONFIGURATIONS

Java Web application ‘VideoKlubProjekat’ is created and developed in NetBeans IDE.

Entry parameters to IDE:

- Server: Apache Tomcat
- Java EE version: Java EE 7 Web
- Backend framework: Spring MVC (libraries Spring Framework 4.0.1 and JSTL)
- Persistence framework: Hibernate 4.3.1
- Database connection: MySQL(Connector/J) driver
  - Customize Connection – database name: dbvideo_club
  - Database dialect: org.hibernate.dialect.MySQLDialect

With these parameters, NetBeans creates skeleton application automatically. In directory **WEB-INF** there a subdirectory **jsp** is created, as well as file **index.jsp** (a welcome file).

4.3. CONFIGURING XML FILES AND BEANS

In **WEB-INF** directory, default xml files for application configurations are created automatically. Spring beans definitions are contained in these files. (More about xml configurations and beans creations [25], [26], [27], [28], [29], [30]).

There are three xml files:

- **web.xml** for web modules deployment into Tomcat servlet container; entry point for Spring MVC application.
- **dispatcher-servlet.xml** for resources and controllers configurations
- **applicationContext.xml** for database connection

4.4. CONFIGURING HIBERNATE XML FILES

NetBeans automatically creates **hibernate.cfg.xml** file in **<default package>** subdirectory [31]. This file consists information about resources that should be mapped.

In the same directory we need to create **hibernate.reveng.xml** file choosing all tables that we want for our POJO objects.

4.5. DIRECTORIES CREATED IN SOURCE PACKAGES

This application uses DAO and MVC Design Pattern.

DAO (Data Access Object) Design Pattern

DAO elements are [32]:

1. 1. DAO – interface in package **com.snezana.dao** has files:
   FilmDAO.java
   UserDAO.java

2. DAO – implementation classes in package **com.snezana.daoImpl** with files:
   FilmDAOImpl.java
   UserDAOImpl.java

3. Model objects – POJO objects in package **com.snezana.models**. POJO classes and hibernate mapping files (**hbm.xml**) in **com.snezana.models** package are automatically created by NetBeans. These are the following files:
   - **Film.hbm.xml**
   - **Film.java**
   - **Role.hbm.xml**
   - **Role.java**
   - **User.hbm.xml**
   - **User.java**

MVC (Model View Controller) Design Pattern

MVC has tree components:

Model – previously mentioned POJO objects in package **com.snezana.models**

View – resolved views with **jsp** files in **WEB-INF/jsp** directory

Controller - responsible for user request processing, model generation and their distribution to the appropriate view. This pattern determines (together with Front Controller Design Pattern) Spring MVC architecture design [33], [34]. In package **com.snezana.controllers** the following files/classes are dedicated to that role:
FilmController.java
UserController.java

4.6. OTHER DIRECTORIES FOR SUPPORT

- **WEB-INF/resources** contains CSS files, images and **.js** files for front-end support
- Also, in **Remote Files** directory there is support for Bootstrap functionality
- Additional directory **com.snezana.util** contains support files for controllers (**ControllerUtils.java**) and UTF-8 character set (**CharacterSetFilter.java**).

Libraries used in this application are shown in the next figure:

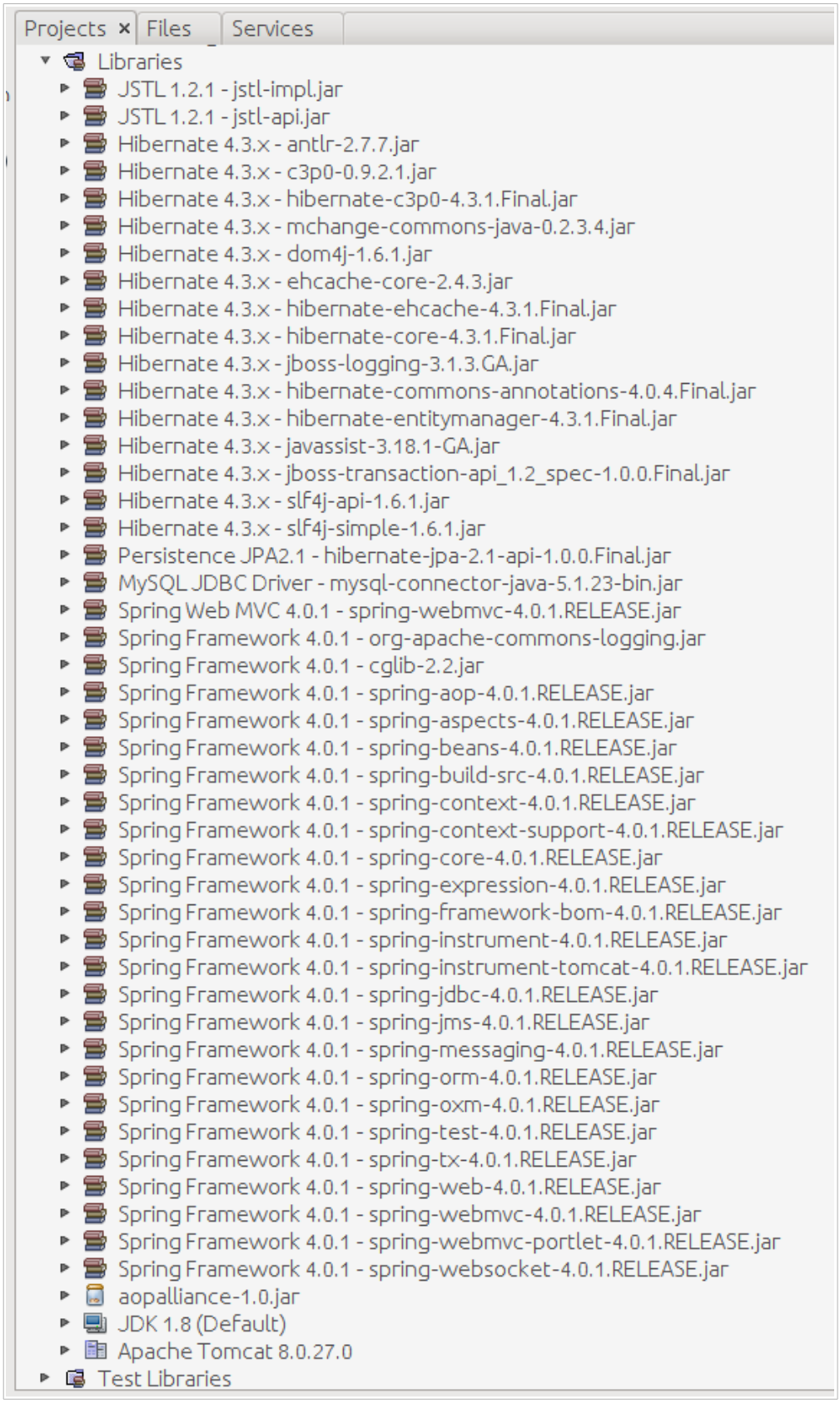

*Figure 2 – Libraries in **'VideoKlubProjekat'** application*

The architecture of '**VideoKlubProjekat**' application in NetBeans IDE is:

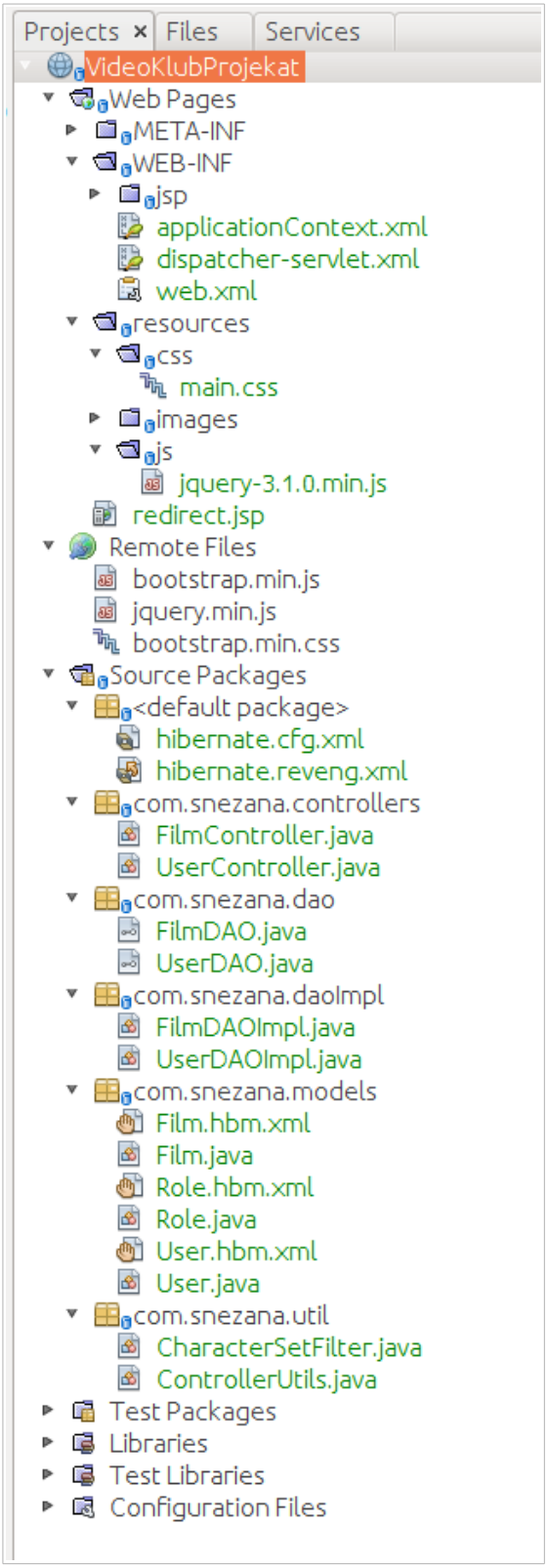

*Figure 3 – Architecture of **'VideoKlubProjekat'** application*

5. DESIGN PARTS

5.1. MODELS

Film.java, **User.java** and **Role.java** are classes with POJO objects.

Film.java contains these fields:
`Long` **`rentTime`** – that represents the duration period of renting; If this film is not rented the value is **rentTime = -1**.
`Integer` **`numRents`** – represents the number of renting
`Set<User>` **`users`** – the collection of users; this field is automatically generated by NetBeans as a consequence of making relation between tables '**film**' - '**user**'.

Also, there are static methods: **currentTime()**, convertTime(long time), remainningTime(long time1, long time2) and isRentalTimeExpired(long time1, long time2). These additional methods are used as time functions in renting and time formats for presentation (according to [20], [21], [22]).

User.java has fields:
Long **jmbg** – unique number of a citizen, previously explained in database part
Integer **rentedFilms** – the number of rented films in total that user achieved
Set<Role> **roles** and Set<Film> **films** – collections, generated automatically by NetBeans as a consequence of making relation between tables '**user**' - '**roles**' and tables '**film**' - '**user**', respectively.

Role.java defines String **name** ={ADMIN, USER}.
Set<User> **users** – collection of users, generated automatically by NetBeans.

5.2. DAO INTERFACE AND IMPLEMENTATION

The methods involved in DAO interface are commented in UserDAO.java and FilmDAO.java. Their implementation is given in UserDAOImpl.java and FilmDAOImpl.java, respectively.

5.3. CONTROLLERS

Controllers are a crucial part of this application. It contains two controllers: **FilmController.java** and **UserController.java**. Every controller has its own service methods that process data taken from database or prepare data for insertion into database, and returns view names to the DispatcherServlet for view resolving.
Service methods of each particular controller have javadoc comments that describe their operation.

For tracking the user on the Web, the HTTP session is introduced (more about that [16], [17], [18], [19]).

Additional methods for user role checking and user redirection (given in **ControllerUtils.java**, **com.snezana.util** directory) are used in controller's service methods.

5.4. JSP PAGES

JSP (Java Server Pages) server-side technology, enables creation of dynamic content in Web applications, in a platform-independent manner.

For complete application functionality we also need view resolving. A good explanation is given in [23].

The next picture represents content of the **WEB-INF/jsp** directory:

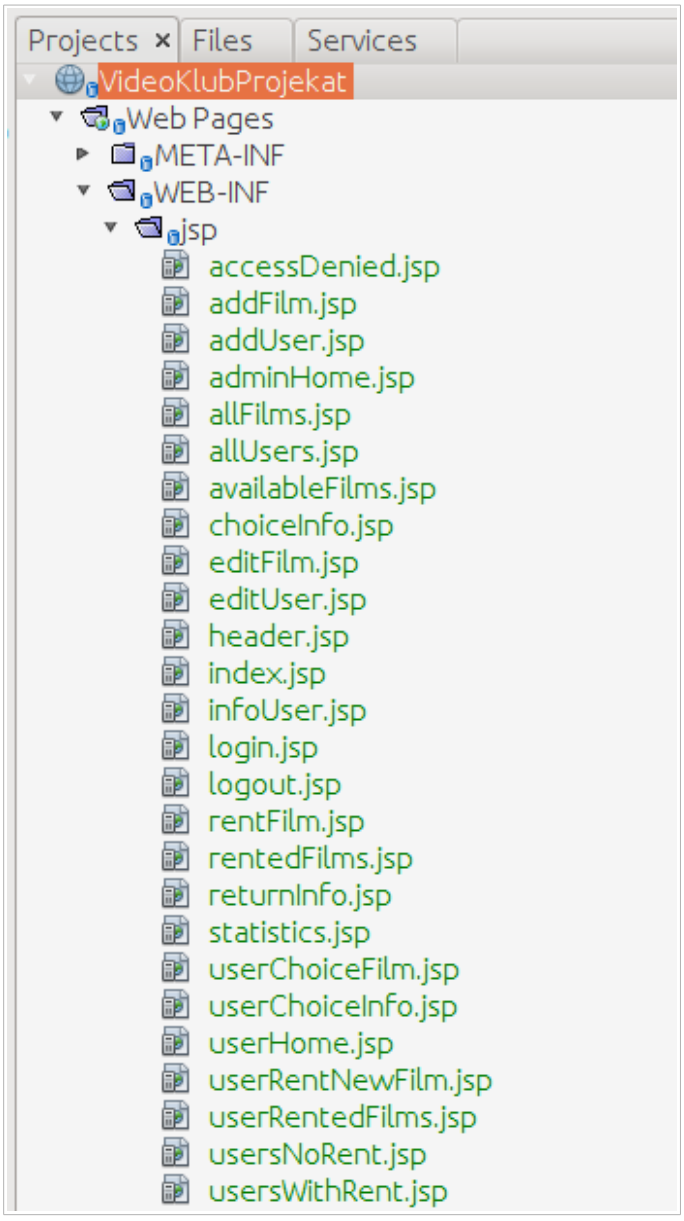

*Figure 4 – jsp pages in **'VideoKlubProjekat'** application*

We can include resources (css, images and js files) into the JSP with Spring MVC in a way given in [36], [37].

This app contains the following jsp pages:

General jsp pages:
- **index.jsp** – welcome application page
- **login.jsp** – entrance form with data checking
- **logout.jsp** – information about successful logout
- **header.jsp** – basic header

Admin pages:
- **adminHome.jsp** – the main admin page with links to other admin pages
- **addFilm.jsp** – form for new film input data
- **addUser.jsp** – form for new user input data
- **allFilms.jsp** – list of all films
- **allUsers.jsp** – list of all users
- **availableFilms.jsp** – list of all available films that you can rent
- **choiceInfo.jsp** – information about user and film that he has rented
- **editFilm.jsp** – updates film data form
- **editUser.jsp** - updates user data form
- **infoUser.jsp** - user data information
- **rentFilm.jsp** – user selection for film renting
- **rentedFilms.jsp** – list of rented films with return functionality
- **returnInfo.jsp** - information about user and film that he has returned
- **statistics.jsp** – statistics data
- **usersNoRent.jsp** – list of users without rented films
- **usersWithRent.jsp** – list of users with their rented films

User pages:
- **userHome.jsp** – the main user page with links to other user pages
- **userChoiceFilm.jsp** - information about film that he has rented
- **userChoiceInfo.jsp** – query about film status and possibility of renting
- **userRentNewFilm.jsp** – list of available films that he can rent
- **userRentedFilms.jsp** – review of films that he has rented
- **accessDenied.jsp** – information about access attempt to admin pages

6. GUIDELINES FOR FURTHER APPLICATION IMPROVEMENT

- In original version there is no functionality for UTF-8 character set encoding. This version is prepared with this feature. (More about that [38]).
- introducing Spring Security
- input data validation
- reducing the number of jsp pages
- controller reorganisation for better performances

- moving time functions from Film.java to utility methods
- improving simultaneous work admin/user and user/user with SpringWebsocket functionality
- introducing the session timeout in another way (for automatic redirection to logout page after session timeout, js solution [24], or with Spring Security with additional functionality – to get currently logged – in users [35]).

7. CONCLUSION

In this document the development of Spring MVC application "**VideoKlubProjekat**" in NetBeans IDE is presented with a list of used technologies. In consecutive steps the database is formed, several application files are configured, additional/helper files are created and formed according to the DAO and MVC patterns, and the application front-end is implemented through various jsp files. Finally, the set of guidelines for further application improvement is given.

8. REFERENCES

[1] http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/

[2] https://spring.io/docs

[3] https://www.tutorialspoint.com/spring/

[4] https://www.tutorialspoint.com/hibernate/

[5] http://www.javatpoint.com/spring-tutorial

[6] http://www.javatpoint.com/hibernate-tutorial

[7] http://www.w3schools.com/html/

[8] http://www.w3schools.com/css/

[9] http://www.w3schools.com/js/

[10] http://www.w3schools.com/sql/

[11] http://getbootstrap.com/getting-started/

[12] http://getbootstrap.com/css/

[13] http://www.w3schools.com/bootstrap/

[14] http://www.journaldev.com/2934/hibernate-many-to-many-mapping-join-tables

[15] https://hellokoding.com/registration-and-login-example-with-spring-xml-configuration-maven-jsp-and-mysql/

[16] http://tutorials.jenkov.com/java-servlets/httpsession.html

[17] http://stackoverflow.com/questions/33668275/how-to-manage-session-in-spring-web-services

[18] http://www.javaroots.com/2013/08/how-to-get-session-object-in-spring-mvc.html

[19] https://dzone.com/articles/using-http-session-spring

[20] http://stackoverflow.com/questions/6782185/convert-timestamp-long-to-normal-date-format

[21] https://www.mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/

[22] https://www.mkyong.com/java/how-to-calculate-date-time-difference-in-java/

[23] https://dzone.com/tutorials/java/spring/spring-mvc-tutorial-1.html

[24] http://www.icesoft.org/JForum/posts/list/21742.page#sthash.z5MG5ipe.dpbs

[25] http://www.javatpoint.com/hibernate-and-spring-integration

[26] https://spring.io/understanding/application-context

[27] https://www.quora.com/How-would-you-explain-dispatcher-servlet-xml-applicationContext-xml-web-xml-and-spring-servlet-xml-to-a-NOVICE-J2EE-Java-programmer-and-the-relationship-between-these-xmls-in-a-Spring-Web-App

[28] http://shengwangi.blogspot.rs/2015/08/understand-webxml-in-spring-mvc-project.html

[29] https://stackoverflow.com/questions/3652090/difference-between-applicationcontext-xml-and-spring-servlet-xml-in-spring-frame/3652125

[30] http://www.codejava.net/frameworks/spring/spring-4-and-hibernate-4-integration-tutorial-part-1-xml-configuration

[31] https://netbeans.org/kb/docs/web/hibernate-webapp.html

[32] https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

[33] http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

[34] http://www.codejava.net/frameworks/spring/understanding-spring-mvc?showall=&start=1

[35] https://www.mkyong.com/spring-security/get-current-logged-in-username-in-spring-security/

[36] https://www.mkyong.com/spring-mvc/spring-mvc-how-to-include-js-or-css-files-in-a-jsp-page/

[37] http://crunchify.com/spring-mvc-4-2-2-best-way-to-integrate-js-and-css-file-in-jsp-file-using-mvcresources-mapping/

[38] https://www.baeldung.com/tomcat-utf-8

NOTE: Last access to the references: 20th September 2018.