

BLACK JACK

PROJECT REPORT

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF
GRADE IN

SECURE SOFTWARE DESIGN AND PROGRAMMING

IN

COMPUTER SCIENCE DEPARTMENT

UNDER THE GUIDANCE OF

Dr. DAVID A WHEELER

BY

NEHA REDDY ALLAM (G01001432)

PADMINI CHILAPPAGARI (G00836190)

MADHURIMA DOPPALAPUDI (G01035890)



DEPARTMENT OF COMPUTER SCIENCE

GEORGE MASON UNIVERSITY

FAIRFAX, VIRGINIA

1.INTRODUCTION:

Black jack, also known as **twenty-one**, is the most popular casino banking game in the world. It is a comparing card game between a player and dealer but not against other players. The game follows a pattern of beating the dealer by getting 21 points or by waiting till the dealer exceeds 21 or by reaching the final score without exceeding 21. We have improvised this basic idea, and developed a graphical game adding multiplayer functionality which is more secure and robust. This game follows the concept of adding all the players into the database through user login or registration pages connected to the database, that will authenticate the player to start the game. The player has four options such as "hit", "stand", "double", "split" after the dealer starts the game. The dealer receives first card faced up and the second card as closed card which only dealer can see. The players has both the card faced up, and can add as many number of the cards as the player want until it bursts (>21). The win/loss of each player is updated into the database in the form of records, which can be viewed by each player when he accesses the game.

2.ARCHITECTURE:

The game was developed using Spring MVC architecture and Maven as the build automation tool. We used JSP pages, JavaScript on the frontend, Java as our programming language for business logic, hibernate for interacting with MySQL database system. The server we used to deploy our application is Apache Tomcat version 7.0.

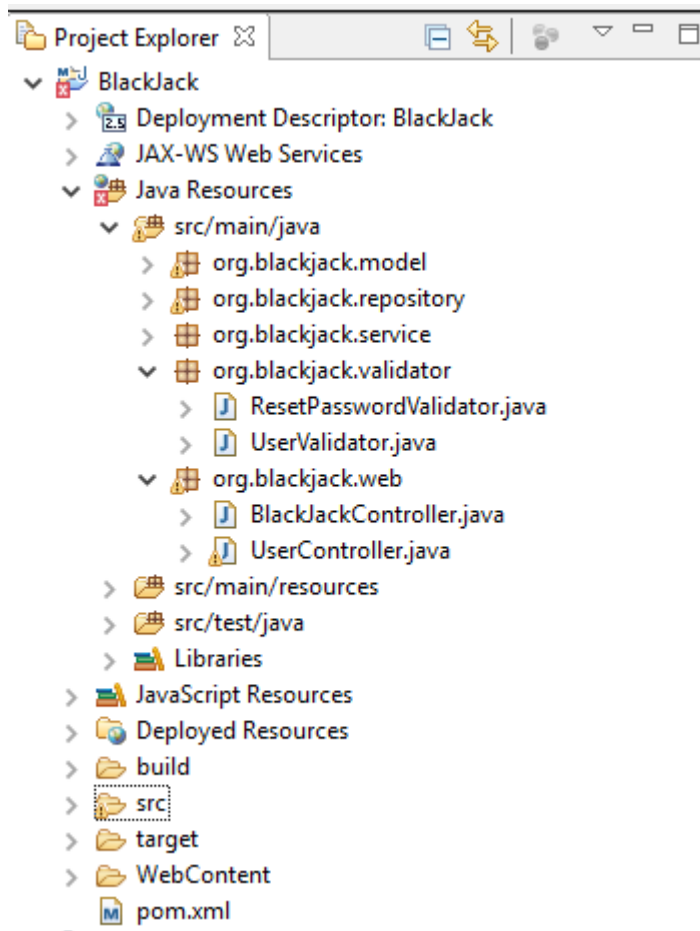
To implement the registration and login modules, we used Spring Security framework, Hibernate for database connectivity, XML configuration and build tool as Maven.

Spring MVC framework is an application framework which provides model-view-controller architecture and ready components that can be used to develop flexible web applications. It contains many modules and they are designed around DispatcherServlet which handles all the requests around servlets. Spring MVC framework supports Spring Security framework and Spring Data JPA. Spring Security focuses on providing both authentication and authorization to java applications. It also contains CRSF token implementation which is used for implementing security.

Hibernate provides object/relational mapping(ORM) that is concerned with data persistence. Maven is a build automation tool used to store jar files. Hibernate's primary feature is a mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

Maven is used for building our application. All required jar files are defined as dependencies in pom.xml file and this ensures that all required jar files are incorporated in the system.

Project Structure:



Design:

The components of our project are:

1. Web pages: Used HTML and CSS files
2. Spring MVC: Application framework which provides model-view-controller architecture for building web application.
3. JSP: Used as a server side scripting language used to create dynamically interactive web pages and to store and retrieve information from the database.
4. Java Programs: Used java programs to code the basic functionality of the game.
5. Database MySQL: Used to store the data.
6. Spring Hibernate: Handled requests around the servlets and ORM that is concerned with data persistence.
7. Maven: Automation build tool for our application.
8. Apache Tomcat 7.0: Server

Implementation:

The major components of the game are:

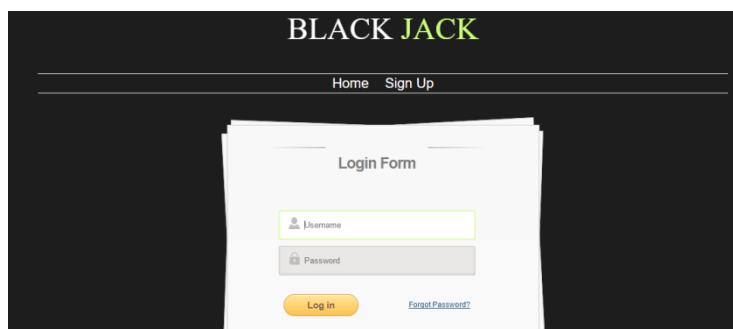
1. Login/Registration
2. Authenticated user's statistics
3. Start new game
4. Play game
5. Double game
6. Split game
7. Exit game
8. Forgot Password module for password reset

1.Login/Registration:

Users have to either login or register in order to play the game. The unregistered users can create an account and the registered users can login directly to the game. The authentication technique in our project is used to validate the user. All the validations are done with the help of regular expressions. The registration page will allow one user to register only once and asks to login if he has registered already. The regular expressions validate the user inputs such as email (invalid emails will be rejected), strong passwords (password format should be followed). Encrypted passwords are stored in the database securely. We have implemented defence in depth mechanism to protect the security level of our system. Implemented SSL/TLS protocols to establish secure connection between client and the server.

Files used for this implementation are:

1. WEB-INF/login.jsp
2. WEB-INF/registration.jsp
3. org.blackjack.model/User.java, org.blackjack.model/Role.java
4. org.blackjack.repository/UserRepository.java,
org.blackjack.repository/RoleRepository.java
5. org.blackjack.validator/UserValidator.java
6. org.blackjack.web/UserController.java



2.Authenticated User's statistics:

All the registered users list is stored in MySQL database. Authenticating the users using Spring framework. Hibernate object-relational mapping is used to map tables to entities and methods are used to access the database.

Files used for this implementation are:

1. org.blackjack.web/UserController.java
2. org.blackjack.validator/UserValidator.java
3. org.blackjack.service.SecurityServiceImpl.java

3.Start new game:

As soon as the user logs into the application, they can start the game by clicking on the new game button or view their last 5 game results or log out of the game. This page helps the user to join any existing games or create a new one.

Files used for this implementation are:

1. org/blackjack/web/BlackJackController.java
2. org/blackjack/service/BlackJackService.java
3. org/blackjack/service/BlackJackServiceImpl.java

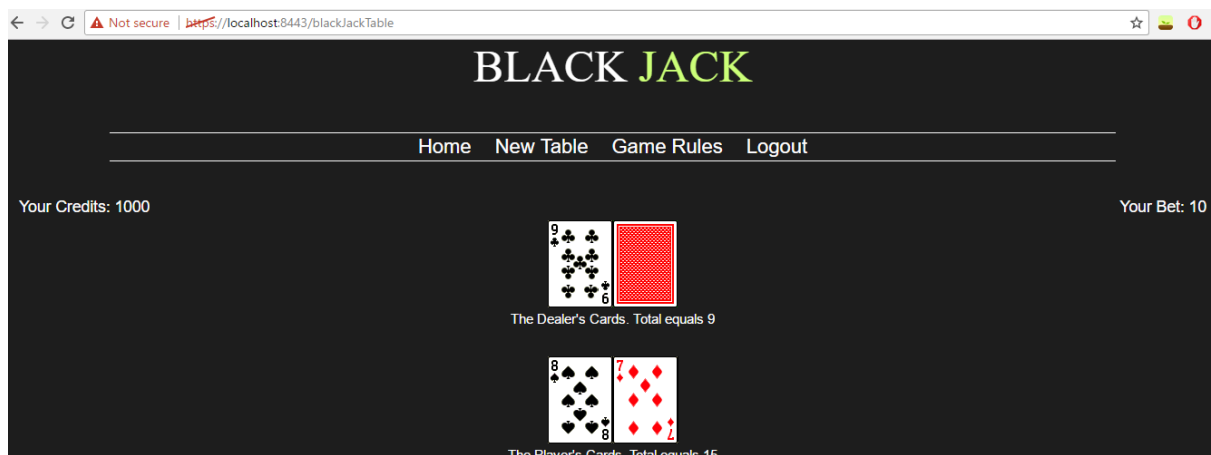
4. Play game: "New Table"

This is the key component of our project that holds the major functionality of our application. After a user starts the new game the dealer shuffles the cards and distributes the cards among dealer and users. There will be two open cards for the users, one open card and a close card for the dealer (the close card only the dealer can peek into). Whoever has the highest score among the dealer and the user wins. As the game is for 21, whoever crosses 21 losses the game. There is an option to get more cards till the players wants it to stop or if it "Busts" (crossing 21). After finishing the game the user will be notified with the status of the game if he won or loose. We store the statistics of the users in our database.

Files used for this implementation are:

1. org/blackjack/model/Game.java, org/blackjack/model/Card.java, org/blackjack/model/Deck.java, org/blackjack/model/Deck.java, org/blackjack/model/Suit.java, org/blackjack/model/Round.java
2. org/blackjack/repository/GameRepository.java
3. org/blackjack/service/BlackJackService.java, org/blackjack/service/BlackJackServiceImpl.java, org/blackjack/service/GameService.java, org/blackjack/service/GameServiceImpl.java

4. org/blackjack/web/BlackJackController.java



5. Double game:

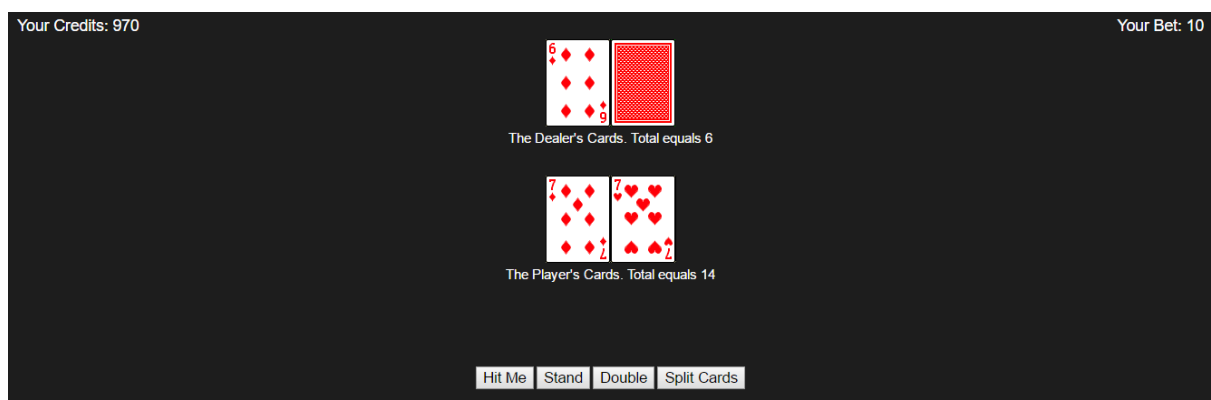
The double option in the game will help the user to double the betting for single game. Files used for this implementation are the same as used for the previous module.

6. Split game:

The split option in the game is used when the player get two same cards of 10, Jack, Queen or King. The user can play two games simultaneously when he gets this option. He should bet the same amount twice while he split the game. He will be playing two games and winning either of them will make him win in the game. We will store the statistics of these two games under the same person that is playing.

Files used for this implementation are the same as the ones used for playing the game. (point 4). The additional file used for split functionality is:

1. org/blackjack/model/SplitHand.java



7. Logout game:

Logout option in the game helps the user to exit the game anytime he wants. This is a button played in the Game UI.

Files used for this implementation are:

1. org/blackjack/web/UserController.java
2. org/blackjack/repository/UserRepository.java
3. org/blackjack/model/User.java

8. Forgot Password Functionality:

This functionality is present to reset the user's password if the user forgets it. An email is sent to the user to reset the password. A hashed token of the user's email address is appended to the url and sent to reset the password.

Example URL:

<https://localhost:8443/newPassword/336627be-cac1-49b7-9347-de1d98efb1a6/>

Files used for this implementation:

1. org/blackjack/validator/ResetPasswordValidator.java
2. org/blackjack/service/MailService.java
3. /org/blackjack/web/UserController.java

3.INSTALLATION INSTRUCTIONS:

1. The project needs Eclipse IDE and Maven support. Maven can be installed using **m2eclipse** plugin.
2. Open Eclipse IDE and click on File>Import >option. Select 'Maven Projects' option.
3. Browse project location and select the zip folder. Maven project is created.
4. For building the application, right click on the project to open context menu. Select 'Run' with 'maven' option. Maven will start building the project.
5. Now right click on project and select 'Run on server' option.
6. Hit the url "<https://localhost:8443/login>" on browser. A security warning appears on the page as it is SSL enabled. Click on "Run with exceptions" to view the login page.

4.OPERATING INSTRUCTIONS:

1. Hit the url "<https://localhost:8443/login>" on browser. A security warning appears on the page as it is SSL enabled. Click on "Run with exceptions" to view the login page.
2. If you are a new user, click on the "Register" link present on the login page. Create an account by entering a username and password.
3. If you are an existing user, enter valid credentials and login to the system.
4. Once the user is logged in, he/she will be able to see three options : Home, Game rules and New Table.
5. Click on 'Game Rules' for checking game rules before playing a game.
6. By clicking on 'New Table', the user will be able to see dealer's cards, player's cards and an option to select a betting amount. The user will have the options of 'Hit', 'Stand', 'Double' and 'Split' – if same cards show up.
7. After playing a game by selecting a betting amount, the user has the option to start a new game or check his/her game statistics.
8. To check the game statistics, click on 'Home' tab. Details such as current credit(balance), betting amount, result of the game. For each game, usage of 'Split' functionality and results of left and right split games are also displayed.
9. The user can log out of the system by clicking on "Log Out" button.

5.GAME RULES:

1. The user can login or register depending on the status of the user.
2. User can click on new game or join the game after he logs in.
3. The user should play with the dealer and whoever is closest to 21 wins.
4. The value of Ace is 1 or 11, The value of Jack, Queen, King is 10.
5. If you go over 21, you are "Bust" and you lose.
6. The dealer must stand on a hand of 17 to 21, and has to hit on a hand of 16 or less.
7. A normal win pays 1:1.
8. A win with a blackjack (Ace and 10 or royal) pays 3:2.
9. A draw (called a push in BlackJack) results in no loss or gain for the player.
10. After the opening cards have been dealt, the player can double down to double the bet, but only receives one card.
11. The dealer must stand if the dealer has an ace that, when treated as 11, brings the dealer's hand to between 17 and 21, e.g. a 6 and an ace.
12. If the player has matching cards, the player can split the cards and play them as two separate hands.
13. Jack, Queen and King, 10 are all considered matching cards for the purposes of a split.
14. Winning with a BlackJack only pays 1:1 after the player has split.
15. Unlike in casinos, a single new deck is used for each game.

6. HOW DO WE PROVIDE SECURITY AT EVERY LEVEL IN OUR GAME?

Our game Blackjack is a web application game and this can be run on any web server. And a game which is out on the network will be prone to many attacks. In the development of our project, we made sure security is provided at every level to implement a defense in depth mechanism for our security system. The following shows what tool was taken up to handle this and how this is applied in our project.

6.1 HOW DO WE ENSURE LOGIN IS SECURE

This is going to be the first step where the user would interact with the system. Here the user will have to give his login username and password if he is already an existing user or will have to create a new account if he is new to the game. For ensuring security in this first step of the game, we have implemented the following:

6.1.1 Confidentiality -

This is the first step to ensure authorized users are accessing the game. First when the user enters the user name and password, the page would be redirected to fetch values from the database. Only if the user has the authorized credentials he will be allowed to play the game else would be stopped. For this a password encrypted page is coded. If a new user is attempting to create an account, the validator will make sure to take in all the details, verify them, and then update them onto the database. This would be the first step for authentication.

For every user, the details that are entered there will be a method in the Spring security configuration to ensure that the user exists and only then access is given to the user. The following is the method in the SecurityServiceImpl class

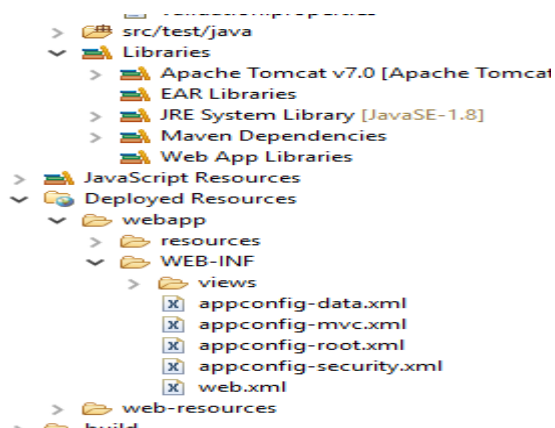
```
public String findLoggedInUsername()
{
    Object userDetails = SecurityContextHolder.getContext().getAuthentication().getDetails();
    if (userDetails instanceof UserDetails)
    {
        return ((UserDetails)userDetails).getUsername();
    }
    return null;
}
```

6.1.2 Spring Security:

Spring is used to provide a framework that is configured and is implemented for authentication and authorization of data. This is mainly used to provide security for logging in and logging out of a web application securely. The authentication for data is present at two levels for this, at the view level and the method level. Using Spring Security for our web application will ensure the following:

1. Our application will provide the capability for logging and logging out securely.
2. It will provide appropriate access controls to the game link depending on the role of the user.
3. It will first check for the authorization of the user and only then provide the ability to view certain portion of a page. It will hide the content, if a user does not have the necessary privileges.
4. Spring security will link to a database or LDAP for authentication.

In our project directory, we have embedded this spring security at the following location:



The services that are provided by the Spring Security framework are the following.

- UserDetailsService – This is used to implement the login and the authentication for it.
- SecurityService – This is used to provide access to the current users who are logged in and gives the permission to users to login in if they have created the account. A secure way to give access to this code, is done in the following method:

```
public void autologin(String username, String password)
{
    UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
    UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(userDetails, password,
userDetails.getAuthorities());

    authenticationManager.authenticate(usernamePasswordAuthenticationToken);
}
```

```
if (usernamePasswordAuthenticationToken.isAuthenticated()) {
```

```
SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthentic  
ationToken);
```

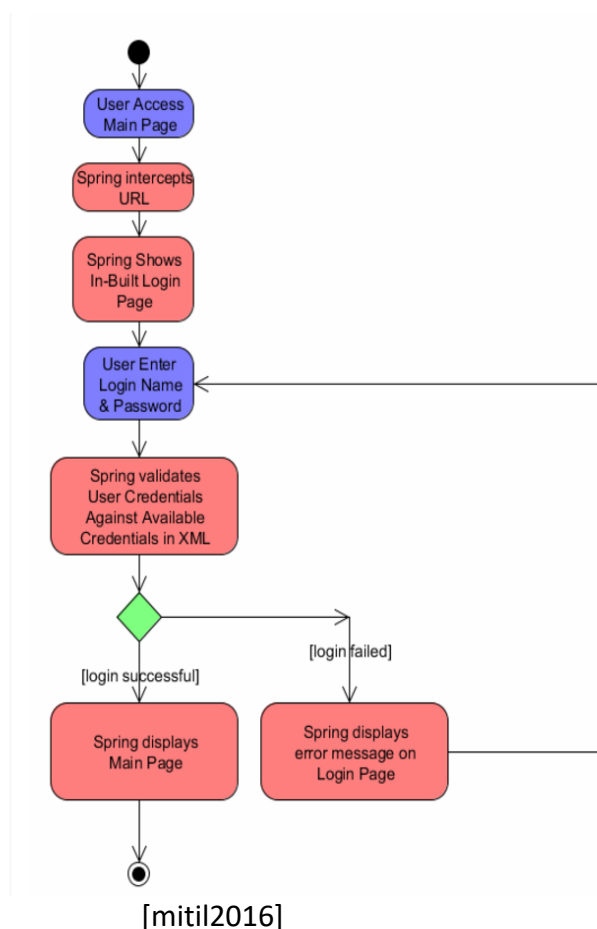
```
logger.debug(String.format("Auto login %s successfully!", username));}
```

- UserService – This special class is implemented to give the service to a user to register as a new user and create an account.

```
public void save(User user)  
{  
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));  
    user.setRoles(new HashSet<>(roleRepository.findAll()));  
    userRepository.save(user);  
}
```

- SpringValidator – This special class is used to perform the input data validation when the user registers to create a new account. The Spring framework provides validation.properties to make sure the user data is validated.

Steps to show how Spring Security is implemented in our project:



For this same purpose, to implement the login authentication, org.springframework.security.core.userdetails.UserDetailsService interface is implemented. And in this file, the authorization is implemented using appconfig-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd">

    <http auto-config="true">
        <intercept-url pattern="/" access="hasRole('ROLE_USER')"/>
        <intercept-url pattern="/welcome" access="hasRole('ROLE_USER')"/>
        <form-login login-page="/login" default-target-url="/welcome" authentication-failure-
url="/login?error" username-parameter="username" password-parameter="password"/>
        <logout logout-success-url="/login?logout" />
    </http>

    <authentication-manager alias="authenticationManager">
        <authentication-provider user-service-ref="userDetailsServiceImpl">
            <password-encoder ref="encoder"></password-encoder>
        </authentication-provider>
    </authentication-manager>

    <beans:beanid="userDetailsServiceImpl"
class="com.hellokoding.account.service.UserDetailsServiceImpl"></beans:bean>
    <beans:bean id="encoder"
        class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder">
        <beans:constructor-arg name="strength" value="11"/>
    </beans:bean>
</beans:beans>
```

6.1.3 SSL/TLS:

SSL/TLS has been implemented and configured on the server. The following command has been used to generate the key tool.

```
Keytool -genkey -alias testKey1 -keyalg RSA -keystore
C:\Users\Padmini\sslCertificates\testKey1
```

testKey1 is the certificate name.

For enabling this SSL, the following code has been added to the server.xml file

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
```

```

clientAuth="false" sslProtocol="TLS"
keystoreFile="C:\Users\Padmini\sslCertificates\testKey1"
keystorePass="testkey1" />

```

6.1.4 Using CSRF for protecting login page -

To prevent from unauthorized users or attackers appending data and performing the Cross – Site request forgery. For every input that is taken in from the user will have to go through a filter that is configured in the project. This filter for the input, is used in all the requests ending in a page that will link, post, or call the other functionalities of the project. In our code, we made sure that the browser sends the information entered, and the server will validate it before approving the functionality. In this way, any unauthorized user will not be able to append malicious data (here in this case, add unauthorized users) or change the game results.

6.2 HOW TO SECURE THE DATABASE

Spring MVC framework that is used in this project is integrated with Hibernate, MySQL and Spring Security. In this framework, we have defined the roles for the users and these are all stored in the database. The following shows the list of attributes we have created to store the values in our Blackjack database.

```
mysql> select * from user;
```

id	username	email	emailhash	password
10	schepuri	srinivaschepuri2012@gmail.com	2df334ba-8451-4c89-baf2-6a88a3383fd0	\$2a\$11\$5xVa9Dr/5E83wB13Zmc07.UKLjEN7gX5FQge9Zg81yNF.04PC/Nq
15	padmini89	padmini89@gmail.com	336627be-cac1-49b7-9347-de1d98efb1a6	\$2a\$11\$ivqmu6d7OjD6OmjyZV.TeXozR.uL594Dx7qgW/yolUlnegiwEj10
16	madhurima.doppalapudi	madhurima01@gmail.com	07dd0180-270d-468c-abe0-ed7716081328	\$2a\$11\$Xr9xUP2cN/3XvkghIVacZO0yecOTR.qWNgOWubug05PydVdk80tX.
17	nehaReddy.allam	neha121@gmail.com	b4c24ca7-adba-4d89-bfae-8290b0fc4155	\$2a\$11\$T06xnr618EQmV781km4EieIwhlccMCpAVK60K04YNY.uLSKr7YIG

4 rows in set (0.00 sec)

```
mysql> select * from role;
```

id	name
1	ROLE_USER
2	ROLE_ADMIN

2 rows in set (0.00 sec)

```
mysql> select * from game;
```

id	username	result	rightHandresult	createddate	playerCredits	playerBet	dealerHandValue	splithand	leftHandresult
6	schepuri	You lost because you went bust (over 21). The Dealer has taken your bet.	NULL	2017-05-06	990.0	10	5	false	NULL
7	schepuri	You lost! The Dealer has taken your bet.	NULL	2017-05-06	980.0	10	20	false	NULL
8	schepuri	The Dealer went bust! You won.	NULL	2017-05-06	990.0	10	22	false	NULL
9	schepuri	You lost because you went bust (over 21). The Dealer has taken your bet.	NULL	2017-05-06	980.0	10	10	false	NULL
10	schepuri	You lost because you went bust (over 21). The Dealer has taken your bet.	NULL	2017-05-06	970.0	10	7	false	NULL

Hibernate is facilitated in our project to protect the database from SQL Injection. This is configured in the pom.xml file

```

<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
  <version>${spring-data-jpa.version}</version>
</dependency>

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>${hibernate.version}</version>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>${mysql-connector.version}</version>
</dependency>

<dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>${commons-dbcp.version}</version>
</dependency>

```

6.3 HOW DO WE ENSURE THE GAME IS SECURE

6.3.1 Prevention of XSS

In the web.xml that we have implemented in the project, we have also configured the XSS filter and this filter wraps the HTTP request object in a class called HttpRequestWrapper. This is responsible for performing the content filtering in our project. Apart from this, in the first step of the web application, the user will be given access only to enter two attributes. These two attributes are the username and the password.

If attackers try to enter malicious query parameters in the url, the url is redirected to the same page and the system is unaffected. Also, the validate() method in org.blackjack.validator.UserValidator class performs input validation before using the input entered by the user.

6.3.2 Prevention of CSRF

The use of CSRF tokens which are available as part of Spring Security framework ensure that a token is enabled with login and registration form as a hidden parameter. It is enabled by default with use of WebSecurityConfigurerAdapter class.

6.3.3 SQL Injection

SQL injection attacks occur when attacker tries to enter malicious SQL commands and data as user input. If the malicious data is used during database retrievals/ insertion, an attack can occur on the database.

In our system, we used Hibernate with Spring Security framework and input validation to protect the database system. Hibernate uses object relational mapping which checks for data type entered and does not use queries to access the database system.

Our system also has client side and server side validations.

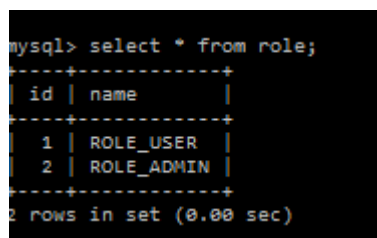
6.3.4 URL redirection

We ensure that the urls which display game statistics and data are available to a third-party user only he/she are logged in. We added a redirection rule which redirects the url of the game to login page when hit by a third party without authentication. This is done in WEB-INF/views/appconfig-security.xml :

```
<http auto-config="true">
  <intercept-url pattern="/" access="hasRole('ROLE_USER')"/>
  <intercept-url pattern="/welcome" access="hasRole('ROLE_USER')"/>
  <intercept-url pattern="/blackJackTable"
access="hasRole('ROLE_USER')"/>
  <form-login login-page="/login" default-target-url="/welcome"
authentication-failure-url="/login?error" username-parameter="username"
password-parameter="password"/>
  <logout logout-success-url="/login?logout" />
</http>
```

6.3.5 Data Security:

The database is secure and can be viewed only with authentication details. User roles are defined with 'Roles' table and only if the user is admin, he can modify the database system manually.



```
mysql> select * from role;
+----+-----+
| id | name  |
+----+-----+
| 1  | ROLE_USER |
| 2  | ROLE_ADMIN |
+----+-----+
2 rows in set (0.00 sec)
```

6.3.6 Session Timeout:

When the user logs in and joins the game, the timer would begin when he enters the username and password. If the user is authenticated, then the application would begin and the dealer would have the panel updated with cards. If within 5 minutes, the player does not make any move, or if he fails to decide what move he makes, then the game would be automatically terminate and the player would be declared as burst and the results will be updated in the database.

6.3.7 Fault Tolerance:

In the web application that we have developed, if at any time in the game, if the browser shuts down or crashes, the time would then be considered. If the user or player cannot restart and login into his game within the time limit of the session timeout, then the game would be terminated and the player would be considered as burst. The results would be updated on the database. If the player resumes within the period then he can resume the game from where it was halted.

6.3.8 Ensure Single Game Participation:

The web application that we have developed is compatible for one player to play, only one given game at a time. He will not be able to login from the same username and password and be able to play the game from a different browser. This makes sure that the user cannot play against himself and is part of only one game at a time.

6.4 HOW DID WE TEST OUR APPLICATION

For ensuring the database we used is secure we have used an in-memory database with the tool Hibernate. We have used this directly instead of configuring a complete clone of our production database and masking the data.

Before we integrated the game functionality with the data base and the front-end web application, we run the entire java code and performed unit testing. We ensured that every method runs successfully to perform its functionality.

After integration, we run the whole code on Eclipse using Junit testing to make sure all the requirements of the game are met.

References

[Tartankane2012] <https://github.com/tartankane/BlackJack>

[AppShah2016] <https://crunchify.com/step-by-step-guide-to-enable-https-or-ssl-correct-way-on-apache-tomcat-server-port-8443/>

[Mkyong2010] <https://www.mkyong.com/spring/maven-spring-hibernate-mysql-example/>

[HelloKoding2016] <https://hellokoding.com/registration-and-login-example-with-spring-xml-configuration-maven-jsp-and-mysql/>

[mitil2016] <http://www.studytrails.com/frameworks/spring/spring-security-custom-login-page/>