WHO WANTS TO BE A MILLIONAIRE

PROJECT REPORT

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

SECURE SOFTWARE DESIGN AND PROGRAMMING

IN

COMPUTER SCIENCE

BY

AKSHAY SONAWANE (G00962779)

SARTHAK SHETH (G00979607)

DEPARTMENT OF COMPUTER SCIENCE

GEORGE MASON UNIVERSITY

FAIRFAX, VIRGINIA

INTRODUCTION

The game we developed is “Who wants to a millionaire”. It is a simple multiplayer game in which each player answers some set of questions and wins money with every correct answer and losses some with a wrong answer. The player with the highest money in the end is the winner. In this game, we made some modification. Instead of having fixed amount for each question, the player can select one amount from the range 1 to 100 as the betting value for that particular question. For example, if a player set a bet of $75 for a question and if he wins then $75 will add to his total else he will lose $75 from total amount he won before. Player cannot bet $0 or more than $100 for each question. Initial amount with each player is $0. In our game, Initially, a player has to do registration. Once his username and password is authenticated, he can login and start the game. He can select any quiz of his interest and start with it. Each quiz is of 10 questions hence maximum money can be won is $1000 if every answer is right and the player bets max that is 100 for every question. This game is tricky and must be played smartly as betting is very important factor here. Knowing more correct answers is useless if you do not bet well. For example, if player A gives 6 correct answers with bet of 100 each and 4 wrong with bet of 20 each then his total will be $520 and player B gives all 10 correct answers with bet of 50 for each. Then total of B will be $500 and which is less than total of A. Hence here B gives more correct answers but A wins more money. In this game, every player can see and compare high scores of other players after his login. One with the highest points wins the game for that quiz. We have also added a admin login using which admin can create new quiz. Using authorized admin using, he can create new quiz.

ARCHITECTURE

The game has been developed using a client-server architecture. We have used JSP as the primary server-side programming language, and have used Oracle 10g as the database system. The server which we have used for the deployment of our application is Apache Tomcat v8.0.

Java Server Page (JSP) is a technology which is used for controlling the content and appearance of web pages through the use of servlets- small programs that are specified in the web page and run on the web server to modify the web page before it is sent to the user who requested it.

JSP helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>. Using JSP, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc. Our entire application runs through a series of servlets on the web browser, which are nothing but small java programs that interactively communicate with the web server.

DESIGN

The major components of our project build are:

- Eclipse: It is an integrated development environment (IDE) used for computer programming. Different java and jsp files are written under this IDE.

- JSP: used as the server side scripting language to store and retrieve information from a database, and create webpages dynamically.
- Servlets: They are small java programs that act as intermediaries between the client and the server. As servlet modules run on the server, they can receive and respond to requests made by the client. Request and response objects of the servlet offer a convenient way to handle HTTP requests and send text data back to the client.
- Oracle 10g: We used Oracle 10g database to store the data.
- Apache-Tomcat 8: Used as a server to create a localhost.

IMPLEMENTATION

The major components of the game applications are:

- Registration/Login
- Display Statistics
- Start new quiz (for player)
- Make new quiz (for admin)
- Logout

Registration/Login:

Users who want to play the game can register themselves by creating an account. If they have registered themselves earlier, they can log into their account through the login page. All the validations are in place with the help of regular expressions. A user can register only once and is not allowed if he tries to register again. A regular expression validates the user inputted password to be strong enough for checking minimum of one small and capital letter and one number. Passwords are encrypted and stored as salted hashes in the database. A secure connection between client and server is established by implementing SSL/TLS.

Files that have been used for this implementation are:

- Index.jsp
- Register.jsp
- Registerprocess.jsp
- Loginprocess.jsp
- Home.jsp

Display Statistics:

By using view stats, players and admin can see the scores of players after taking the quiz. In that table, player's user name, quiz he played, number of correct answers and total amount won are four rows. Hence here all the players can see the scores of others and winner can be decided by that statistics.

Files that have been used for this implementation are:

- Home.jsp
- Savestats.jsp
- Result.jsp
- View.jsp

Start new quiz:

Here players can start their new quiz after login in. By entering their interest, they can see the quiz according to that interest. When player select that quiz, the game will start and he has to answer all 10 answers.

Files that have been used for this implementation are:

- Takequiz.jsp
- Findname.jsp
- Start.jsp
- Get.jsp

Game:

When the player starts new game, there is a question and its four options are given on screen in which only one is correct. Player has to guess the correct one and along with that he has to bet for that question. Betting possible is in between 1 to 100 for each question. After selecting the answer and his bate, player will click next and next question will appear. This process will run 10 times and after final question, player will see his total amount.

Files that have been used for this implementation are:

- get.jsp
- get1.jsp
- findname2.jsp
- findname3.jsp
- result.jsp

Make new quiz:

this functionality is given only to admin and not to players. Admin can login using admin login Id and password, and then he can create a new quiz. First admin will select a subject for that quiz and then he will add 10 questions and 4 options for each question and one of which is correct.

Files that have been used for this implementation are:

- askq.jsp
- createquiz.jsp
- createquiz1.jsp
- makequiz.jsp

Exit Game:

This module makes sure that the database is properly updated on the exit function. It checks the status of the game when a player hits the exit button. If a player has exited from a current game erroneously, he will be having a chance of joining the game back if it is still in progress by the time he logs in. If one player exited the game by clicking logout, then that player will get 0 score, and the database will be updated accordingly.

INSTALLATION INSTRUCTIONS:

1. First create an account in oracle 10g with username “system” and password “oracle” or we can change the same in the web application while configuring the JDBC connectivity.
2. Import war file “OnlineQz1.war” on eclipse IDE.

3. Create a new apache-tomcat V8 server and configure it to eclipse and host the project “OnlineQz1”.
4. Start Oracle10g and create tables mentioned in the project.
5. Start tomcat server.
7. Open any web browser of your choice and enter the URL **https://localhost:8443/OnlineQz1/** to access the index.jsp page for the game. The above URL is a SSL/TLS secured connection.

GAME RULES:

1. Player can start a new quiz after logging in.
2. Each quiz is of 10 questions with 4 options having 1 correct answer.
3. Player has to guess the correct answer from 4 options and with that there is one betting text box where he can enter a betting value for each question from 1 to 100.
4. Initial betting value is 0. For every correct answer, player will win the betting amount he set for that question and for every wrong answer, he will lose that much of money.
5. At the end of the quiz, he will know his total score that is number of questions he answered and how much money he wins. He will also see the correct answers of every question after finishing the quiz.
6. Every player can see the scores of other and then the winner can be decided from that statistics.

SYSTEM REQUIREMENTS:

Operating System: Windows XP/VISTA/2007/8/10. Web browser: Internet Explorer, Edge, Firefox and Google Chrome.

Software: JRE 7 and above, Oracle10g and Tomcat Server v8.0.

WHY WE BELIEVE OUR GAME IS SECURE:

While developing our game, we were aware of the many security loopholes that may arise during the course of development. Hence, we were extremely vigilant and wary of our coding practices, and made sure to check for security at multiple instances during the entire development process. The crucial countermeasures that we have deployed to make our game as secure as possible are as follows.

SSL/TLS:

SSL/TLS has been implemented and configured on the server. The following piece of code has been added to the web.xml file of application which will enable the encrypted connection between client and server.

Below piece of code is added in server.xml present in tomcat installation directory

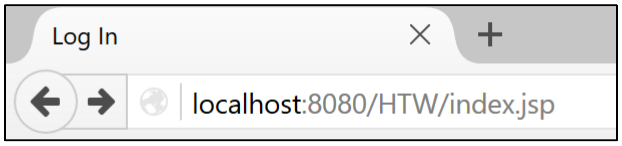
```
<Connector port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true" scheme="https"
secure="true"
```

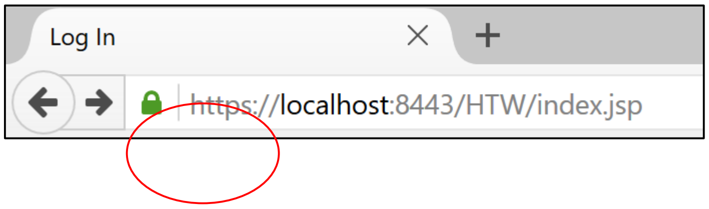
```
                    clientAuth="false" sslProtocol="TLS"
keystoreFile="N:/MyCert1.cert" keystorePass="vai681" />
```

Here, the above path is the path to the keystore file, which is a self-signed certificate generated using below command:

```
Keytool-genkey-alias <<name>> -keyalg RSA -keystore <<name>>
```

The self-signed certificate generated using the above command will allow us to have an encrypted channel for sending the user sensitive data





From the above two screenshots, we can assert that even if we type the index.jsp as HTTP, it redirects us to HTTPS which is an encrypted channel, thereby ensuring SSL/TLS connection is active.

Input Validation:

Regular expressions have been used at appropriate places for validating the input. The user’s given input is checked against a whitelist of allowed entries. This has been applied at Login and Registration, which makes sure that the user does not create bogus entries to get unvalidated access.

By employing input validation, we have been able to secure the application against cross site scripting, SQL injection and other such attacks. Various validation rules have been set for various fields of the input.

For login field during registration or login:

```
<tr>
<td>User Name:</td>
      <td><input type="text" required name="username” onkeyup="sendInfo()"
      title="Must start with a Capital Letter and Only Letters are allowed"
            required pattern="[A-Z][a-zA-Z]*" /></td>
            </tr>
            <tr>
<td><br /></td>
            </tr><tr>


<td>Password:</td>

td><input type="password" required
pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
```

```
      title="Must contain at least one number and one uppercase and lowercase
letter, and at least 8 or more characters"
      name="userpass" /></td>
```

Fault Tolerance:

During the game, if a user’s browser crashes, he can log back in again and continue his game as long as he is able to come back within the five minutes time frame, which is the session timeout limit.

SQL Injection:

Our game is also secured against SQL injection attacks. Prepare statements have been used whenever there is any form of interaction with the database. Using prepared statements, we can force the user input to be handled as the content of a parameter (and not as a part of the SQL command).

<u>Sample Query:</u>

```
st=con.prepareStatement("SELECT * from winstats where userid=
?");

st.setString(1,username);

rs = st.executeQuery();

rs.next();

out.println("<h1> Win/Loss Statistics</h1>");

out.println("<br></br>");
```

Salted Hashes for Passwords:

Using input validation, the password given by the users during registration are first validated against a whitelist. Using these passwords, random salts are generated, which are unique to each and every password. In Java, Secure Random is used to generate secure salts. This class supports the SHA1PRNG pseudo random generator algorithm. A random salt is generated and is passed on to the PBKDF2WithHmacSHA1 algorithm which is a cryptographic hash function and then the hashed password is stored in the database.

<u>Sample Code:</u>

```
final int ITERATIONS = 1000;
final int KEY_LENGTH = 192; // bits
char[] passwordChars = pwd.toCharArray();
byte[] saltBytes = rs.getString(3).getBytes();

PBEKeySpec spec = new PBEKeySpec(
passwordChars, saltBytes, ITERATIONS, KEY_LENGTH
);
SecretKeyFactory key;

key = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
byte[] hashedPassword = key.generateSecret(spec).getEncoded();
String       password       =       String.format("%x",       new
BigInteger(hashedPassword));
```

Ensure Single Game Participation:

The business logic has been implemented in such a way that the user can be a part of only one game a time. When a user creates a game and logs in from another browser, he will not be able to log in to the game system itself. This makes sure that the user cannot play against himself and is part of only one game at a time.

Buffer Overflow:

Buffer overflow has been prevented in this application by checking minimum and maximum limits for every input field.

```
Your name   <center>  <input name="name" type="text" id="name" 
maxlength="15" /><br><br></center>
      Email ID     <center> <input name="emailid" type="text" 
id="emailid" maxlength="25"/><br><br></center>
```

The attribute in html tag **maxlength** will not allow user to type more than 25 characters.

```
<input name="emailid" type="text" maxlength="25" id="emailid" 
value="" />
if (emailid == null || emailid == "") { alert("Emailid must be 
filled out"); return false;
```

The javascript shown above will check for no character input.

Server Side Session Configuration:

The session timeout has been set to 30 minutes on the server side with the following piece of code being placed in web.xml

```
<session•config>

<session•timeout>30</session•timeout>

</session•config>
```

Usage of Data Access Object:

Throughout our game we have made use of Data Access Objects to separate the Data Related part from the coding part. A data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, DAO provide some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data accesses the application needs.

The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently. Changing business logic can rely on the same DAO interface, while changes to persistence logic do not affect DAO clients as long as the interface remains correctly implemented. All details of storage are hidden from the rest of the application. Thus, possible changes to the persistence mechanism can be implemented by just modifying one DAO implementation while the rest of the application isn't affected. DAOs act as an intermediary between the application and the database. They move data back and forth between objects and database records.

Most of the critical database related operations in our game like Registration, Login have been implemented using DAO and none of the related code is present in the JSP files. This makes this data invisible to the users and adds some security.

CONCLUSION:

Finally, we would like to conclude that after getting this amazing opportunity to develop our very own gaming application, we have learnt that no software or web application is perfect or unbreakable. But as software developers, making our projects as secure as possible such that it becomes extremely difficult for malicious attackers to break in should be of paramount importance to us.