GO GAME

Secure Software Design and Programming

ISA/SWE 681

By

Ankita Acharya (G01165373)

Avinash Arunachalam Arunachalachettiar Murugappan (G01163980)

Jothi Arul Prakash Ponnusami (G01159998)

(Under the guidance of David A. Wheeler)

May 10, 2019

GEORGE MASON UNIVERSITY

Table of Contents

1. INTRODUCTION :

Go is a two-player abstract strategy game, in which the objective is to surround more territory than the opponent. The playing pieces are called “stones”. One player uses the white stone and the other uses black. The players take turns placing the stones on the vacant intersections (“points”) of a board. Once the stones placed on the board, they may not be moved, but stones are removed from the board if "captured". Capture happens when a stone or group of stones is surrounded by opposing stones on all orthogonally adjacent points. The game proceeds until neither player wishes to make another move or say, there is no profitable move left for the current player to play. When a game concludes, the winner is determined by counting each player’s surrounded territory along with captured stones. The standard GO board has 19*19 grid of lines, containing 361 points. This interesting game had been developed with the security requirements that were taught during the course.

1.1. CLIENT-SERVER ARCHITECTURE:

A Client-Server Architecture is a distributed application structure that splits the tasks separately. The Architecture contains server, client and database modules. In a network, one or many clients request service to the server and the server receives and responds to the request from the clients.

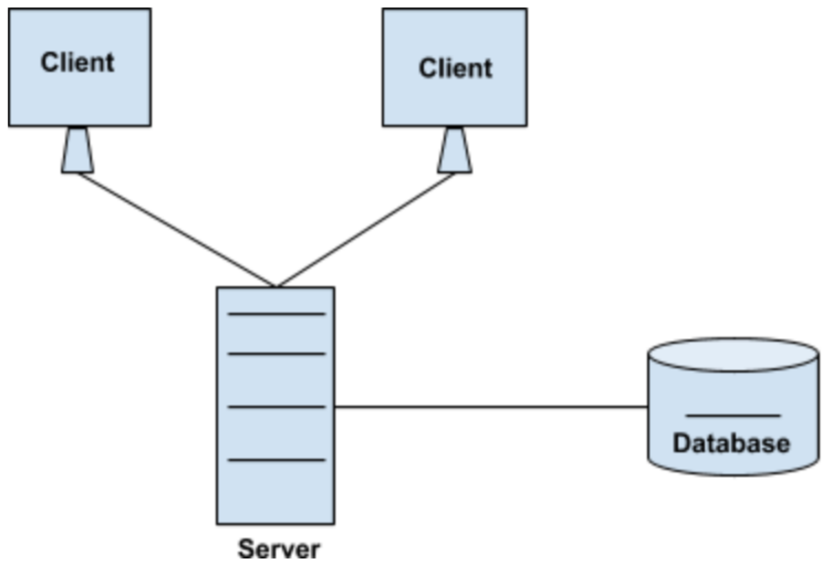


Figure 1: Client-Server Architecture

2. PROJECT ARCHITECTURE:

The game is designed for the main purpose of the project using Client-Server Architecture. Project Architecture is represented below:

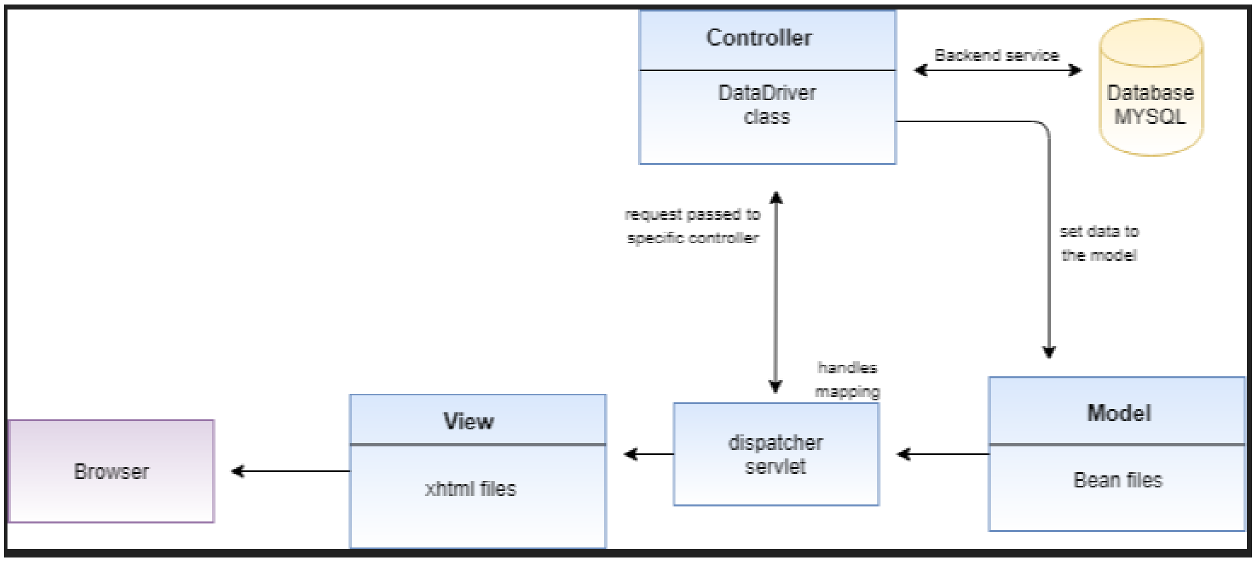


Figure 2: Project Architecture

2.1. JSF MVC FRAMEWORK:

JSF is very convenient as it is based on the Model View Controller architecture, where all the components are readily available to develop a web application at developer's dispose. The model encapsulates the data, View is responsible for the UI logic of the application and the Controller is responsible for processing the requests from the users by controlling the model data and the view. In our project, the architecture we used is modeled using JSF MVC Framework so that different aspects of the application were kept separately. Our game is developed in a Java-based application.

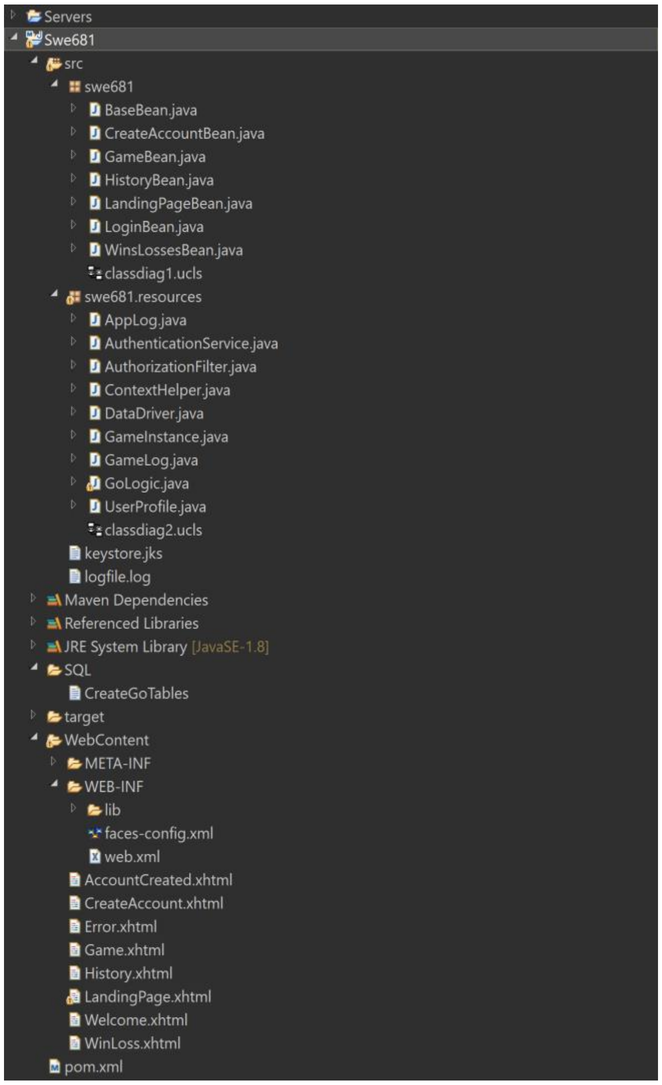
The project made use of the MYSQL database management system to store and retrieve the data that was used and forms the bottom layer in our architecture. Once we figured out the flow between the database object and the client, we utilized the Service layer to operate on the data. In this way, we were able to follow the Dependency Injection mechanism. The next step was to execute the functionality specific to the request that was made by the client which was accomplished by the AppController component. The server-side validations were also performed in this layer. The presentation layer that we created using CSS and Javascript, provides a rich graphical user interface.

Since our focus was more on the security aspect, we used the Spring security framework for the best.

2.2 SPRING SECURITY FRAMEWORK:

This framework provided us the support for authentication and authorization security features. Authentication is identifying the individual who needs access to the entity. And Authorization is the process to allow access to that individual. One of the main advantages of using this framework was that it provided protection against Cross Site Scripting attacks. This security framework was integrated with the JSF MVC framework in our project.

3. PROJECT STRUCTURE:



```
Servers
Swe681
  src
    swe681
      BaseBean.java
      CreateAccountBean.java
      GameBean.java
      HistoryBean.java
      LandingPageBean.java
      LoginBean.java
      WinsLossesBean.java
      classdiag1.ucls
    swe681.resources
      AppLog.java
      AuthenticationService.java
      AuthorizationFilter.java
      ContextHelper.java
      DataDriver.java
      GameInstance.java
      GameLog.java
      GoLogic.java
      UserProfile.java
      classdiag2.ucls
    keystore.jks
    logfile.log
  Maven Dependencies
  Referenced Libraries
  JRE System Library [JavaSE-1.8]
  SQL
    CreateGoTables
  target
  WebContent
    META-INF
    WEB-INF
      lib
      faces-config.xml
      web.xml
    AccountCreated.xhtml
    CreateAccount.xhtml
    Error.xhtml
    Game.xhtml
    History.xhtml
    LandingPage.xhtml
    Welcome.xhtml
    WinLoss.xhtml
  pom.xml
```

Figure 3: Structure of Project

Type about each java file:

CreateAccountBean: This component handles all the data from the user, like username, password, etc., required to create her account.

GameBean: This component handles the data like move, capture piece, win, loss, etc., required to run the game.

LandingPageBean: This component handles sessions at the start of the new game and when the player joins the game. It also manages the log out.

LoginBean: This component has the regular expression to check for the allowed characters in passwords and login name.

AuthenticationService.java: This component verifies the credentials of the users. It checks on the number of attempts a user can make, checks if the player has not left any of these mandatory things blank or empty. If credentials are wrong for a certain number of attempts, it locks out the user.

AutherizationFilter.java: This component makes a request to the servlet to handle views and related classes and creates sessions.

DataDriver.java: This component helps to access the database and perform the CRUD (create, read, update, delete) operations.

GameInstance.java: Sets and gets every detail of the player like the number of passes, scores, etc.

ContextHelper.java: Gets the current player owning the session. Also, terminates it when the player logs out.

GoLogic.java: This component handles the main interactivity of the game like capturing pieces, setting colors of the board and changing the gameState array which contains the color of the board which player has placed. It handles the logic while the player is capturing the piece. The invalid capture is prompted as an error to the user.

GameLog.java: Maintains the log of which player is playing which move.

Userprofile.java: A POJO class which contains all the statistics of the player like win and loss, username, hashed password, etc.

XHTML Pages: These add up the view into the bean classes. These are the view components of the MVC.

Web.xml: A deployment descriptor file for servlet-based java web application declaring which URL's the servlet should handle.

Pom.xml: It is the fundamental unit of work in Maven that contains information about the project and configuration details used by Maven to build the project.

4. PROJECT DESIGN:

Programming Language: JAVA

Database: MYSQL Workbench 8.0

Maven: Build automation tool for our application.

Server: Apache Tomcat 9.0.

5. INSTALLATION INSTRUCTIONS:

5.1 SOFTWARES REQUIRED:

Application: Eclipse Java EE IDE for Web Developers

Server: Apache Tomcat version 9.0

Database: MySQL WorkBench 8.0

5.2 STEPS FOR INSTALLATION:

1. Install Apache Tomcat 9 and MySQL WorkBench.
2. Database Preparation:
    a. Run the GoTables.sql script. The Script holds the entire schema of the database with 4 tables in it.
    b. Create a database called “swe681” with access rights like “SELECT, INSERT, UPDATE, DELETE, DROP” -etc

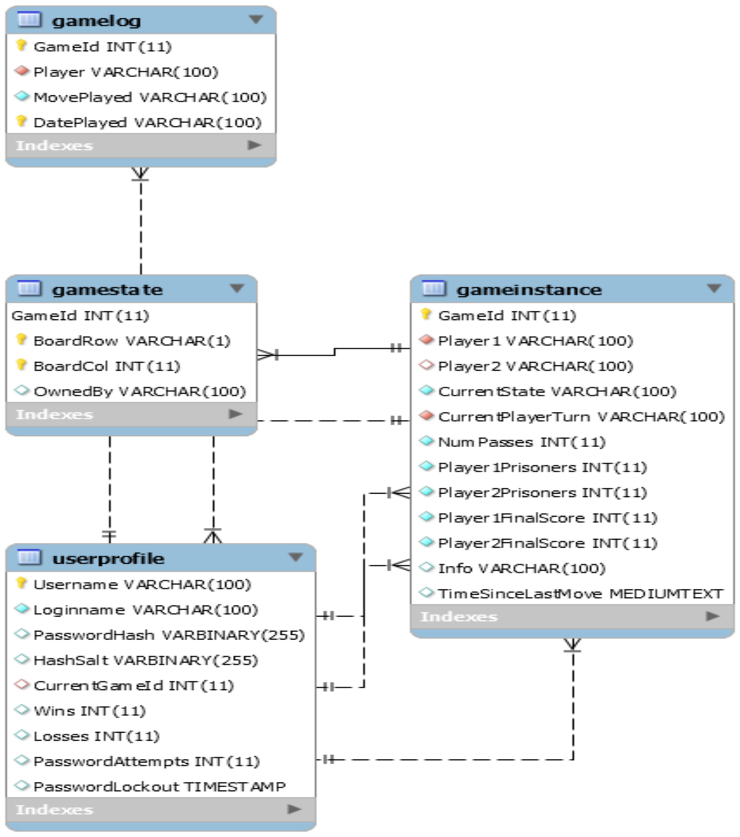
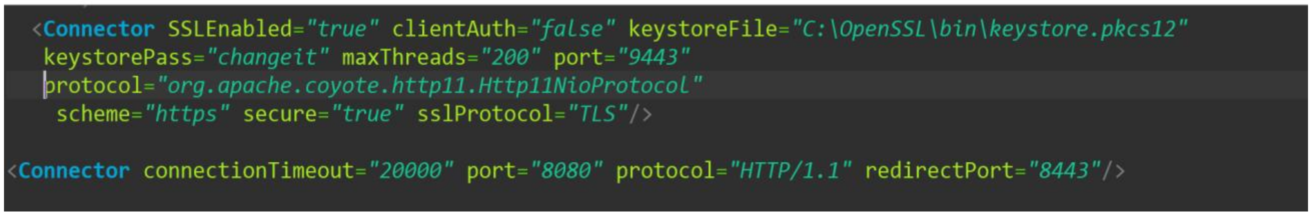


Figure 4: Database Schema

Note: Use the default username/password as root/root for the project created in the MySQL WorkBench.

1. Tomcat Installation:

a. Include the https port definition inside the server.xml as below:



```
  <Connector SSLEnabled="true" clientAuth="false" keystoreFile="C:\OpenSSL\bin\keystore.pkcs12"
   keystorePass="changeit" maxThreads="200" port="9443"
  protocol="org.apache.coyote.http11.Http11NioProtocol"
    scheme="https" secure="true" sslProtocol="TLS"/>

<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>
```

Figure 5: HTTPS port definition

b. Set the path of the *keystore* in the server.xml (under keystorePath).

c. After all set, start tomcat the server (Tomcat>bin>startup).

Note: All the required jar files and corresponding dependencies are managed by the Maven.

6. OPERATING INSTRUCTION:

1) By accessing the URL, a login page will pop up enabled with TLS secure connection.
2) Login page authorize the player by confirming the username and password.
3) In order to check the user's authentication, the user must create an account using "Create Account" where login name, username, password are created. Without logging into the game, the information about the game or players statistics will not be revealed.
4) Once the login is successful, the homepage is popped, and the player has options to start a new game and view the game statistics.
5) After the Start new game is clicked, the player is to wait for the opponent to join the game. Once the second player joins, the game starts.
6) Another player, who is not already in the game, is prohibited to enter a game which is still going on.
7) Inside the game,players are allowed to make moves by entering the positions in the box.
8) After the game ends, the players are directed to the homepage where they can see the updated statistics.
9) Every move made by each player can be viewed by clicking the history.
10) By clicking logout button, the player can exit the game.

7. GAME RULES:

The rules specified here are the basic rules that are common over different types of go game played all over the world. These rules provide a convenient basis for beginners.

7.1. BASIC BOARD RULES:

- Go game is played between two players who are given colors -Black and White.
- Played on a grid of 19 x 19, 13 x 13, 9 x 9 sizes which we call as "board".
- Game is played with tokens which are called "stones". Each player has sufficient color stones for their turns, that means, they are never going to run out of these tokens.
- Black takes advantage of the first move every time.

- **Positions:** Position indicates the state of every intersection. Each intersection on the board will be under one of the following states:
    - **Empty State:** The position is unoccupied.
    - **Black Occupied State:** The position has been occupied by the black stone.
    - **White Occupied State:** The position has been occupied by the white stone.
- A vacant point which is adjacent to a stone is called liberty for that particular stone. Liberty for a stone will be lost when a chain is surrounded by opposing stones. Hence, that position is captured and the opposite stone is removed from the board.
- **Ko Rule:** The Stones on the board should not repeat to capture the previous position of stones. It means that the players are not allowed to attempt a move that reverses game to the previous state [WikiGoRules2019].

7.2. PLAYING RULES:

- A Player should place a stone only on an empty intersection.
- A Player can remove any of the stones of the opponent from the board that have no liberties.
- An Occupied Position cannot be taken by another player. A play is considered as illegal that would reflect the game by retaining it to the position that occurred previously in the game.
- A Player score is calculated based on the number of stones that the player has on the game board along with the number of empty positions that are surrounded by the stones of the player.

7.3. END OF THE GAME:

When neither of the players has any meaningful or profitable move to play, then they can conclude the game. For example, if there are empty sections in the board where supposedly white has to play and black can counter every move which white can play, then the game is not profitable. If the move is not in the benefit of the current player playing it and there are no such useful move, the game ends there.

8. ASSURANCE CASES:

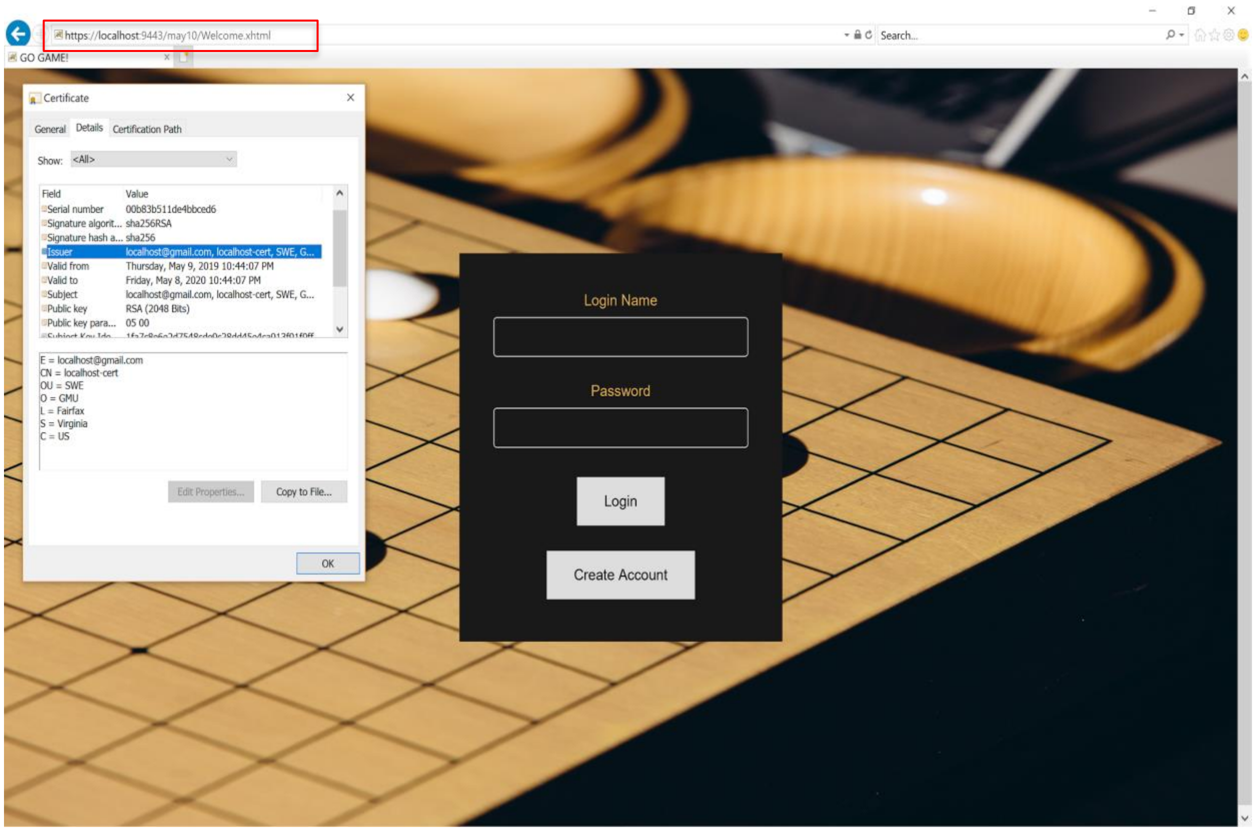
The game is developed with the intention to run the game in any web server and to make the application secure for real-world purposes. Here, we will describe the claims to show how secure is our application and corresponding evidence to assist the claim.

8.1 CLAIM - SECURE ENCRYPTED CHANNEL:

The Game Application is enabled with the Transport Layer Security (TLS), thus the entire information which is transmitted is under protection. The TLS protocol aims primarily to provide privacy and data integrity between two communicating computer applications [Dierks2008]. End-To-End secure communication is enabled by using the cryptographic protocol. TLS is more efficient than SSL and it also improves security.

8.1.1 EVIDENCE

Tomcat Server is configured with TLS protocol with the help of the self-signed certificate. This minimizes the man-in-middle attacks CWE 300 and also limits replay attack by any attack. er CWE 294. The certificate uses RSA public key of 2048 bits. Hence, the connection between the client and server is made using this method.


https://localhost:9443/may10/Welcome.xhtml

Field	Value
Serial number	00b83b511de4bbced6
Signature algorit...	sha256RSA
Signature hash a...	sha256
Issuer	localhost@gmail.com, localhost-cert, SWE, G...
Valid from	Thursday, May 9, 2019 10:44:07 PM
Valid to	Friday, May 8, 2020 10:44:07 PM
Subject	localhost@gmail.com, localhost-cert, SWE, G...
Public key	RSA (2048 Bits)
Public key para...	05 00

```
E = localhost@gmail.com
CN = localhost-cert
OU = SWE
O = GMU
L = Fairfax
S = Virginia
C = US
```

Figure 6: SSL implementation in browser

8.2 CLAIM - AUTHENTICATION

In order to get into the game, the player needs to be authenticated. This can be done by creating a user account by entering the necessary details about the player in a separate page. We can assure that players who are already registered can only request access to enter into the game.

8.2.1 EVIDENCE

Here, the account creation is designed with options for entering Username which is used in the game, Login name which is used for login purpose only, and Password to safeguard the information of the player and his game. Moreover, the spring security framework is implemented in the application to ensure this.

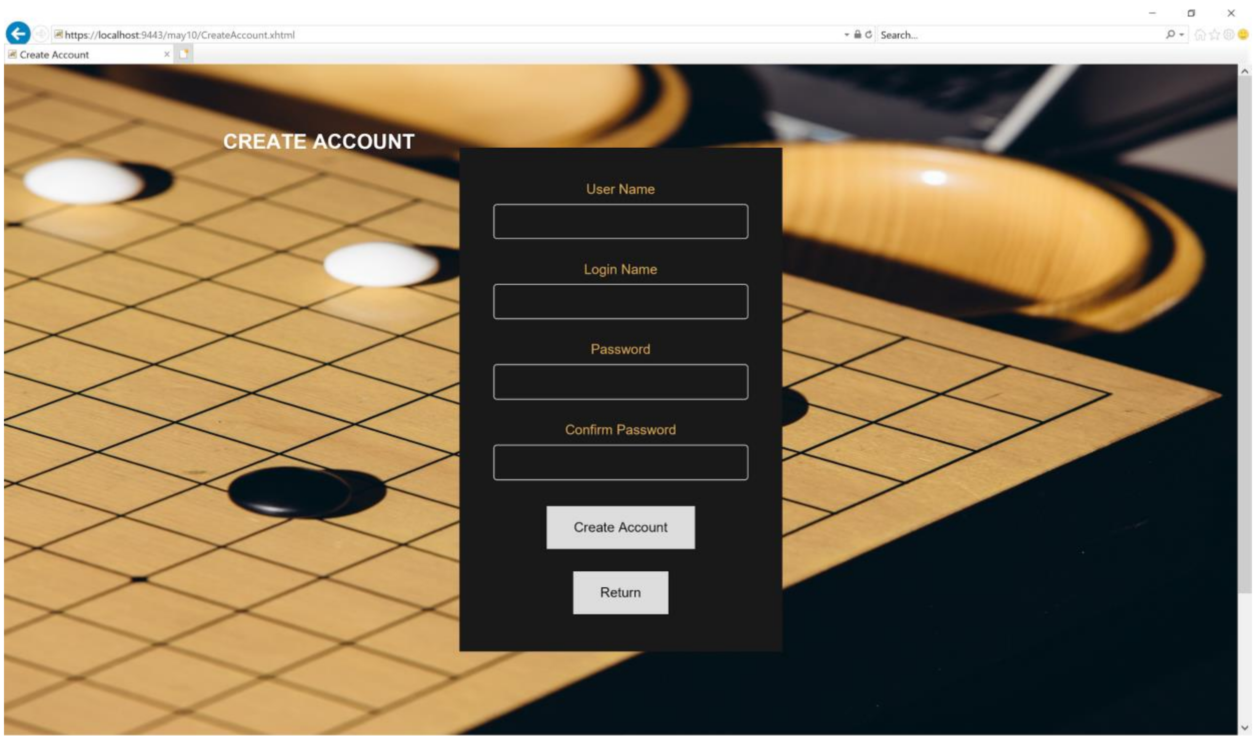


Figure 7: Account Creation Page

8.3 CLAIM - AUTHORIZATION

After the player is authenticated to utilize the game application, authorization of the player is implemented. The Authorization is based on the credentials matches with the details which are entered by the player during the account creation. If so, access is granted to play the game. Thus, the authentication and authorization ensure protection against untrusted users.

8.3.1 EVIDENCE

Here, we check the username and password entered with the information entered in the create account page, then the player is authorized to play the game. The risk of information exposed is reduced in our game. If anyone one of the credentials is wrong, an error message will be popped and the user is authorized. Only correct credentials are accepted. Hence, sensational information is protected in this secure way.

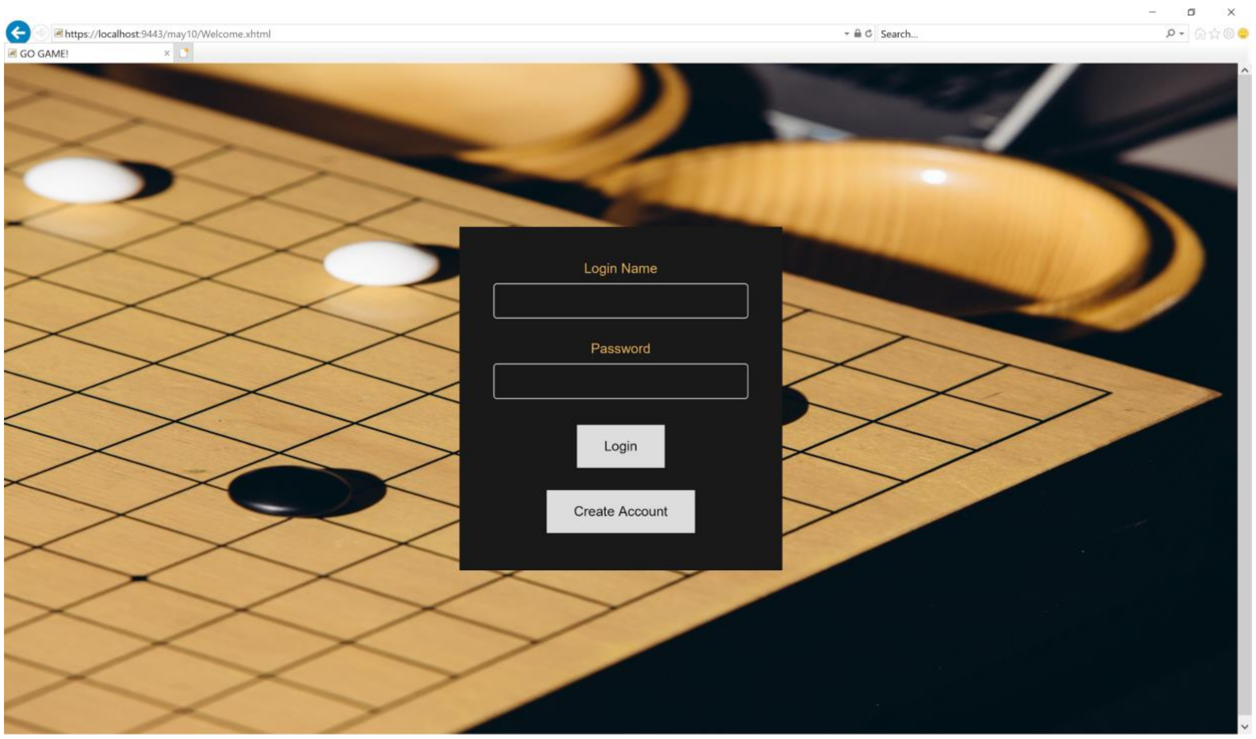


Figure 8: Login Page

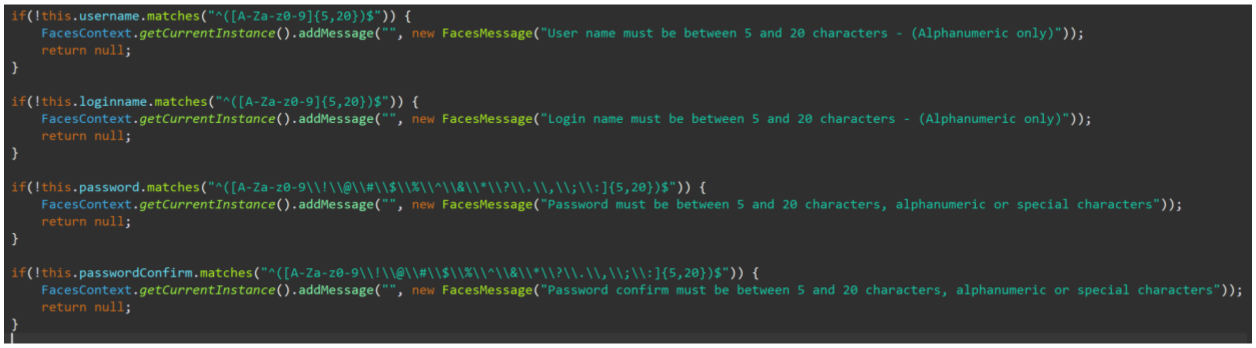
8.4 CLAIM - INPUT VALIDATION

The claim is to show that our application is secure against vulnerabilities like ***Cross-Site Scripting and SQL Injection.*** In some cases, while using regular expressions (regexes) there is a risk of enabling regular expression denial of service (reDOS) attacks [Wheeler2015]. Since the application requests only for username and password for authentication of the player, these vulnerabilities can exploit the trust that the application had towards the players. There are certain possibilities that a malicious attacker can add some random code in the form of SQL statement as an input with the intention to run that statement in the game database. This can be prevented by input validation.

8.4.1 EVIDENCE

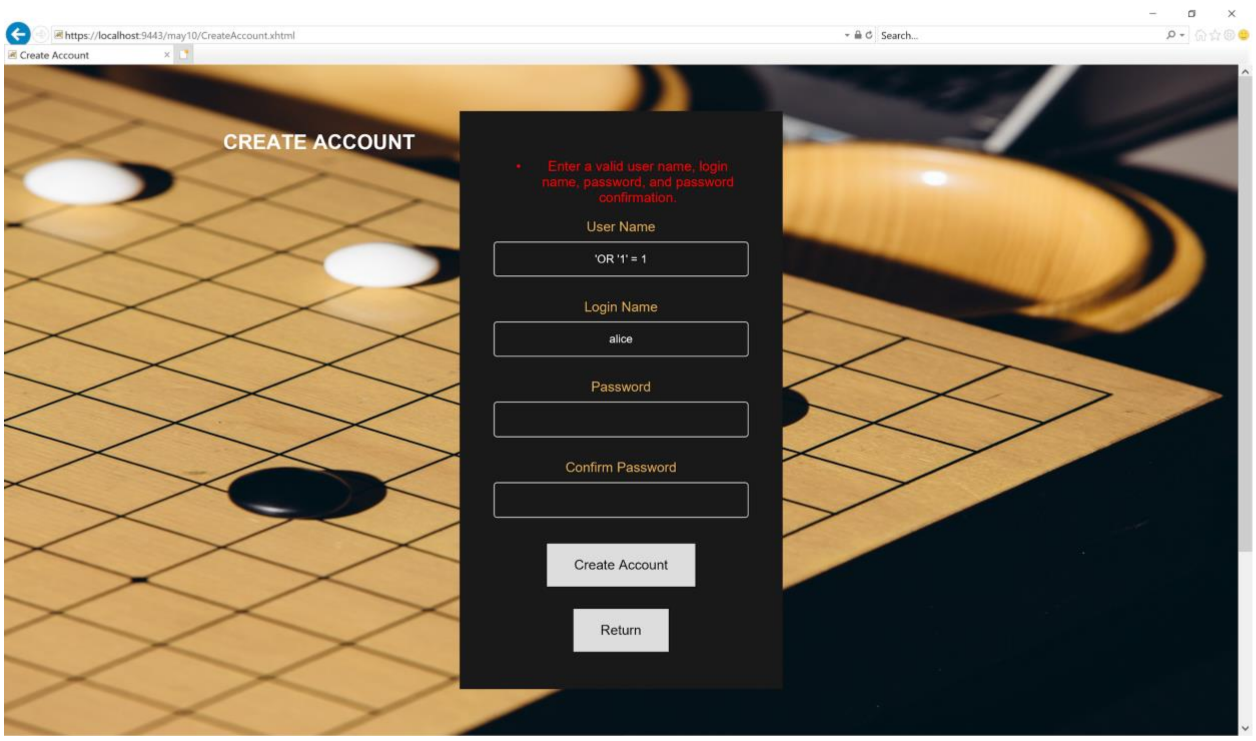
Here, the input entered by the user is validated in the Client side so that the unauthorized user cannot enter into the game and collect information about the game and other player's statistics. Regular Expressions are used to whitelist any scripting tag available in the input. Meanwhile, this feature is inserted in both Login and Create Account page. Even the username and password format are validated based on the input entered.

Moreover, once the input is found as an invalid one, an error message is shown regarding the fault made.



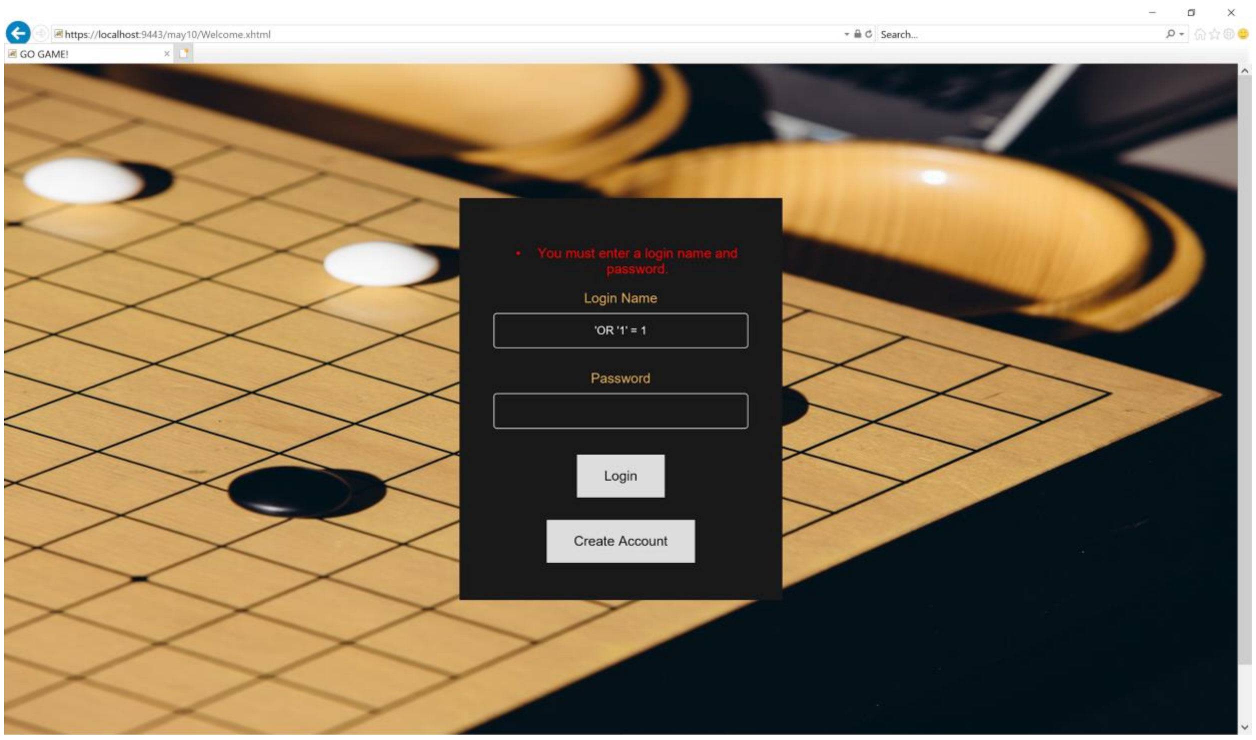
```
if(!this.username.matches("^([A-Za-z0-9]{5,20})$")) {
    FacesContext.getCurrentInstance().addMessage("", new FacesMessage("User name must be between 5 and 20 characters - (Alphanumeric only)"));
    return null;
}

if(!this.loginname.matches("^([A-Za-z0-9]{5,20})$")) {
    FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Login name must be between 5 and 20 characters - (Alphanumeric only)"));
    return null;
}

if(!this.password.matches("^([A-Za-z0-9\\!\\@\\#\\$\\%\\^\\&\\*\\?\\.\\,\\;\\:]{5,20})$")) {
    FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Password must be between 5 and 20 characters, alphanumeric or special characters"));
    return null;
}

if(!this.passwordConfirm.matches("^([A-Za-z0-9\\!\\@\\#\\$\\%\\^\\&\\*\\?\\.\\,\\;\\:]{5,20})$")) {
    FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Password confirm must be between 5 and 20 characters, alphanumeric or special characters"));
    return null;
}
```

Figure 9: Input Validation using Regex



https://localhost:9443/may10/CreateAccount.xhtml
Create Account
CREATE ACCOUNT
Enter a valid user name, login name, password, and password confirmation.
User Name
'OR '1' = 1
Login Name
alice
Password
Confirm Password
Create Account
Return

Figure 10: Input Validation of Regex in Create Account Page



https://localhost:9443/may10/Welcome.xhtml
GO GAME!
You must enter a login name and password.
Login Name
'OR '1' = 1
Password
Login
Create Account

Figure 11: SQL Injection Prevention in Login page

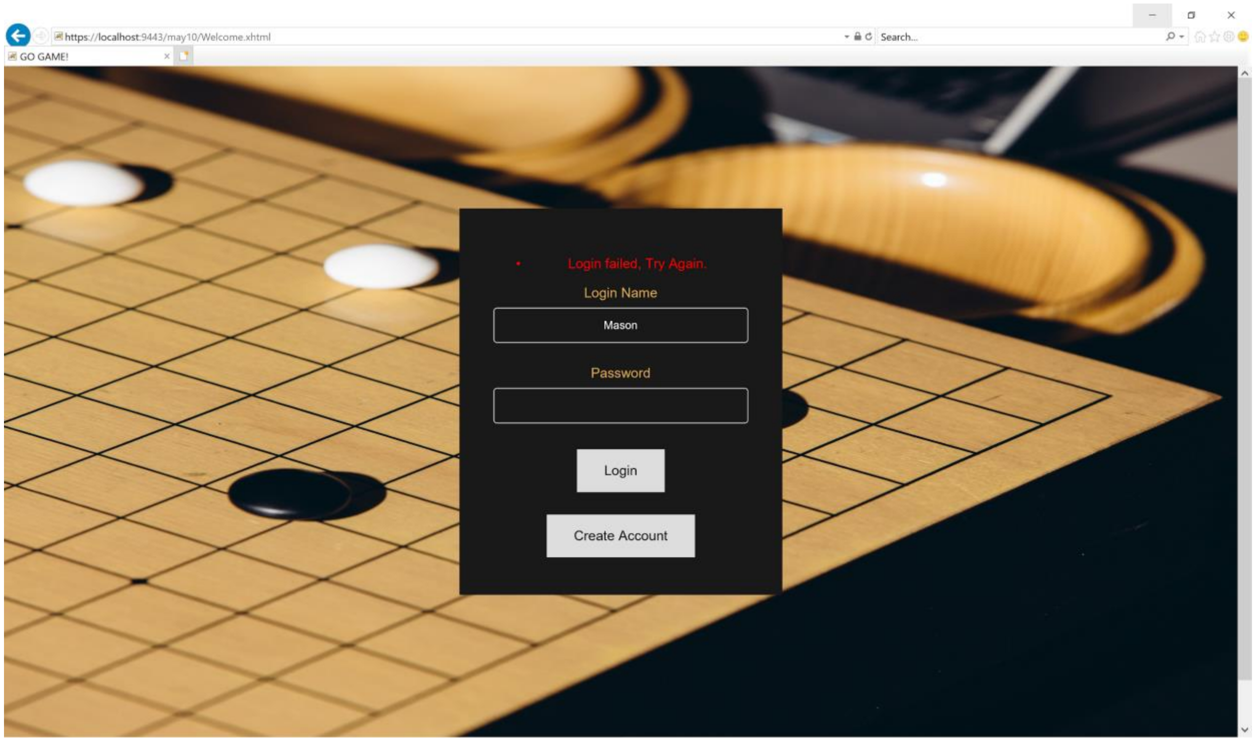


Figure 12: Input Validation in Login Page

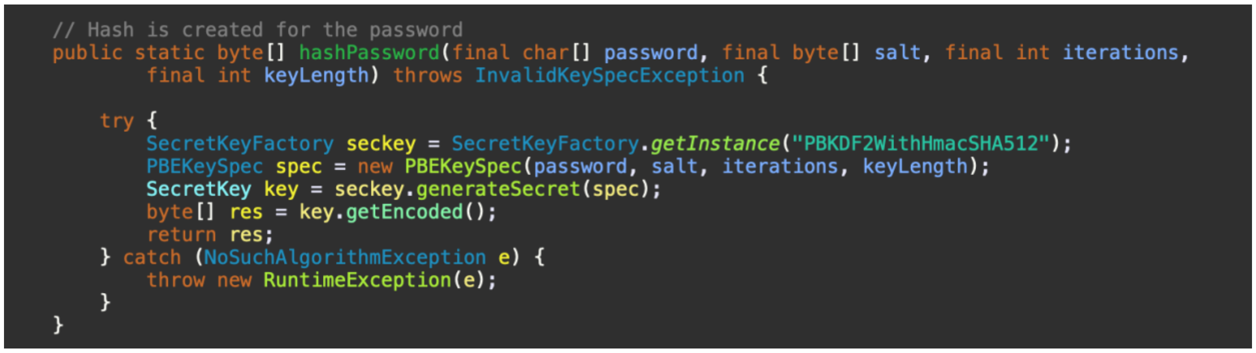
8.5 CLAIM - PASSWORD SALTING

Player's information is more important than anything and it is to be protected. Under this consideration, the password of the player is protected by the hashing by a unique hash value. But still, hashing leaves some possibilities of a password being recovered by an attacker, as for the hashed value can be the same for another player, (if they have the same password by chance) in the game database. Hence it is not a secure way and probability of risk is quite large. By considering all of these issues, we implemented salting to the player's password.

Since the hash value of the password is unique, we added a random value to the password to each user and then they are used in the hashing process. The salt argument should be random data and vary for each user [OWASP2016]. Thus, the hash values will be unique every for different users even with the same password.

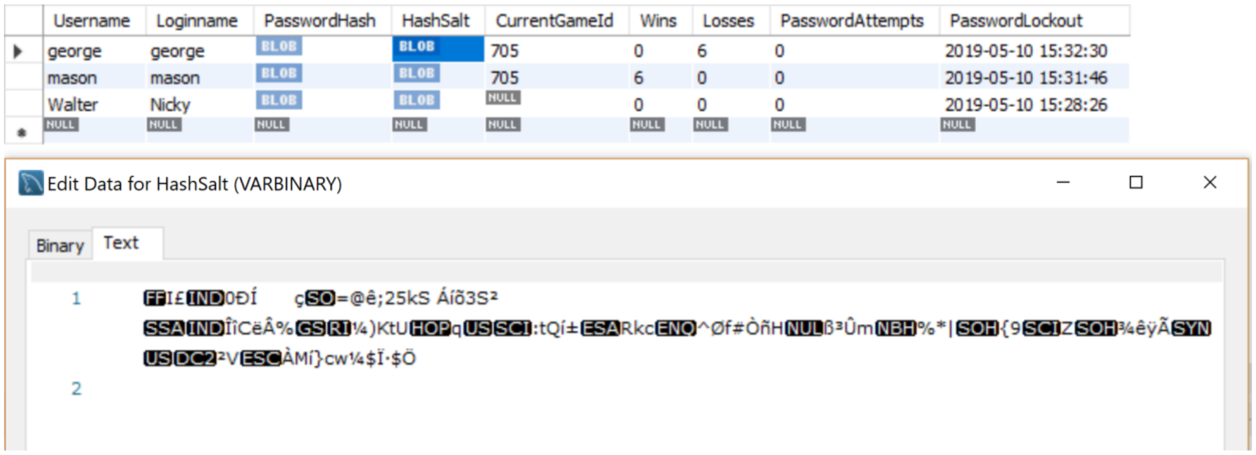
8.5.1 EVIDENCE

We have implemented secure storage of player's password by hashing it and salting. Once the password is salted, it is used to verify the user's password when login in attempt occur. Each salted password is unique from other. So even, players having similar passwords won't have same salted password. Key Length of 256 is used.



```java
// Hash is created for the password
public static byte[] hashPassword(final char[] password, final byte[] salt, final int iterations,
        final int keyLength) throws InvalidKeySpecException {

    try {
        SecretKeyFactory seckey = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
        PBEKeySpec spec = new PBEKeySpec(password, salt, iterations, keyLength);
        SecretKey key = seckey.generateSecret(spec);
        byte[] res = key.getEncoded();
        return res;
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 13: Password Salting



	Username	Loginname	PasswordHash	HashSalt	CurrentGameId	Wins	Losses	PasswordAttempts	PasswordLockout
▶	george	george	BLOB	BLOB	705	0	6	0	2019-05-10 15:32:30
	mason	mason	BLOB	BLOB	705	6	0	0	2019-05-10 15:31:46
	Walter	Nicky	BLOB	BLOB	NULL	0	0	0	2019-05-10 15:28:26
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
1    FFI£IND0ÐÍ     çSO=@ê;25kS Áíõ3S²
     SSAINDÎîCëÂ%GSRI¼)KtUHOPqUSSCI:tQí±ESARkcENQ^Øf#ÒñHNULß³ÛmNBH%*|SOH{9SCIZSOH¾êÿÃSYN
     USDC2²VESCÀMí}cw¼$Ï·$Ö
2
```

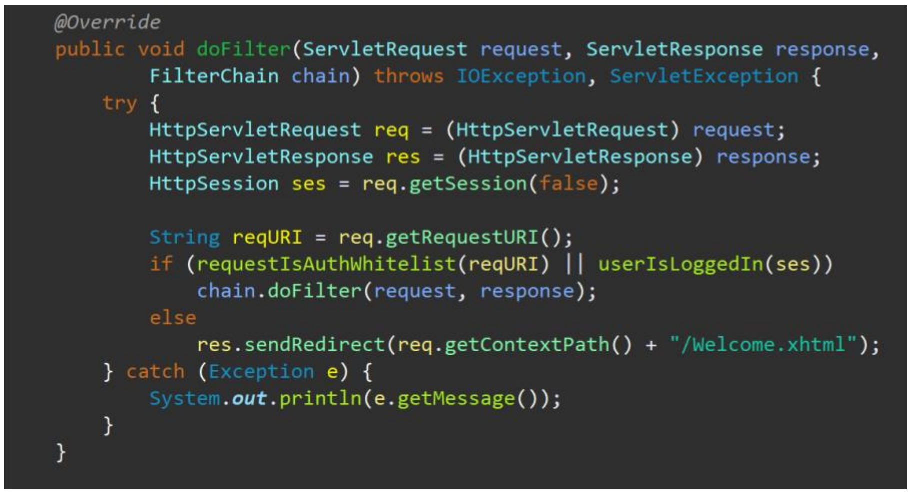
Figure 14: Salted Password stored in Database

8.6 CLAIM - SESSION ID

The access of each player to the game is managed by organizing a well established and authenticated session. Information about the player and other important credentials should not be leaked outside the session. Thus the user credentials are kept safe. It was assured that no unauthorized user or outsider can steal any data.

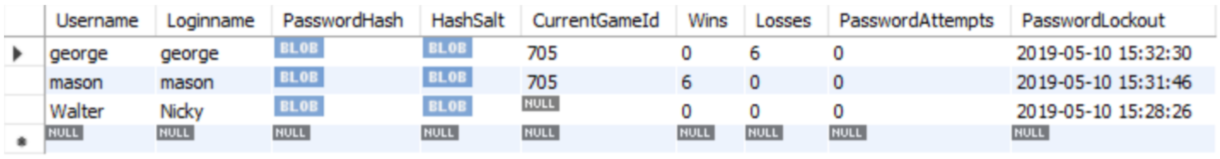
8.6.1 EVIDENCE

By creating secure session gameID between two players who are mutually in the game. This gameID is created when the first player starts new game and when the second player joins the game will receive the session. This avoids session hijacking to occur [WikiSession2019]. This makes clear that a particular game between two players cannot be interrupted by a third player because of the session which is being created.



```java
@Override
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
    try {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        HttpSession ses = req.getSession(false);

        String reqURI = req.getRequestURI();
        if (requestIsAuthWhitelist(reqURI) || userIsLoggedIn(ses))
            chain.doFilter(request, response);
        else
            res.sendRedirect(req.getContextPath() + "/Welcome.xhtml");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Figure 15: Session creation



Username	Loginname	PasswordHash	HashSalt	CurrentGameId	Wins	Losses	PasswordAttempts	PasswordLockout
george	george	BLOB	BLOB	705	0	6	0	2019-05-10 15:32:30
mason	mason	BLOB	BLOB	705	6	0	0	2019-05-10 15:31:46
Walter	Nicky	BLOB	BLOB	NULL	0	0	0	2019-05-10 15:28:26
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 16: Session created for Players

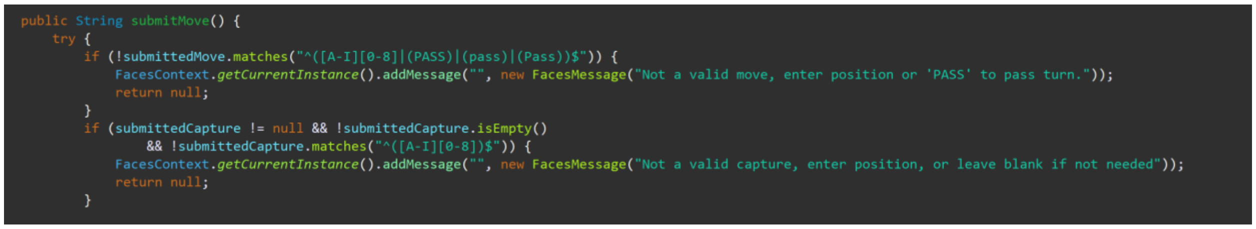
A shorter session time is given to our game so that an attacker cannot hijack the associated session. To prevent him from re-using it, a session time-out of 3 minutes is also added to our game. If the player is not active for 3 minutes, the session will log him out.

8.7 CLAIM- GAME MOVE VALIDATION

In the game, a move made by one player should not be modified by another player. Even invalid move cannot be made. The action cannot be done.

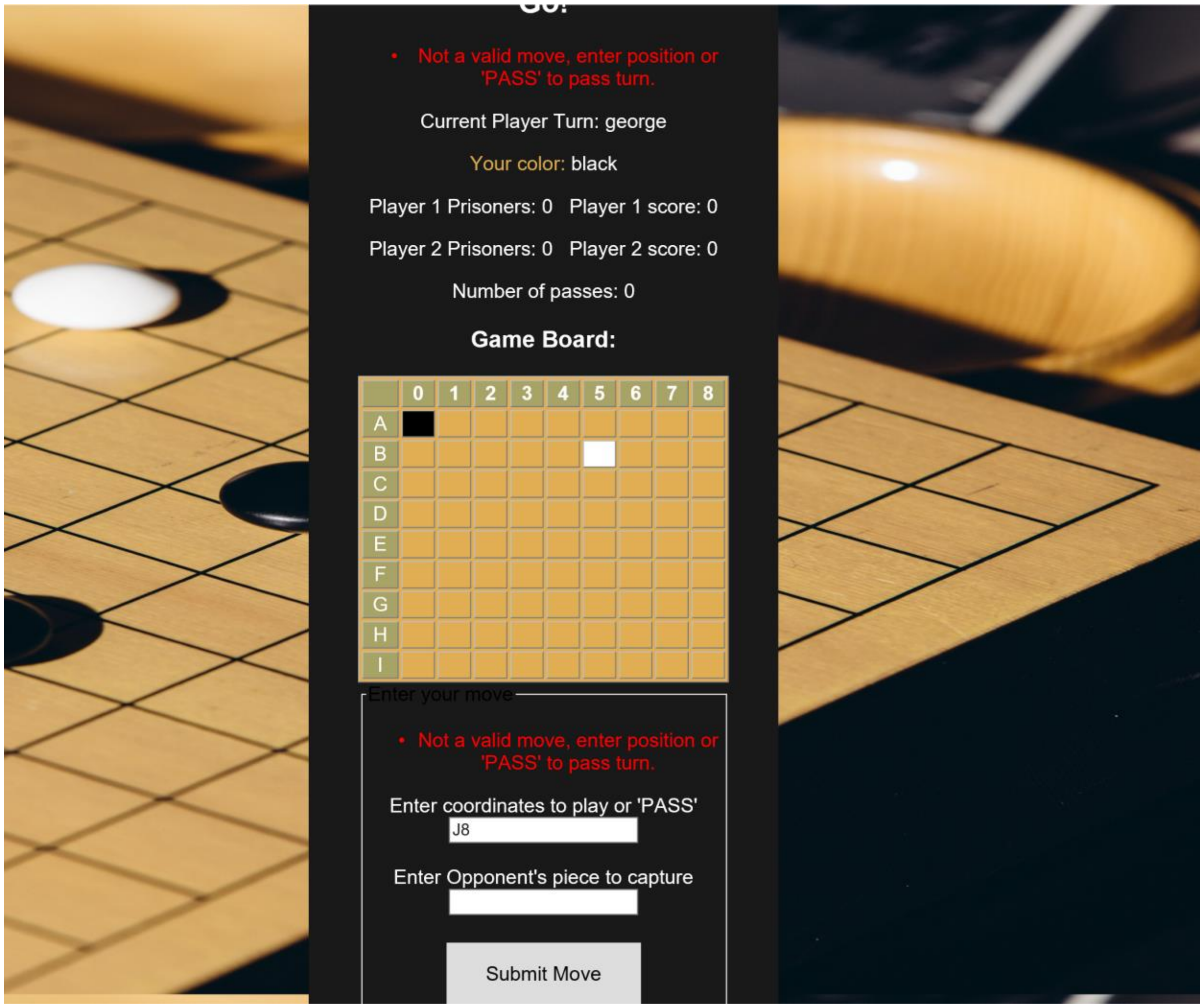
8.7.1 EVIDENCE

The move submitted by the user is nothing but sequence of an alphabet and a number. Anything other than this is not permitted. By giving regular expression on the moves we are ensuring that.



```
public String submitMove() {
    try {
        if (!submittedMove.matches("^([A-I][0-8]|(PASS)|(pass)|(Pass))$")) {
            FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Not a valid move, enter position or 'PASS' to pass turn."));
            return null;
        }
        if (submittedCapture != null && !submittedCapture.isEmpty()
                && !submittedCapture.matches("^([A-I][0-8])$")) {
            FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Not a valid capture, enter position, or leave blank if not needed"));
            return null;
        }
```

Figure 17: Move Validation using Regex



- Not a valid move, enter position or 'PASS' to pass turn.

Current Player Turn: george

Your color: black

Player 1 Prisoners: 0   Player 1 score: 0

Player 2 Prisoners: 0   Player 2 score: 0

Number of passes: 0

Game Board:

	0	1	2	3	4	5	6	7	8
A	■								
B						□			
C									
D									
E									
F									
G									
H									
I									

Enter your move

- Not a valid move, enter position or 'PASS' to pass turn.

Enter coordinates to play or 'PASS'
J8

Enter Opponent's piece to capture

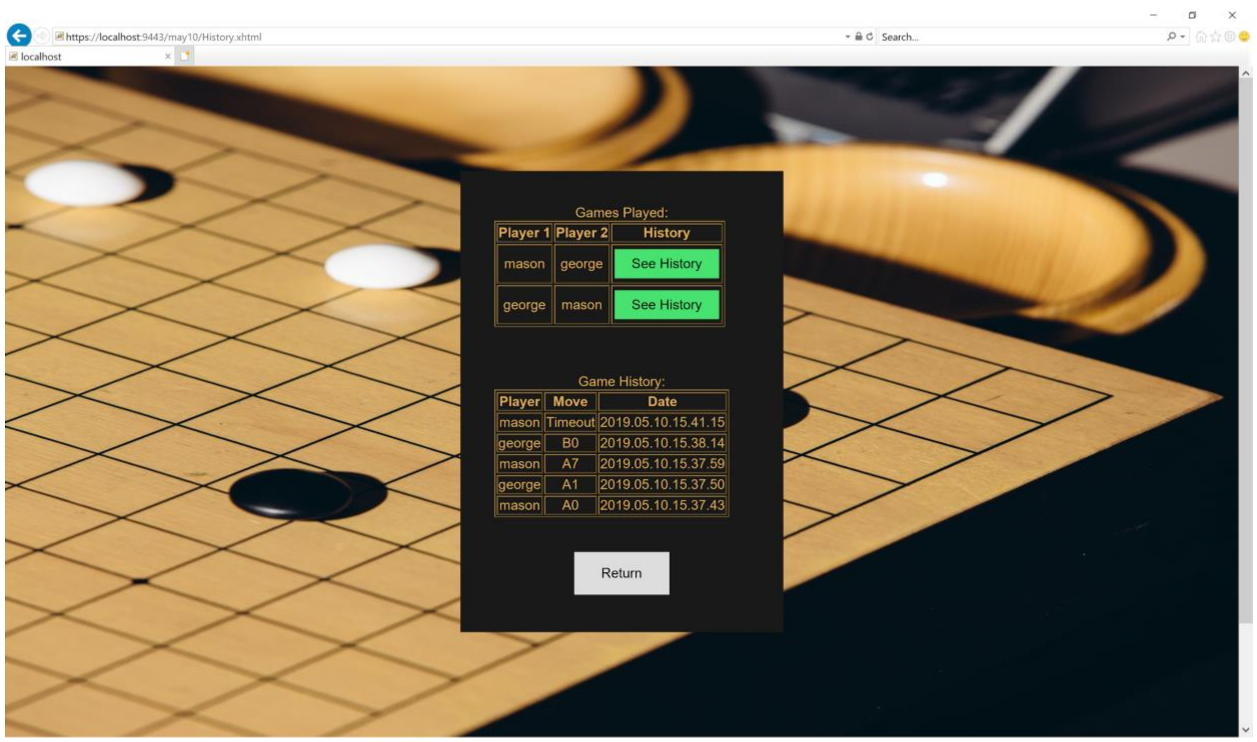
Figure 18: Move Validation in Game

8.8 CLAIM - READ ONLY ACCESS

The Game is designed in a way by which the players cannot modify or manipulate any change to the information about game history or scores in any way. By this, we have followed the least privilege method to prevent unnecessary access to the users.

8.8.1 EVIDENCE

The read only access is induced, so the players cannot manipulate their scores in any way. The Players have no access to modify the information available. By implementing this, it ensures we followed the least privilege method to prevent unnecessary access to the users.



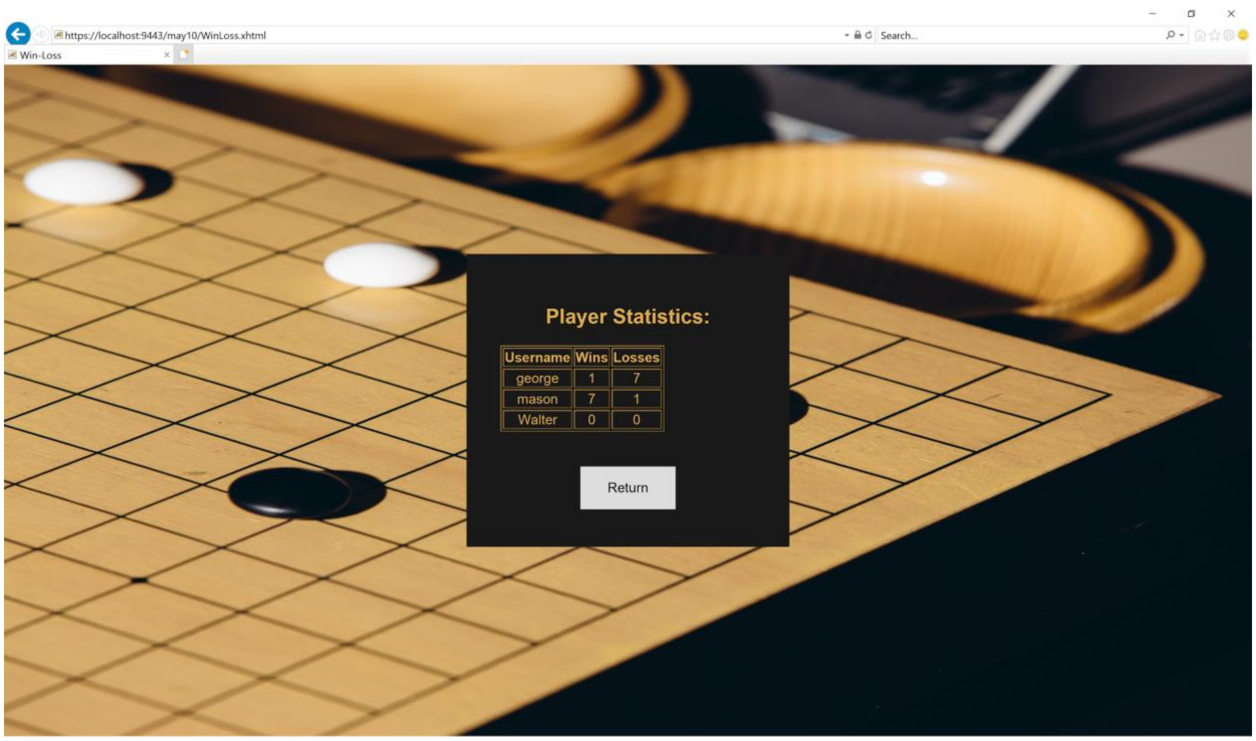
Games Played:

Player 1	Player 2	History
mason	george	See History
george	mason	See History

Game History:

Player	Move	Date
mason	Timeout	2019.05.10.15.41.15
george	B0	2019.05.10.15.38.14
mason	A7	2019.05.10.15.37.59
george	A1	2019.05.10.15.37.50
mason	A0	2019.05.10.15.37.43

Figure 19: Game History



Player Statistics:

Username	Wins	Losses
george	1	7
mason	7	1
Walter	0	0

Figure 20: Game Statistics

9. GAME REQUIREMENTS

The Game requirements which we have fulfilled are listed below,

1	Requirement	An Encrypted channel to be used when transmitting passwords
	Justification	Section 8.1.1
2	Requirement	Client-Server Architecture to be used
	Justification	Section 2.1
3	Requirement	Proper audit trails, show of game details
	Justification	A proper list of user history is shown inside of the game

4	Requirement	Password Salting
	Justification	Section 8.5
5	Requirement	Only permitted move must be allowed
	Justification	Section 8.7
6	Requirement	Session should be managed properly to safeguard player's sensitive data from attackers
	Justification	Section 8.6

10. CIA TRIAD

Confidentiality: There must be audit trails for every move which is made inside the game. This must be properly logged in some entity which can be accessed only after the completion of the game. When the game ends, these trails must be returned to the authenticated user as a final audit record, that means these records are to be made public once and only when the game is done. This depicts the confidentiality of the game. The records must be kept off of the unauthenticated users. These records include the final scores of the players and the win-loss data. It must ensure that these data are not leaked to any malicious or unauthenticated third person.

Justification: Data like the history of a player, name of another player who wants to join the game, etc, must be shown only after the user is logged in.

Integrity: It must be ensured that only correct data is set into audit trails and the databases. The moves of the player must be validated before reflecting the changes anywhere. For example, it should be made sure that the piece, the current player is capturing must actually be captured only with the legal move of the player, and only after validation, it should update the value. Make sure that the audit trail is not changed by the invalid or syntactically incorrect move. Records must not be changed unless the moves are justified to be legal.

Justification: Playing the move which is already there in the table must throw an appropriate error, notifying the player that it is invalid.

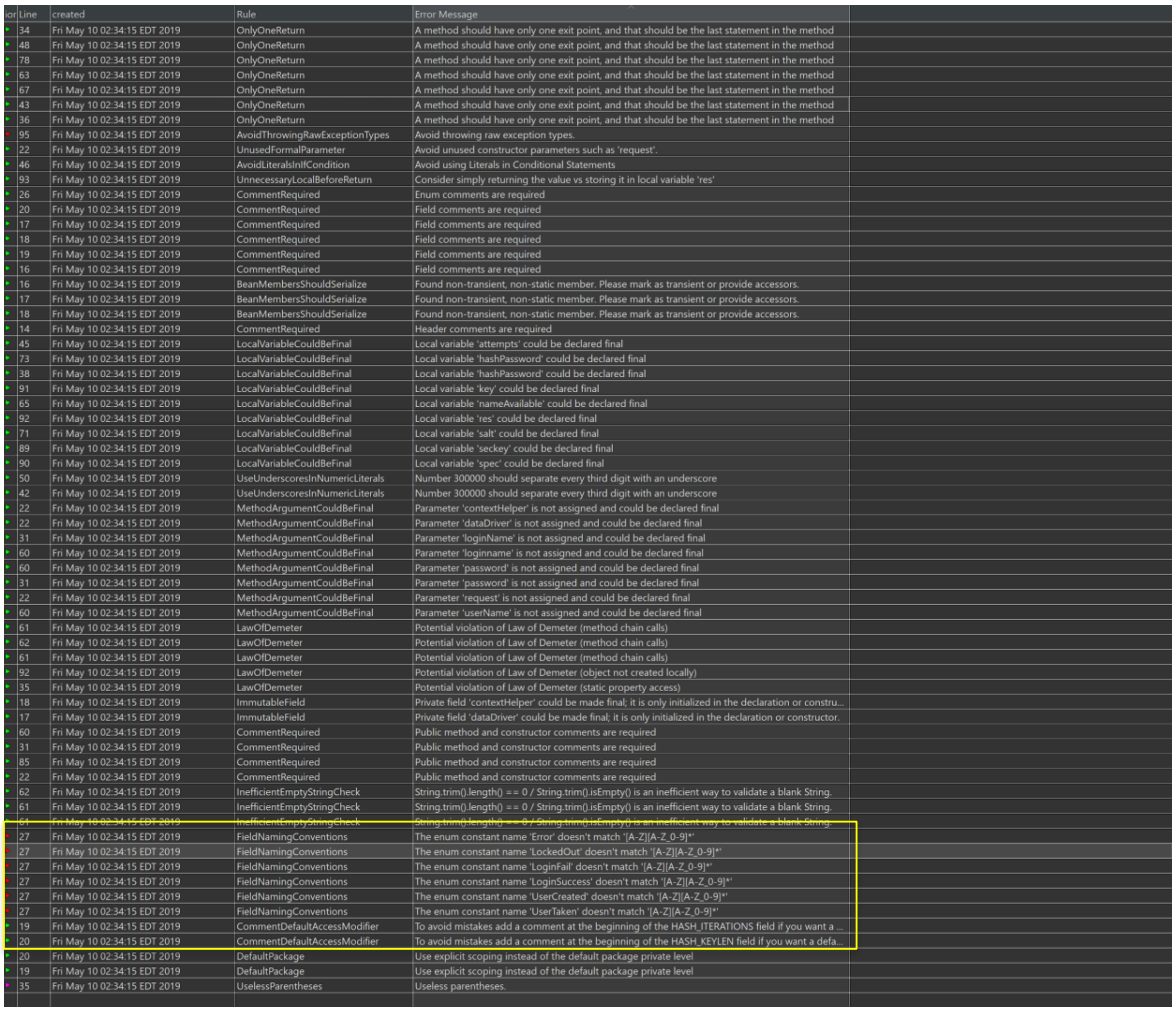
Availability: The game should have a timeout when the current player is taking too long to make the move. The game should not be paused forever. It should timeout after a span of time.

Justification: The timeout for each move is given 3 minutes.

11. PMD - STATIC ANALYSIS TOOL

A static analysis tool PMD is used for our game application to analyze the vulnerabilities of code. The red flagged vulnerabilities are understood and fixed in the application. Below are the images showing the before and after screenshots.

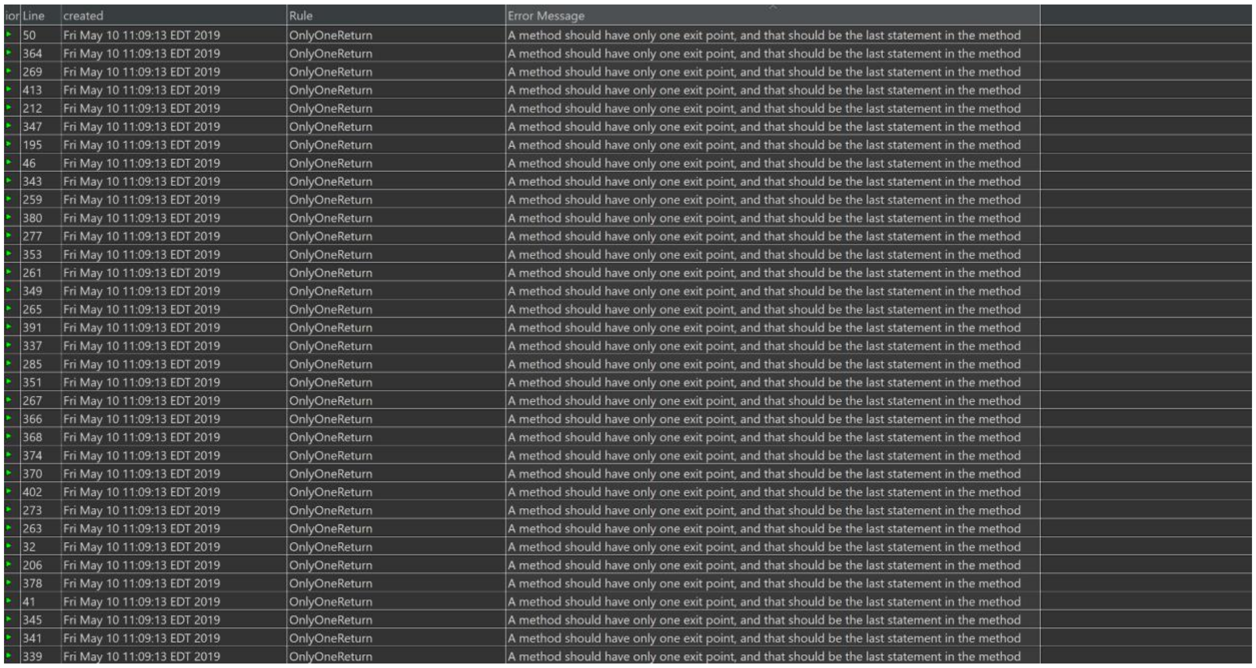
BEFORE:



ior	Line	created	Rule	Error Message
🟩	34	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	48	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	78	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	63	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	67	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	43	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟩	36	Fri May 10 02:34:15 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
🟥	95	Fri May 10 02:34:15 EDT 2019	AvoidThrowingRawExceptionTypes	Avoid throwing raw exception types.
🟩	22	Fri May 10 02:34:15 EDT 2019	UnusedFormalParameter	Avoid unused constructor parameters such as 'request'.
🟩	46	Fri May 10 02:34:15 EDT 2019	AvoidLiteralsInIfCondition	Avoid using Literals in Conditional Statements
🟩	93	Fri May 10 02:34:15 EDT 2019	UnnecessaryLocalBeforeReturn	Consider simply returning the value vs storing it in local variable 'res'
🟩	26	Fri May 10 02:34:15 EDT 2019	CommentRequired	Enum comments are required
🟩	20	Fri May 10 02:34:15 EDT 2019	CommentRequired	Field comments are required
🟩	17	Fri May 10 02:34:15 EDT 2019	CommentRequired	Field comments are required
🟩	18	Fri May 10 02:34:15 EDT 2019	CommentRequired	Field comments are required
🟩	19	Fri May 10 02:34:15 EDT 2019	CommentRequired	Field comments are required
🟩	16	Fri May 10 02:34:15 EDT 2019	CommentRequired	Field comments are required
🟩	16	Fri May 10 02:34:15 EDT 2019	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.
🟩	17	Fri May 10 02:34:15 EDT 2019	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.
🟩	18	Fri May 10 02:34:15 EDT 2019	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.
🟩	14	Fri May 10 02:34:15 EDT 2019	CommentRequired	Header comments are required
🟩	45	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'attempts' could be declared final
🟩	73	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'hashPassword' could be declared final
🟩	38	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'hashPassword' could be declared final
🟩	91	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'key' could be declared final
🟩	65	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'nameAvailable' could be declared final
🟩	92	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'res' could be declared final
🟩	71	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'salt' could be declared final
🟩	89	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'seckey' could be declared final
🟩	90	Fri May 10 02:34:15 EDT 2019	LocalVariableCouldBeFinal	Local variable 'spec' could be declared final
🟩	50	Fri May 10 02:34:15 EDT 2019	UseUnderscoresInNumericLiterals	Number 300000 should separate every third digit with an underscore
🟩	42	Fri May 10 02:34:15 EDT 2019	UseUnderscoresInNumericLiterals	Number 300000 should separate every third digit with an underscore
🟩	22	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'contextHelper' is not assigned and could be declared final
🟩	22	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'dataDriver' is not assigned and could be declared final
🟩	31	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'loginName' is not assigned and could be declared final
🟩	60	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'loginname' is not assigned and could be declared final
🟩	60	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'password' is not assigned and could be declared final
🟩	31	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'password' is not assigned and could be declared final
🟩	22	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'request' is not assigned and could be declared final
🟩	60	Fri May 10 02:34:15 EDT 2019	MethodArgumentCouldBeFinal	Parameter 'userName' is not assigned and could be declared final
🟩	61	Fri May 10 02:34:15 EDT 2019	LawOfDemeter	Potential violation of Law of Demeter (method chain calls)
🟩	62	Fri May 10 02:34:15 EDT 2019	LawOfDemeter	Potential violation of Law of Demeter (method chain calls)
🟩	61	Fri May 10 02:34:15 EDT 2019	LawOfDemeter	Potential violation of Law of Demeter (method chain calls)
🟩	92	Fri May 10 02:34:15 EDT 2019	LawOfDemeter	Potential violation of Law of Demeter (object not created locally)
🟩	35	Fri May 10 02:34:15 EDT 2019	LawOfDemeter	Potential violation of Law of Demeter (static property access)
🟩	18	Fri May 10 02:34:15 EDT 2019	ImmutableField	Private field 'contextHelper' could be made final; it is only initialized in the declaration or constru...
🟩	17	Fri May 10 02:34:15 EDT 2019	ImmutableField	Private field 'dataDriver' could be made final; it is only initialized in the declaration or constructor.
🟩	60	Fri May 10 02:34:15 EDT 2019	CommentRequired	Public method and constructor comments are required
🟩	31	Fri May 10 02:34:15 EDT 2019	CommentRequired	Public method and constructor comments are required
🟩	85	Fri May 10 02:34:15 EDT 2019	CommentRequired	Public method and constructor comments are required
🟩	22	Fri May 10 02:34:15 EDT 2019	CommentRequired	Public method and constructor comments are required
🟩	62	Fri May 10 02:34:15 EDT 2019	InefficientEmptyStringCheck	String.trim().length() == 0 / String.trim().isEmpty() is an inefficient way to validate a blank String.
🟩	61	Fri May 10 02:34:15 EDT 2019	InefficientEmptyStringCheck	String.trim().length() == 0 / String.trim().isEmpty() is an inefficient way to validate a blank String.
🟩	61	Fri May 10 02:34:15 EDT 2019	InefficientEmptyStringCheck	String.trim().length() == 0 / String.trim().isEmpty() is an inefficient way to validate a blank String.
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'Error' doesn't match '[A-Z][A-Z_0-9]*'
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'LockedOut' doesn't match '[A-Z][A-Z_0-9]*'
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'LoginFail' doesn't match '[A-Z][A-Z_0-9]*'
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'LoginSuccess' doesn't match '[A-Z][A-Z_0-9]*'
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'UserCreated' doesn't match '[A-Z][A-Z_0-9]*'
🟥	27	Fri May 10 02:34:15 EDT 2019	FieldNamingConventions	The enum constant name 'UserTaken' doesn't match '[A-Z][A-Z_0-9]*'
🟩	19	Fri May 10 02:34:15 EDT 2019	CommentDefaultAccessModifier	To avoid mistakes add a comment at the beginning of the HASH_ITERATIONS field if you want a ...
🟩	20	Fri May 10 02:34:15 EDT 2019	CommentDefaultAccessModifier	To avoid mistakes add a comment at the beginning of the HASH_KEYLEN field if you want a defa...
🟩	20	Fri May 10 02:34:15 EDT 2019	DefaultPackage	Use explicit scoping instead of the default package private level
🟩	19	Fri May 10 02:34:15 EDT 2019	DefaultPackage	Use explicit scoping instead of the default package private level
🟪	35	Fri May 10 02:34:15 EDT 2019	UselessParentheses	Useless parentheses.

Figure 21: PMD Analysis

AFTER:



ior	Line	created	Rule	Error Message	
▶	50	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	364	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	269	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	413	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	212	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	347	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	195	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	46	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	343	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	259	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	380	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	277	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	353	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	261	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	349	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	265	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	391	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	337	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	285	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	351	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	267	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	366	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	368	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	374	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	370	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	402	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	273	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	263	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	32	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	206	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	378	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	41	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	345	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	341	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	
▶	339	Fri May 10 11:09:13 EDT 2019	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method	

Figure 22: PMD Analysis-After fixing

The above images reveal that the highlighted vulnerabilities has been fixed.

12. CONCLUSION

This game we develop is to learn about secure software design and programming. We learned different aspects of secure programming and implemented all that was taught in the class. Practical implementation of the project was very helpful, as on the way we learned lot new things. We are aware that not any software can be fully secured. We tried to reduces as many security issues and vulnerabilities as possible. There are claims and evidences given in the report to prove that and we believe we followed all possible methods to make our game secure. Now that we are greatly aware how important is security in any phase of a software development, and we hope all these aspects will be helpful in future.

13. REFERENCES

[WikiGo2019]	https://en.wikipedia.org/wiki/Go_(game)
[Dierks2008]	T. Dierks; E. Rescorla. "The Transport Layer Security (TLS) Protocol, Version 1.2". 2008 https://tools.ietf.org.html/rfc5246
[WikiGoRules2019]	https://en.wikipedia.org/wiki/Rules_of_Go
[Wheeler2015]	https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/validation-tools-regex.html
[OWASP2016]	https://www.owasp.org/index.php/Hashing_Java
[WikiSession2019]	https://en.wikipedia.org/wiki/Session_hijacking
[WikiSSL2019]	https://en.wikipedia.org/wiki/Go_(game)
[MMC2018]	https://docs.microsoft.com/en-us/skype-sdk/sdn/articles/installing-the-trusted-root-certificate
[PMD2017]	https://pmd.github.io/latest/index.html