Ludo Game

SWE 681 Programming Project – Fall 2013

Alireza Sadeghi

INTRO

The "Ludo" project, which is a web-based multiplayer implementation of the Ludo game, is implemented as the programming project of SWE 681/ ISA 681 (Secure Software Design and Programming), Fall 2013. The project is designed and implemented by Alireza Sadeghi. This report first describes the architectural elements of the projects. Then, the installation guide, playing instructions and rules are provided. Finally, the assurance case of the project, which shows the security mechanism considered in the project's design and implementation, is elaborated.

Table of Contents

Architecture

Figure 1 shows the overview of the project architecture. Accordingly, each layer receives/provides services form/to the lower/upper layer.

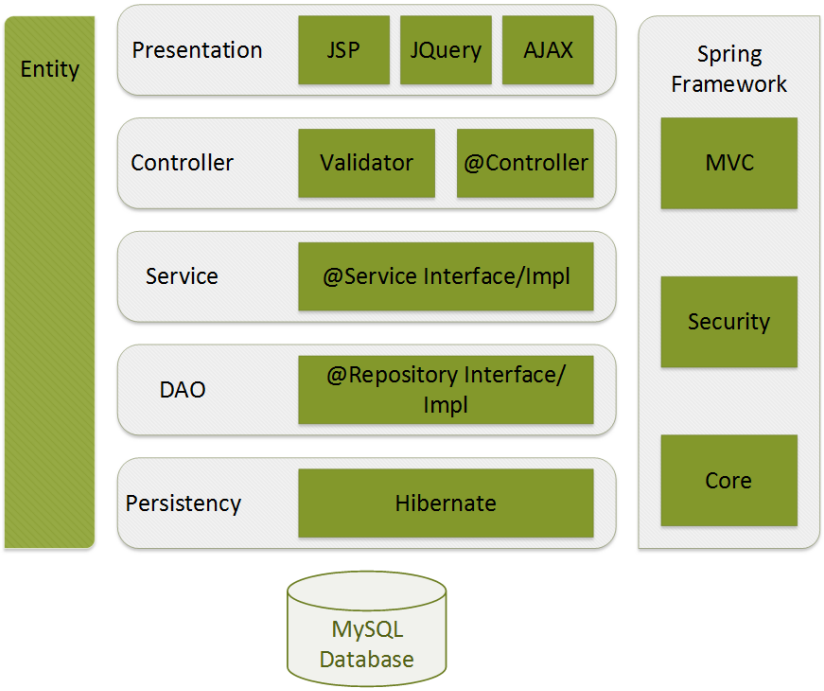


Figure 1 - Architecture Overview

In the bottom tier, MySQL as the database management system provides the data storage and retrieval capabilities. In the next layer, Hibernate [1] provides Object/Relational mapping. Data Access Object (DAO) and Service layer together implement the business logic of the system. Controller layer works as the mediator between the business logic (model) and the presentation layer. The server side validation is also implemented in this layer. Finally, the presentation layer provides the web-based graphical user interface of the project. Since this project is an online game, a rich interactive user interface is required. Thus, JQuery and AJAX technologies are leveraged beside the Java Servlet Pages (JSPs).

The entity objects that represent the model of the Ludo game, interacts with all layers. Figure 2 depicts the structural dependencies (Class diagram) of the entity Objects.

GameBoard class represents each instance of the Ludo game that is created and played. *Player* class represents each *User*, who is currently involved in a game instance. Each game contains between 2-4 *Player*s and each player has 4 *Piece*s to play with. *AuditTrail* class records every movement of each player in each game.

[1] www.hibernate.org

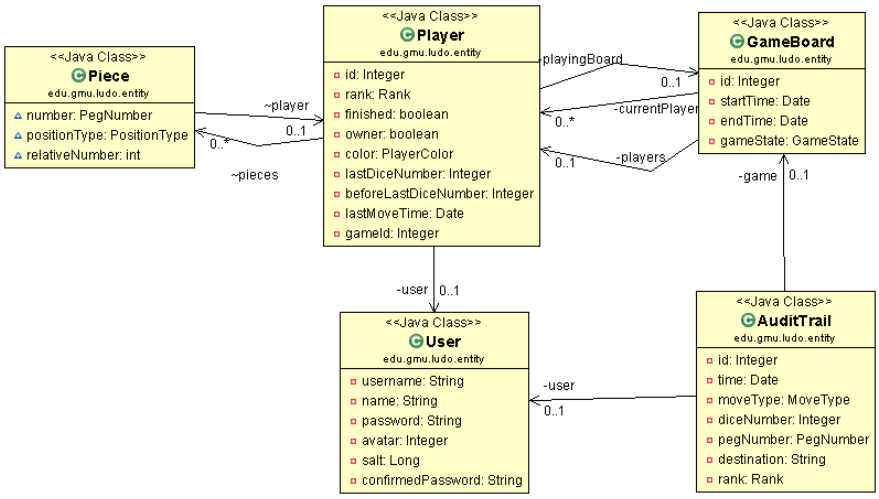


Figure 2 Class Diagram

Figure 3 shows the four controllers of the project and their related views. The green icons show the JSP pages forwarded by each controller, and the red icons represent the AJAX responses return by them. *UserController* handles the creation of new user as well as updating the users information, such as full name (not username!) and password. *GameManagementController* manages the requests related to creating a new game or joining an existing game. *GameBoardContoller* controls the board of each game instance, for example it initialize the game board page, and then refresh the dynamic part of it (including dice, pieces and players status) of the board by AJAX request/responses. Finally, *GameStatisticsController* handles the game statistics and audit trail pages.

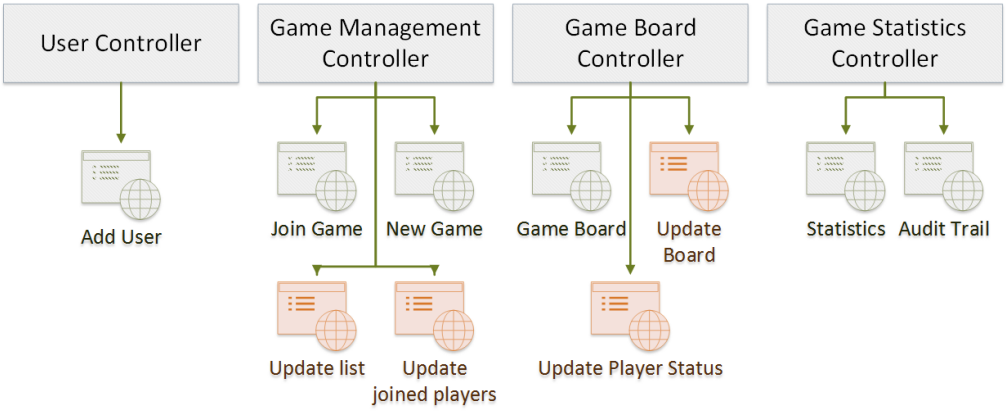


Figure 3 - Controllers and Views

Installation

Requirements:

1. **MySQL Community Server 5**[2]
2. **Apache Tomcat 7**[3]

Installation steps:

1. Install MySQL Community Server 5 and Apache Tomcat 7.
2. Database preparation:
   a. Execute the Ludo.sql script. This script will create a schema named **ludo** and the four tables shown in Figure 4.

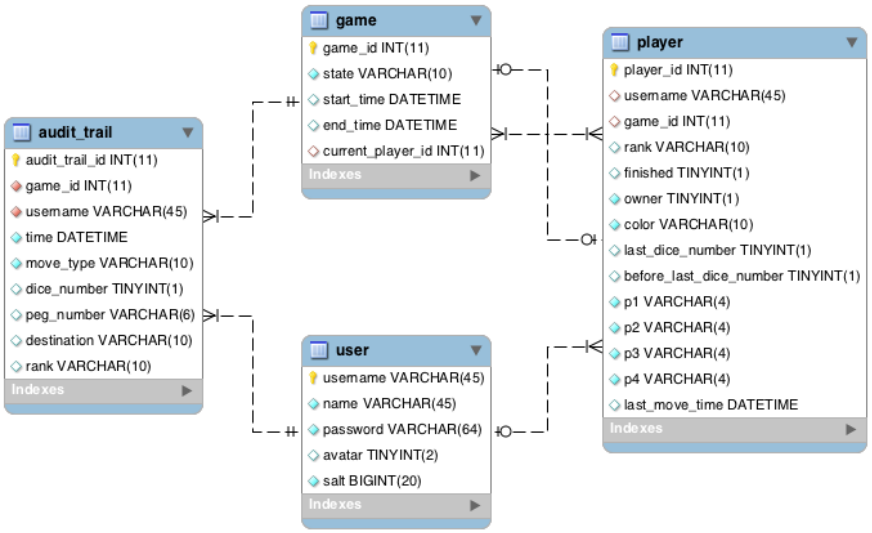


Figure 4 Database Tables

   b. Create a database user with the access rights "SELECT, INSERT, UPDATE" for the schema 'ludo' created in the previous step.
   Note: Default username/password are `swe681/Ludo@Swe_681 (the password is encrypted)`. If a user with the same username/password is created, ignore step 2.c.

   c. The *database.property* file (in /Ludo/src/main/resources) should be updated based on the created user's properties. Note that the password is encrypted (See the Vulnerability #9 at the section Checking against Common Vulnerabilities) and you can use the use the `encrypt` method in `EncryptedDataSource` utility class to encrypt your password.

```
…
jdbc.user=swe681
jdbc.password=hoByACD7P0+8Dk6xGXM6hg==
```

[2] www.mysql.com
[3] tomcat.apache.org

3. Tomcat preparation:
   a. Insert the https port definition into the server.xml (TOMCAT_INSTALL/conf), as follows:

```
<Connector
        port="8443" maxThreads="200" scheme="https" secure="true"
        SSLEnabled="true"keystoreFile="keyStorePath/.ludo_keyStore"
        keystorePass="Swe_681" clientAuth="false" sslProtocol="TLS" />
```

   b. Copy the `.ludo_keyStore` from the installation package to the path set in server.xml (***keyStorePath***) in the previous step.
   c. Put the Ludo package in the (TOMCAT_INSTALL/webapps) and start the tomcat. (TOMCAT_INSTALL/bin/startup).

Note: All jar file dependencies are managed by Maven[4], and the required jars are put into the ludo app bundle.

Operating Instructions

The project package name is Ludo. Hence, if the tomcat http port is (default) 8080, the project is accessible by following address:

http://localhost:8080/Ludo

Note: The http channel is automatically redirected to https (8443 default port).

Independent of the web page (url) you are requested to access, you have to login into the system, if you have not already logged in. Also, you can create a new user if you don't have a user.

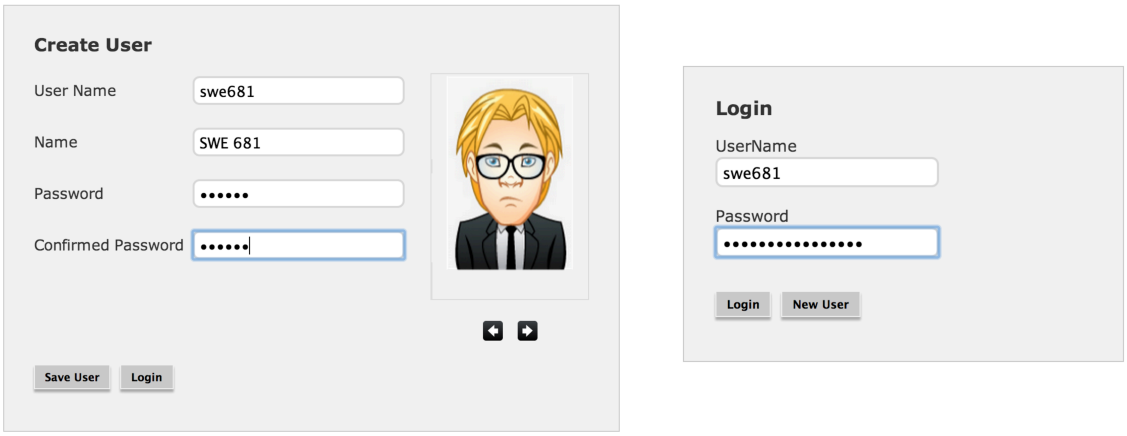


Figure 5 A) Create User (Left), B) Login (Right)

[4] maven.apache.org

The Ludo main page looks like this:

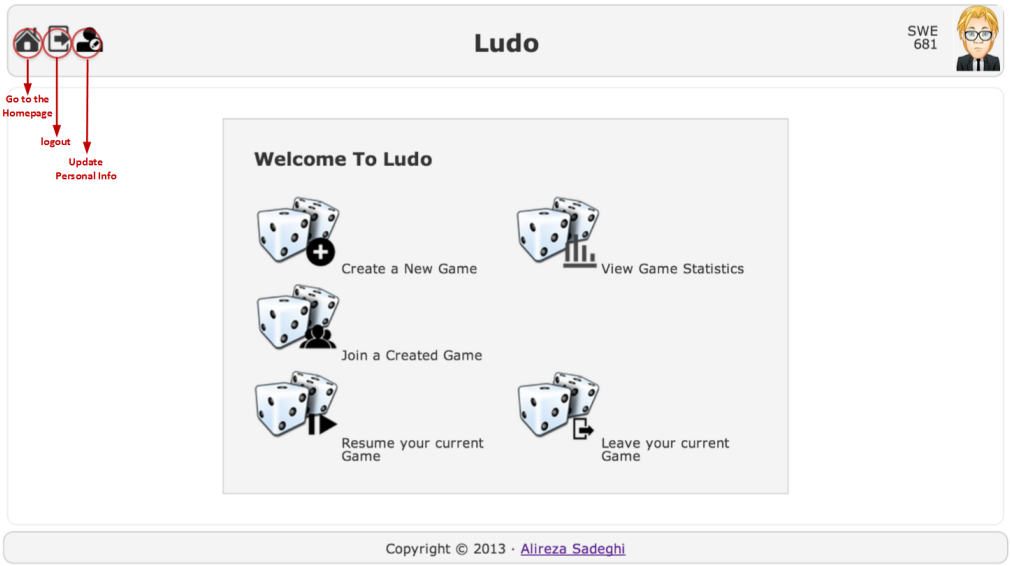


Figure 6 Ludo main page

Based on the status the user, the following menus are available in the main page:

- Create a new game.
- Join a created game.
- View Game Statistics
- Resume the current game (available if user is in the middle of a game)
- Leave the current game (available if user is in the middle of a game)

You can create a new game or join the created (but not started) games. Once the minimum players (2 players) joined the game, the game owner (the users who has created the game) can start the game.

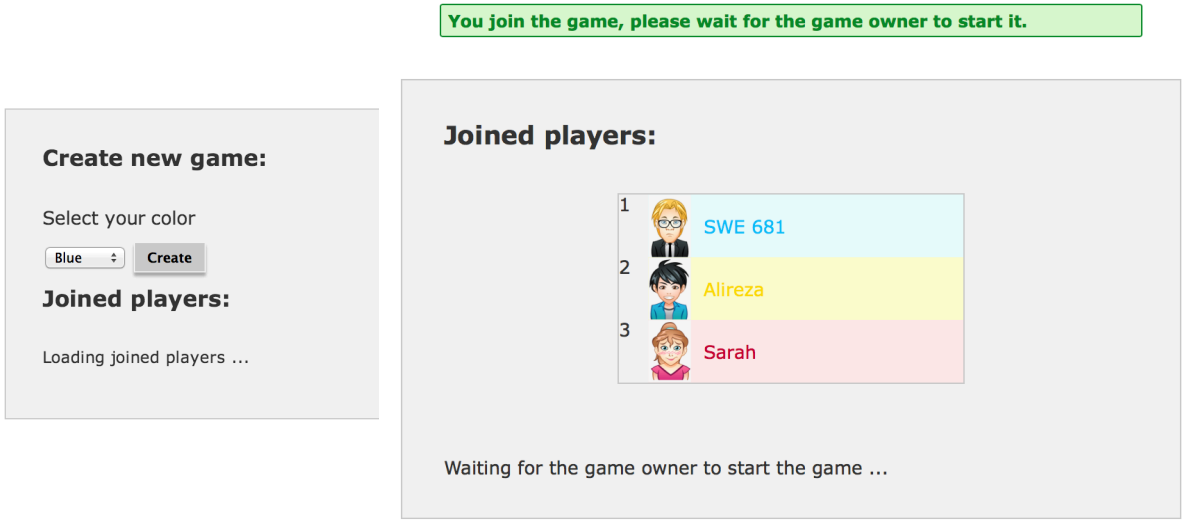


Figure 7 A) Create (Left), B) Join Game (Right)

After the game is started, you can play the game based on its rules. The Ludo game board looks like this:

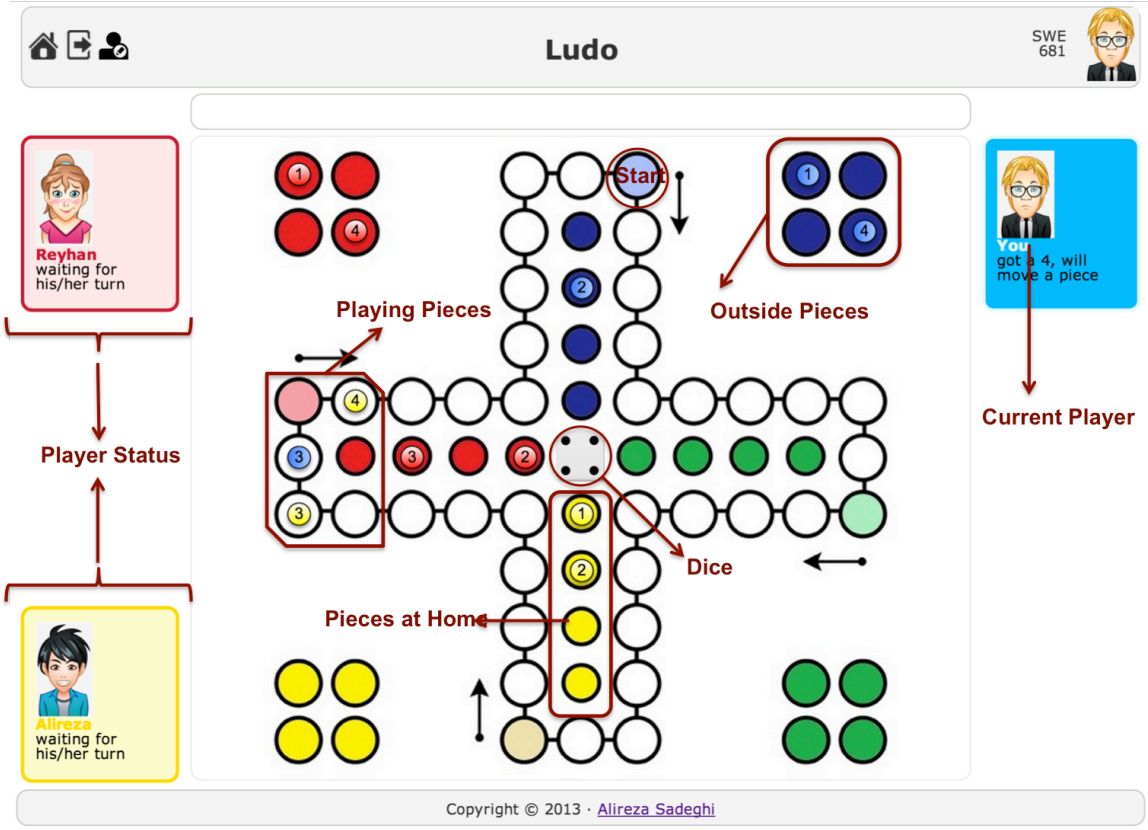


Figure 8 Game board

After the game is finished, its statistics are visible in game statistics:

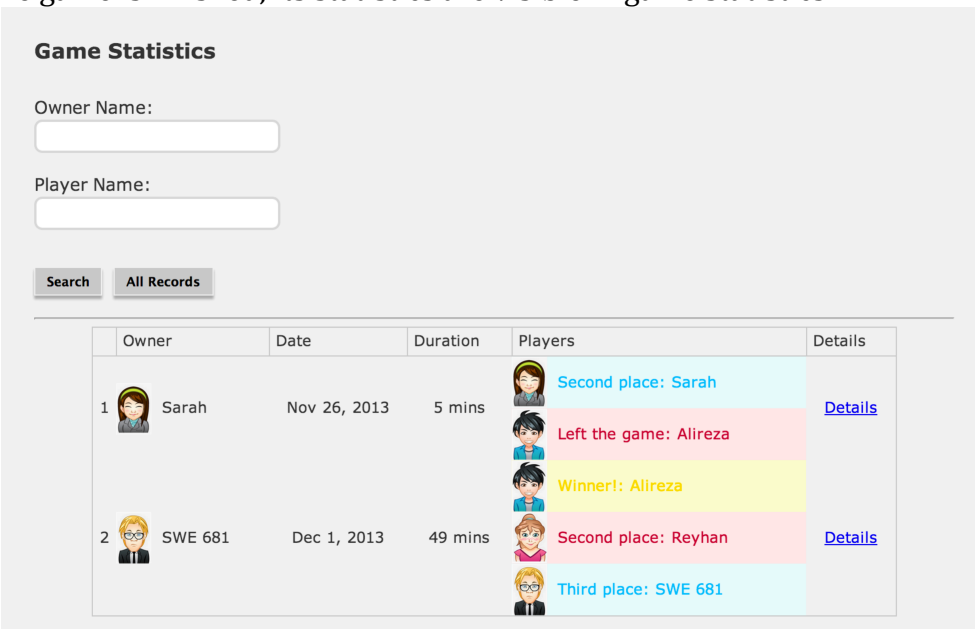
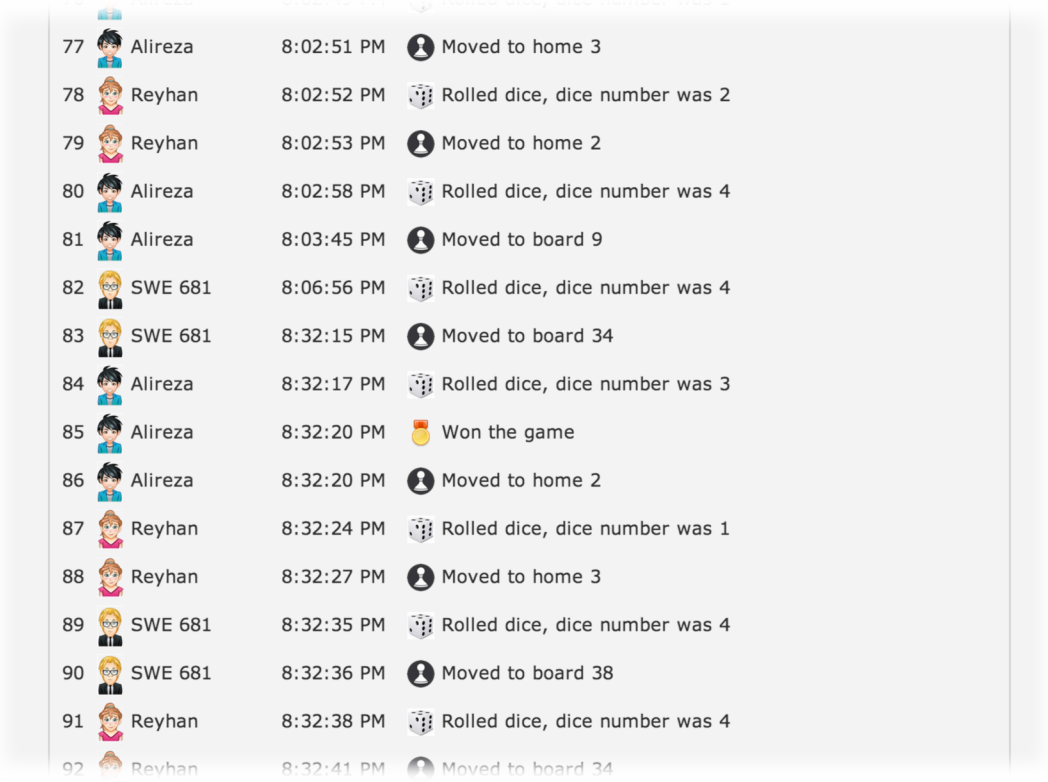


Figure 9 Game statistics

And the details (Audit Trail) of each game are available too.



77	Alireza	8:02:51 PM	Moved to home 3
78	Reyhan	8:02:52 PM	Rolled dice, dice number was 2
79	Reyhan	8:02:53 PM	Moved to home 2
80	Alireza	8:02:58 PM	Rolled dice, dice number was 4
81	Alireza	8:03:45 PM	Moved to board 9
82	SWE 681	8:06:56 PM	Rolled dice, dice number was 4
83	SWE 681	8:32:15 PM	Moved to board 34
84	Alireza	8:32:17 PM	Rolled dice, dice number was 3
85	Alireza	8:32:20 PM	Won the game
86	Alireza	8:32:20 PM	Moved to home 2
87	Reyhan	8:32:24 PM	Rolled dice, dice number was 1
88	Reyhan	8:32:27 PM	Moved to home 3
89	SWE 681	8:32:35 PM	Rolled dice, dice number was 4
90	SWE 681	8:32:36 PM	Moved to board 38
91	Reyhan	8:32:38 PM	Rolled dice, dice number was 4
92	Reyhan	8:32:41 PM	Moved to board 34

Figure 10 Game audit trail

Game Rules

In this project a modified version of Ludo, named **"Mensch ärgere dich nicht"**[5], is implemented. The game rules are quoted from Wikipedia[6] :

> "The game can be played by 2, 3 or 4 players – one player per board side. Each player has 4 game pieces, which are in the "out" area when the game starts, and which must be brought into the player's "home" row.
>
> The rows are arranged in a cross position. They are surrounded and connected with a circle of fields, over which the game pieces move in clockwise direction. There are 3 fields nearest to each side of the board; the left one is the player's "start" field (highlighted with player's color) and the middle one leads to the "home" row. This means that each game piece enters the circle at the "start" field, moves (clockwise) over the board and finally enters the "home" row. The first player with all of their pieces in their "home" row wins the game.
>
> The players throw a dice in turn and can advance any of their pieces in the game by the thrown number of dots on the dice. Throwing a six means bringing a piece into the game (by placing one

[5] It is originally a German game. In Farsi, we call it just "Mench".
[6] http://en.wikipedia.org/wiki/Mensch_%C3%A4rgere_dich_nicht

from the "out" area onto the "start" field) and throwing the dice again. If a piece is on the "start" field and there are still pieces in the "out" area, it must be moved as soon as possible. If a piece cannot be brought into the game then any other piece in the game must be moved by the thrown number, if that is possible.

Pieces can jump over other pieces, and throw out pieces from other players (into that player's "out" area) if they land on them. A player cannot throw out his own pieces though, and cannot advance further than the last field in the "home" row."

Assurance case

In this section we provide the arguments and justifications to show that this project fulfills the security constraints. In this regard, we first show that in this project the common vulnerabilities are considered and prevented. Then, the satisfaction of security requirements defined in the project spec is checked in the project design and implementation. Finally, **Fortify** static analysis tool is used to detect any vulnerability missed by manual inspection.

Checking against Common Vulnerabilities

In this subsection, the project is checked against a list common vulnerabilities and weaknesses, provided by OWASP top 10 (2013) and CWE/SANS top 25.

<table>
<tr><td>1</td><td>Vulnerability</td><td>A1: Injection
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td></tr>
<tr><td></td><td>Argument</td><td>• Hibernate built-in APIs are used that prevents SQL injection.
• Spring[7] validation is used for input validation. (Samples are provided in other arguments)</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>Parameterized Query:
<pre>String hql = "SELECT DISTINCT game FROM GameBoard game WHERE game.gameState = :state";
return getCurrentSession().createQuery(hql).setString("state", state.name()).list();</pre>Hibernate find by Id method
<pre>public UserDetails loadUserByUsername(String username){
    return userDao.findByUsername(username);
}
public E findById(I id) {
    return (E) getCurrentSession().get(entityClass, id);
}</pre>The second sample retrieves the user information, used by Spring Security for authentication.</td></tr>
<tr><td>2</td><td>Vulnerability</td><td>A2: Broken Authentication and Session Management
A6: Sensitive Data Exposure
CWE-306: Missing Authentication for Critical Function
CWE-311: Missing Encryption of Sensitive Data</td></tr>
</table>

[7] http://spring.io

	Argument	*Spring Security* is used authentication and session management.
	Justification (sample code)	1. All request in http (particularly for login and create user page that password is sent) are automatically redirected to https channel to encrypt the sensitive data. 2. After logging out, the user session is invalidated. 3. Spring Security's Session Fixation Protection assigns a new session id, for each new session. `<security:http use-expressions='true' auto-config="true">` `<security:intercept-url pattern="/login.htm" access="permitAll" requires-channel="https" />` `<security:intercept-url pattern="/addUser.htm" access="permitAll" requires-channel="https" />` `<security:intercept-url pattern="/**" access="isAuthenticated()" requires-channel="https" />` `<security:form-login login-page="/login.htm" authentication-failure-url="/login.htm?login_error=1" default-target-url="/index.htm" always-use-default-target="true" />` `<security:logout logout-url="/j_spring_security_logout" delete-cookies="JSESSIONID" invalidate-session="true" />` `</security:http>` `<security:authentication-manager alias="authenticationManager">` `<security:authentication-provider ref="authenticationProvider" />` `</security:authentication-manager>`
3	Vulnerability	**A3:** Cross-Site Scripting (XSS) **A8:** Cross-Site Request Forgery (CSRF) **CWE-79:** Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') **CWE-352:** Cross-Site Request Forgery (CSRF)
	Argument	*Spring* validation is used for server side input validation.
	Justification (sample code)	1. In controllers, all objects created from user inputs are tagged by **@Valid** to apply the spring validation on it before further processing: `@RequestMapping(value="/addUser.htm",method=RequestMethod.POST)` `public String saveUser(Model model, @ModelAttribute @Valid User user, BindingResult result) {` `if (result.hasErrors()) {` `return "gameManager/addUser";` `}` 2. In *spring validators,* whitelist checking is performed: `...` `if (!user.getUsername().matches("^([a-zA-Z_0-9]){5,}$"))` `errors.put("username", "user.username.invalid");` For example, the above regex checks if the username is valid. (i.e. at least 5 alphanumeric plus _ characters.)
4	Vulnerability	**A4:** Insecure Direct Object References **A7:** Missing Function Level Access Control **CWE-250:** Execution with Unnecessary Privileges
	Argument	• Layered architecture, which each layer has its own security mechanism and provide the service transparently, acts as a defense-in-depth mechanism and prevents the layers from directly accessing the resources of non-adjacent layers. • According to the least-privilege principle, database user for

<table>
<tr><td></td><td></td><td>the application has only the access rights to SELECT, UPDATE, and DELETE the records in this project schema (4 tables).</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>1. Controller beans can only access to Service beans:

```
@Controller
public class GameManagementContoller {
        @Autowired
        private LudoService ludoService;
```

Service beans can only access to DAO beans:

```
@Service
public class LudoServiceImpl implements LudoService, … {
        @Autowired
        private UserDao userDao;
        @Autowired
        private PlayerDao playerDao;
        …
```

and DAO beans can only access to Hibernate (Persistency) APIs:

```
@Repository
public class UserDaoImpl extends … {
        @Autowired
        private SessionFactory sessionFactory;
```

2. *root* user is not used as the application's database user. And access rights of defined user is based on least-privilege:

```
!SWE_681: This db user has granted limited rights -- Just Data
manipulation language (DML), not Data definition language.
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ludo
jdbc.user=swe681
```
</td></tr>
<tr><td>5</td><td>Vulnerability</td><td>A9: Using Components with Known Vulnerabilities
CWE-561: Dead Code
CWE-676: Use of Potentially Dangerous Function</td></tr>
<tr><td></td><td>Argument</td><td>The two major frameworks that are used in this project, namely Hibernate and Spring, have an active development team, forum and community that consider and resolve the reported security issues.</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>The last version of both Hibernate and Spring are used in this project. Also their dependencies are managed by Maven, which is a trusted repository. A part POM file of the project:

```
<properties>
   <org.springframework.version>
        3.2.1.RELEASE
   </org.springframework.version>
   <org.hibernate.version>
        4.1.5.Fina
   </org.hibernate.version>
</properties>
…
<!-- Spring framework -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${org.springframework.version}</version>
</dependency>
…
```

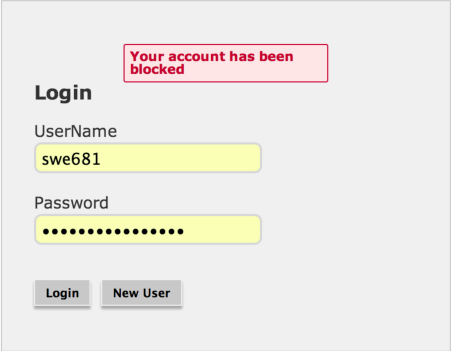
The guideline provided by OWASP to use Hibernate securely is also considered in this project.</td></tr>
</table>

<table>
<tr><td>6</td><td>Vulnerability</td><td>CWE-328: Reversible One-Way Hash
CWE-327: Use of a Broken or Risky Cryptographic Algorithm
CWE-759: Use of a One-Way Hash without a Salt</td></tr>
<tr><td></td><td>Argument</td><td>In this project the passwords are encrypted by SHA-256, which is provided by Spring Security. The hash is salted by a pseudorandom number generator (PRNG), which is generated by Java SecureRandom method.</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>Hash function definition is spring securityContex:
<pre>
<bean id="authenticationProvider"
class="org.springframework.security.authentication.dao.DaoAuthenti
cationProvider">
<property name="userDetailsService" ref="userDetailsService" />
<property name="passwordEncoder" ref="passwordEncoder" />
<property name="saltSource">
        <bean class="org.springframework.security...">
           <property name="userPropertyToUse" value="salt" />
        </bean>
</property>
</bean>

<bean class="org.springframework.security...ShaPasswordEncoder"
id="passwordEncoder">
        <constructor-arg value="256" />
        <property name="iterations" value="1000" />
</bean>
</pre>
Using PRNG to generate the salt for the hash function:
<pre>
public void encodePassword(ShaPasswordEncoder passwordEncoder) {
  this.salt = new SecureRandom().nextLong();
  this.password = passwordEncoder.encodePassword(password, salt);
}
</pre></td></tr>
<tr><td>7</td><td>Vulnerability</td><td>CWE-862: Missing Authorization
CWE-863: Incorrect Authorization</td></tr>
<tr><td></td><td>Argument</td><td>In this project the least-privilege principle is applied in authorizing the uses. The access of the anonymous user is restricted to all resources, except couple of pre-specified ones.</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>In spring security context, the access of the anonymous user is restricted to all resources (/**), except couple of pre-specified:
<pre>
<security:intercept-url pattern="/**" access="isAuthenticated()"/>

<security:intercept-url pattern="/login.htm" access="permitAll" />
<security:intercept-url pattern="/logout.htm" access="permitAll"/>
<security:intercept-url pattern="/resources/**/*.*" access="permitAll"/>
<security:intercept-url pattern="/addUser.htm" access="permitAll" />
</pre></td></tr>
<tr><td>8</td><td>Vulnerability</td><td>CWE-307: Improper Restriction of Excessive Authentication Attempts</td></tr>
<tr><td></td><td>Argument</td><td>The failed login attempts of users are logged (into the database), and if its number reached to a predefined threshold (by default set 5), the user account will be blocked.</td></tr>
<tr><td></td><td>Justification (sample code)</td><td>I have wrapped the standard implementation of Spring's <code>AuthenticationProvider</code> and extended it to record the failed login attempt and block the users if necessary.</td></tr>
</table>

```
try {
   User user = ludoService.getUserByUsername(username);
   if(user != null && user.isBlocked())
         throw new BadCredentialsException(BLOCKED);
   Authentication authenticate = super.authenticate(authentication);
         return authenticate;
   } catch (BadCredentialsException e) {
         ludoService.logFailedLogin(username);
         throw e;
}
```



9	Vulnerability	**CWE-311:** Missing Encryption of Sensitive Data. (Storing a plaintext password in a configuration file)
	Argument	***Note:** This vulnerability detected by Fortify analysis. The database password is encrypted by AES algorithm and the encrypted value is put in the config file.
	Justification (sample code)	`jdbc.driver=com.mysql.jdbc.Driver` `jdbc.url=jdbc:mysql://localhost:3306/ludo` `jdbc.user=swe681` `jdbc.password=hoByACD7P0+8Dk6xGXM6hg==` The Cipher class is used for password encryption (with AES algorithm). `public static String decrypt(String encryptedData) throws Exception {` `  Key key = generateKey();` `  Cipher c = Cipher.getInstance(ALGORITHM);` `  c.init(Cipher.DECRYPT_MODE, key);` `  byte[] decordedValue = new BASE64Decoder().decodeBuffer(encryptedData);` `  byte[] decValue = c.doFinal(decordedValue);` `  String decryptedValue = new String(decValue);` `  return decryptedValue;` `}`
10	Vulnerability	
	Argument	Error information sent back to the user is limited.
	Justification (sample code)	Tomcat's default error pages are overridden. `<error-page>` `  <error-code>400</error-code>` `  <location>/error.html</location>` `</error-page>` `...` `<error-page>` `  <exception-type>java.lang.Exception</exception-type>` `  <location>/error.html</location>` `</error-page>`
11	Vulnerability	**CWE-120:** Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') **CWE-131:** Incorrect Calculation of Buffer Size
	Argument	Buffer overflows not possible in Java programming language.

Checking Basic/Security Requirements

In this subsection, the basic or security requirements specified in the project specification are reviewed. Then, each requirement is tracked in the project design/source code to make sure that it is considered and satisfied.
To make the source code review easier for security inspection, each part of the code that considers a security concern is tagged by **//SWE_681** comments[8].

Basic Requirements

1	Requirement	*When users connect to the game system server, they must be able to either log in or create a new account.*
	Justification	Figure 5 A) Create User (Left), B) Login
2	Requirement	*Users must be able to pick their username and password … you should impose limitations on both. You must not let someone create an account that already exists.*
	Justification	1. Limitation on username and password: `if (!user.getUsername().matches("^([a-zA-Z_0-9]){5,}$"))` `errors.put("username", "user.username.invalid");` `...` `if (!user.getPassword().matches("^(?=.*[A-Z])(?=.*[0-9]).{8,}$"))` `errors.put("password", "user.password.invalid");` For example password should has at least 8 characters, including a capital letter and a number. 2.checking if username already exists. Unique constraint in database: `CREATE TABLE `user` (` `  `username` varchar(45) NOT NULL,` `  `name` varchar(45) NOT NULL,` `  ...` `  UNIQUE KEY `username_UNIQUE` (`username`)` `)`
3	Requirement	*You must use an encrypted channel when transmitting passwords.*
	Justification	Justification of vulnerability 2 in Checking against Common Vulnerabilities.
4	Requirement	*You must store passwords using some salted hash system; you must not store passwords in the clear or as bare hashes.*
	Justification	Justification of vulnerability 6 in Checking against Common Vulnerabilities.
5	Requirement	*Once a user logs in, they must be able to: see the win/loss statistics of themselves and others.*
	Justification	Figure 9 Game statistics
6	Requirement	*Once a user logs in, they must be able to: see game moves of completed games*
	Justification	Figure 10 Game audit trail
7	Requirement	*Once a user logs in, they must be able to: start a new game*

[8] In the XML files, these comments are tagged by **<!-- SWE_681**, and in property files by **!SWE_681**

<table>
<tr><td></td><td>Justification</td><td>Figure 7 A) Create (Left), B) Join Game (Right)</td></tr>
<tr><td>8</td><td>Requirement</td><td>Once a user logs in, join an existing game that needs players</td></tr>
<tr><td></td><td>Justification</td><td>Figure 7 A) Create (Left), B) Join Game (Right)</td></tr>
<tr><td>9</td><td>Requirement</td><td>Once a player has joined a game, they must be able to re-log in and rejoin (e.g., if their computer crashes)</td></tr>
<tr><td></td><td>Justification</td><td>• The current state of each game (i.e. current player, state) and each player (the position of all pieces, the last number of rolled dice) are stored in the database.
• Figure 6 Ludo main page (Resume and leave current game)</td></tr>
<tr><td>10</td><td>Requirement</td><td>If a player leaves, there must be a timeout where eventually they will forfeit.</td></tr>
<tr><td></td><td>Justification</td><td>• 5 minutes is maximum wait time for a user.
• After any player rolls a dice, the server stores the current time. Then, it checks whether any user has left the game.
<pre>long currentTime = Calendar.getInstance().getTime().getTime();
  for (Player player : players) {
        long lastPlayedTime = player.getLastMoveTime() == null ?
currentTime : player.getLastMoveTime().getTime();
        if ((currentTime - lastPlayedTime) > MAX_WAIT_TIME) {
           player.setRank(LEFT);
           player.setFinished(true);
        }
  }</pre>This function is called by every AJAX refresh request, sent form all playing users.</td></tr>
<tr><td>11</td><td>Requirement</td><td>Your program must reject invalid inputs including inputs that are syntactically correct but are semantically invalid based upon the current state of game play.</td></tr>
<tr><td></td><td>Justification</td><td>Before processing every request of players, the validity of the request is checked based on the current state of the game and the players. Here are some samples:
• Rejects if an unauthorized player (who is not in the game) tries to join an started game
<pre>if (!game.isUserAlreadyJoined(user))
        throw new LudoException("You have not joined this game!",
RETURN_PAGE_LIST);</pre>• Make sure if all players are alive and expels the players who have left the game.
<pre>game.checkPlayersAreAlive(ludoService);</pre>• Rejects if the user tries to do any action that is not valid according to the game state, for example only "created" game could be started, or the "started" game could be played.
<pre>if (!Arrays.asList(validState).contains(game.getGameState()))
        throw new LudoException("This action is not valid
according to the game state (" + game.getGameState().name() +
")");</pre>• Rejects if a finished player (who has left or won) tries to make a move</td></tr>
</table>

		`` if (alivePlayerValidAction && game.getPlayerByUser(user).isFinished()) throw new LudoException("You have finished or left the game."); ``` • Rejects if an unauthorized (who is not the game's owner) tries to do some action that is just valid for the game's owner, for example starting the game. ``` if (ownerValidAction && !game.getOwner().getUser().equals(user)) throw new LudoException("Just the games owner is authorized to do this action."); ``` • Rejects if a players tries to inject form fields to play out of his/her turn. ``` if (currentPlayerValidAction && !game.getCurrentPlayer().getUser().equals(user)) throw new LudoException("It is not your turn, please wait!"); ```
12	Requirement	*The game communications and current game state of active games must not be accessible to a third party during play (other than trusted admins). E.G., use SSL/TLS.*
	Justification	Justification of vulnerability 2 in Checking against Common Vulnerabilities.
13	Requirement	*Any games that rely upon random values (e.g. dice, shuffled card deck, etc) will need a source of randomness that is trusted by both players and is actually random.*
	Justification	Dice Number is generated by Java SecureRandom method. ``` public static Integer rollDice() { return new SecureRandom().nextInt(DICE_NUMBER_SIX) + 1; } ```
14	Requirement	*You may use external dependencies… However, it is your job to ensure that there's a relatively low risk of them being security vulnerabilities.*
	Justification	Justification of vulnerability 5 in Checking against Common Vulnerabilities.

Security Requirements

1	Requirement	*Confidentiality: You must create an audit trail of every game move, and these game moves must not be available to those not in the game until the game has completed. After the game is completed, the final audit record (including every move) must be publicly available to all authenticated users (but to not to unauthenticated ones). The win/loss/draw record of each user must be made available to authenticated users (but not to unauthenticated ones).*
	Justification	• Figure 9 Game statistics • Figure 10 Game audit trail • Only the audit trail and statistics of finished games are available.

		`` public List<AuditTrail> getAuditTrailsForGame(...) { String hql = "SELECT DISTINCT at FROM AuditTrail at WHERE at.game.gameState = '" + GameState.FINISHED + "'"; ... ``` • Unauthenticated users do not access to any resource, except creating a new user. (Justification of vulnerability 7 in Checking against Common Vulnerabilities.)
2	Requirement	*Only the current player can make a move in a given game, and it must be a legal move (syntactically and given the current situation).*
	Justification	Justification of Basic Requirement 11 in Basic Requirements.
3	Requirement	*The player must not be able to make any game (including one he's playing) pause forever. A timeout eventually means forfeit, and sending 1 byte should not cause a stall forever.*
	Justification	Justification of Basic Requirement 10 in Basic Requirements.
4	Requirement	*Your system must support multiple simultaneous games by different users*
	Justification	• Any user who has not already created a game or currently playing a game can create a new game.

Static Analysis Tools

As the last security assurance step, Fortify static analysis (in J2EE mode) was run on the whole project to check the potential vulnerabilities, missed in manual inspection phase. The Fortify analysis result indicated that the project has no Critical issue, 3 distinct High priority issues and no medium priority issue. Here is the list of detected High priority issues. [9]

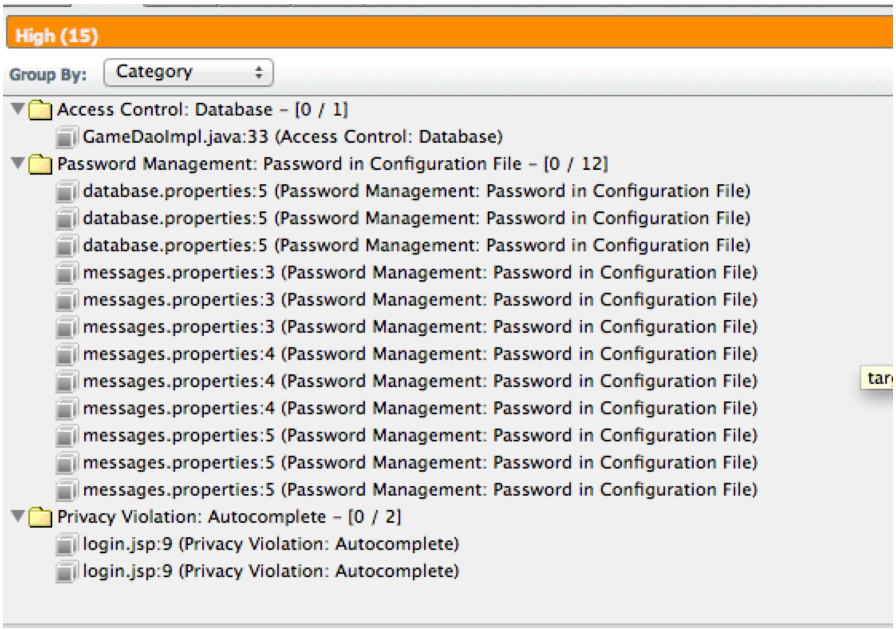


Figure 11 Analysis Results in Fortify Audit Workbench

[9] Some reports are False Positive. For example, the message.properties is the resource bundle message files and all the "password" string in this file are incorrectly reported as vulnerability.

The issued vulnerabilities are fixed. For example, the second one (repeated 3 times!), which is about putting clear database password in the config file, is fixed and addressed in Vulnerability #9 at the section Checking against Common Vulnerabilities).