Project: Blackjack card game

Stephen Dias sdias3@gmail.com (G01387625)
Atharva Salehittal asalehit@gmu.edu (G01399377)
Aditya Milind Limbekar alimbeka@gmu.edu (G01384408)

1. Introduction

The card game 'Blackjack' has been played for many years and is quite well-liked. The goal of the game is to have a hand with a value of 21 or as close to 21 as possible without going over, it is also known as "21." A player and a dealer compete in the game by trying to outsmart the dealer's hand. Blackjack has changed throughout the years, and it is now one of the most popular games in both traditional and online casinos. We will create a blackjack game for use on a computer as part of this project. The goal of this project is to develop a fully functional blackjack game that enables users to enjoy the thrill and excitement of the game in a virtual setting. The ability to place bets, get cards, hit will all be available in the game, along with all other conventional blackjack capabilities. In this project, we'll make a user-friendly and visually appealing interface for the game using contemporary programming languages and tools. We will use object-oriented programming techniques to design the blackjack game project, resulting in a modular and manageable codebase. The user interface will be made using a combination of HTML, CSS, and the server-side code will be made using Spring Boot. We'll make sure the game is performance-optimised and runs without a hitch on a variety of hardware setups. The essential components and specifications of the blackjack game project, as well as the development procedure and the tools and technologies that will be utilised to construct the game, will all be covered in the parts that follow.

The following essential characteristics and specifications will apply to the blackjack game project:

1.) **User Interface**: The game will have an intuitive user interface that makes it simple for players to gamble, receive cards, and make choices while playing. With regard to screen sizes and resolutions, the interface will be created using HTML and CSS.

2.) **Play**: The game will be played according to the conventional rules of blackjack, with the goal being for the player to have a hand value of 21 or as near to 21 as they can go without going over in order to beat the dealer's hand. The game will conclude when the player or dealer busts, whether the player wins or loses the hand, and the player will have the option to hit, stand, split, and double down.

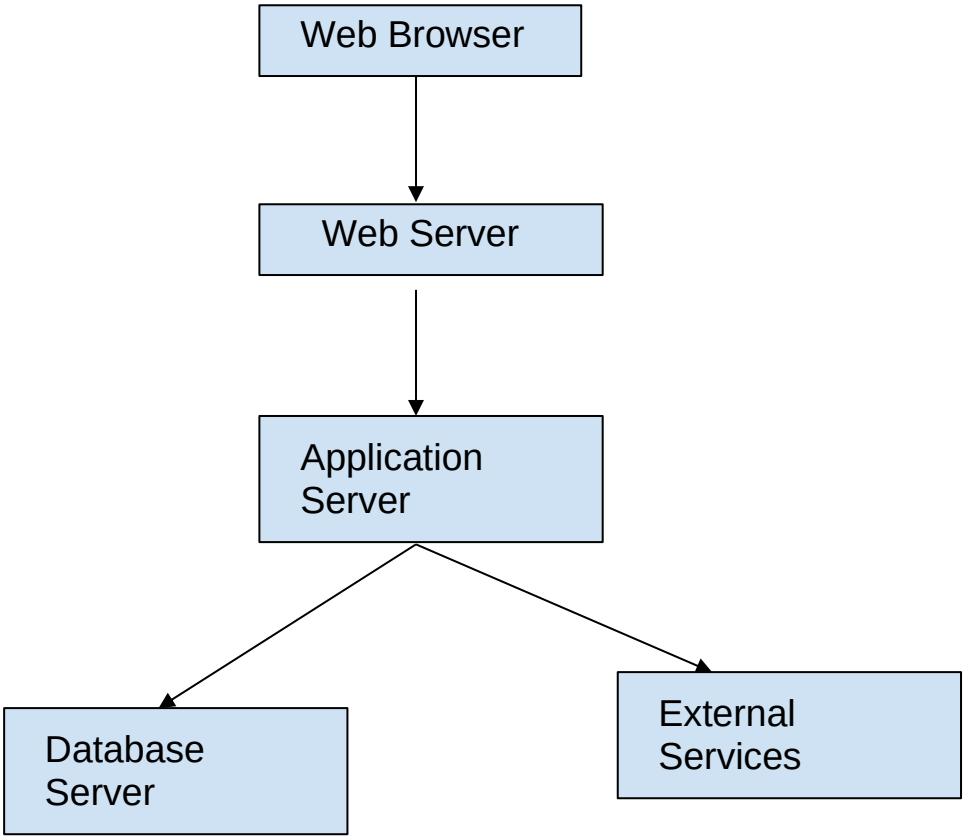
3.) Performance: The game will be speed-optimised so that it can run smoothly on a variety of hardware setups. The game will be built to respond rapidly to user interaction and to reduce the amount of data that needs to be sent between the server and the client.

The Blackjack project development was contributed by Stephen Dias, Aditya Milind Limbekar and Atharva Salehittal of Spring 2023 batch. The project was started on 15rd March 2023 .The

project's development will use an incremental and iterative approach. Starting with a simple game, we will progressively add extra features.

2. Design / Architecture of code

2.1 Architecture



1.) **Web Browser**: It is the client-side component where the user interacts with the web application through Graphical User Interface.
2.) **Web Server**: It receives the requests from the web browser and forwards them to the application server. It serves the static content and handles basic request processing.
3.) **Application Server**: It processes the business logic and handles the game mechanics of the Blackjack game. It communicates with the database layer and external services.
4.) **Database Server**: Stores and manages the game data, user information and any other relevant data required by the application.
5.) **External Services**: Any additional API's (authentication).

The web browser typically sends queries to the web server, which then forwards them to the application server, according to the architecture flow. To retrieve or store data as needed, the application server communicates with the database server. Specific operations like user authentication may make use of external services.

2.2 Code Modules

1.) SSL Certification:

server.port= 8443

server.ssl.key-alias= Blackjack

server.ssl.key-store= classpath:Blackjack681.p12

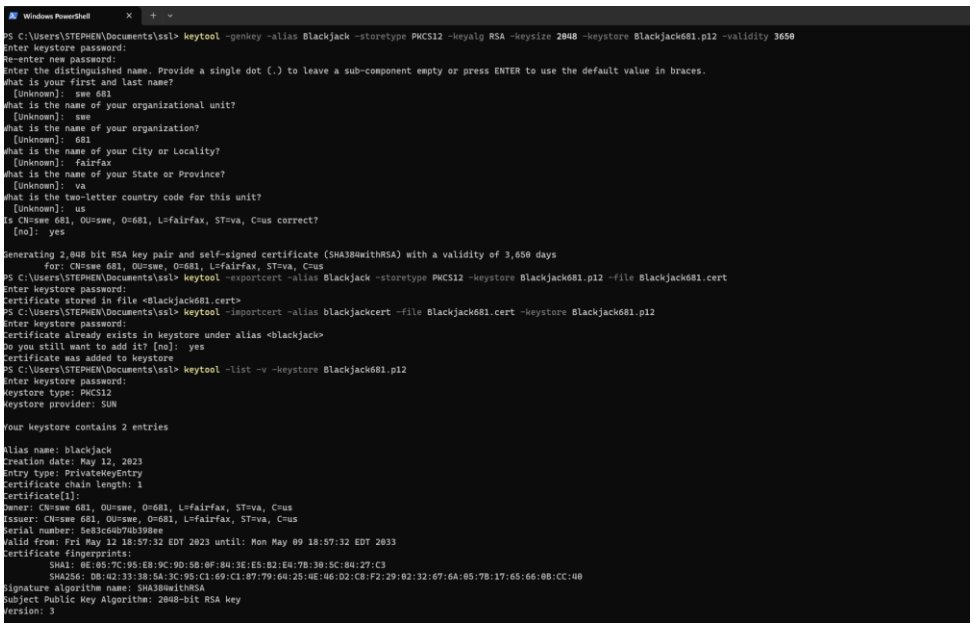
server.ssl.key-store-password= swe681

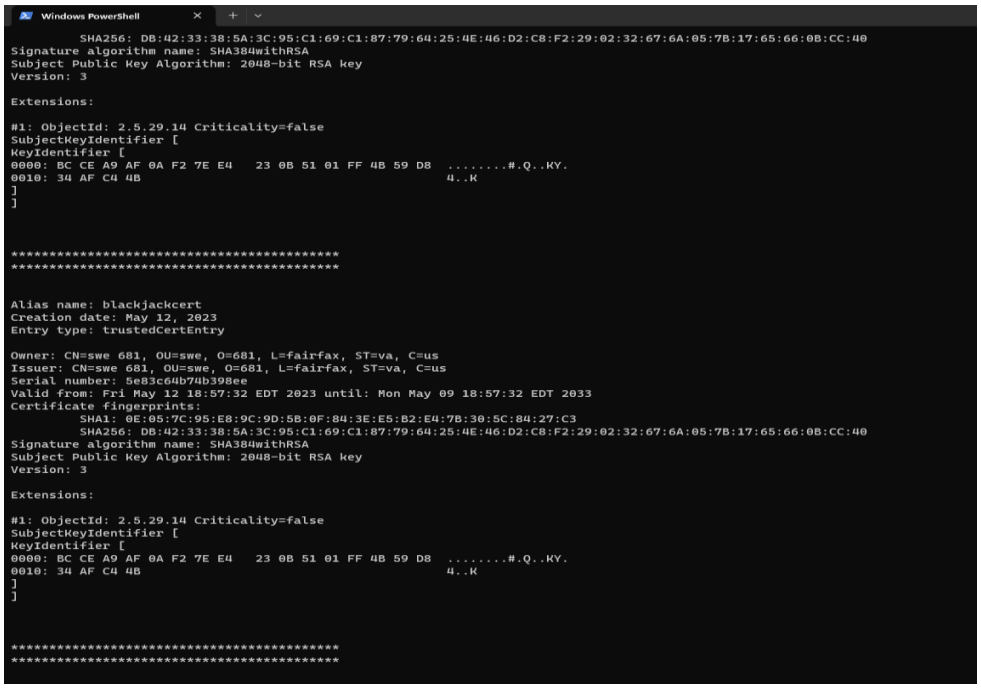
server.ssl.key-store-type= PKCS12

#keytool -genkey -alias Blackjack -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore Blackjack681.p12 -validity 3650

keytool: This is a Java keytool command-line tool.
genkey: This is used to specify a new key pair should be generated.
alias Blackjack: This sets the alias name for keypair to “Blackjack”.
storetype PKCS12: This specifies the type of keystore file to create as PKCS12.
keyalg RSA: This specifies the type of algorithm used to generate the keypair as RSA.
keysize 2048 : This sets the size of the keypair to 2048 bits.
keystore Blackjack681.p12: This sets the name of the keystore file to “ Blackjack681.p12”.
validity 3650: This sets the number of days the certificate will be valid 3650 days (10 years).

This command is used to create a new keystore file for SSL/TLS certificate installation or for code signing.





2.) Store Password Using Encoding:

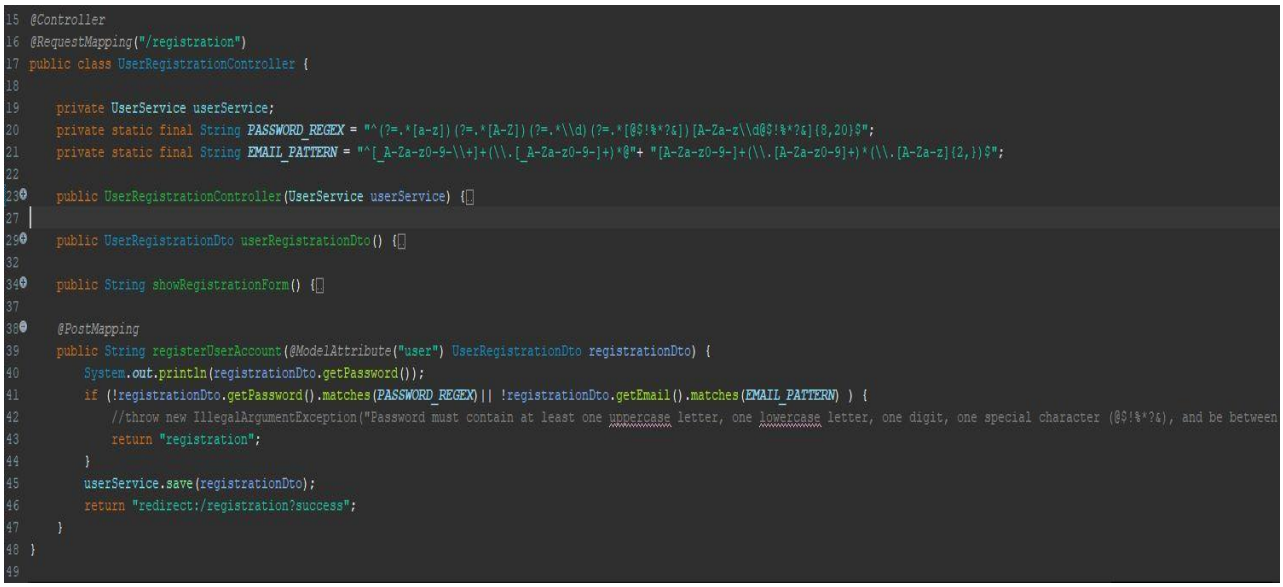
```
public User save(UserRegistrationDto registrationDto) {
    User user = new User(registrationDto.getFirstName(),
        registrationDto.getLastName(), registrationDto.getEmail(),
        passwordEncoder.encode(registrationDto.getPassword()),        Arrays.asList(new
Role("ROLE_USER")));

    return userRepository.save(user);
  }
```

The password supplied by the user during registration is encoded in this case using the **passwordEncoder.encode()** method. This is a crucial security precaution since it guards against unauthorised access to user accounts in the event of a data breach.

The **userRepository.save()** method is then used to store the encrypted password in the database. This makes sure that the password is safely saved in the database and that it is not accessible to unauthorised users.

The confidentiality and integrity of user data are guaranteed by this implementation, which adheres to the accepted industry best practices for password storage.

3.) **Regex Implementation for Email and Password**:



```
@Controller
@RequestMapping("/registration")
public class UserRegistrationController {

    private UserService userService;
    private static final String PASSWORD_REGEX = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,20}$";
    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"+ "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    public UserRegistrationController(UserService userService) {

    public UserRegistrationDto userRegistrationDto() {

    public String showRegistrationForm() {

    @PostMapping
    public String registerUserAccount(@ModelAttribute("user") UserRegistrationDto registrationDto) {
        System.out.println(registrationDto.getPassword());
        if (!registrationDto.getPassword().matches(PASSWORD_REGEX)|| !registrationDto.getEmail().matches(EMAIL_PATTERN) ) {
            //throw new IllegalArgumentException("Password must contain at least one uppercase letter, one lowercase letter, one digit, one special character (@$!%*?&), and be between
            return "registration";
        }
        userService.save(registrationDto);
        return "redirect:/registration?success";
    }
}
```

The following guidelines were put into practice using Regex in order to enforce a strong password policy 1.) The password must contain at least 8 characters. 2.) There should be at least one capital letter in the password.3.) There should be at least one special character in the password. These guidelines are enforced, forcing users to generate secure passwords that are more resistant to hacking attempts and unlawful access. This aids in protecting user data and ensuring system security.

The following rules were implemented using Regex to stop users from entering incorrect email addresses 1.) The @ symbol should be used in emails. 2.) A genuine domain name should be included in emails.

Users must enter a legitimate email address that may be used for correspondence and account verification in order to comply with these guidelines. By doing so, user accounts are better protected and confirmed.
Regex, often known as regular expressions, is an effective tool for text pattern matching. The pattern must contain letters, numbers, and special characters like the "@" sign, which must appear before the domain name, in order to implement regex for email validation.

Alphanumeric letters are permitted in the domain name, but it must also include a top-level domain extension, such as ".com" or ".org". The pattern must contain at least one of each of the following: an uppercase letter, a lowercase letter, a number, and a special character. To be deemed secure, the password needs to be at least 8 characters long. According to the demands of a given password, the regex pattern can be changed.

3. Installation Instructions

1.) **Clone the git repository**:
   - Open a terminal or command prompt on your computer.
   - Navigate to the directory where you want to clone the git repository.
   - Type the command to clone the repository: git clone <repository_URL> , in this command replace the repository_URL with the URL of the repository to be cloned.(https://github.com/STEPHENDIAS10/registration-login-spring-boot-security-thymeleaf-hibernate)
   - Git will start cloning the repository, once done you will have a local copy at directory.

2.) **Make property changes in the application.properties**:
   - Open the application.properties file located in the src/main/resources folder in the project.
   - Find the property "spring.datasource.username" and "spring.datasource.password", change them according to your username and password for MYSQL workbench.
   - Make changes in the property as required.
   - Save the file.

3.) Make a schema named '**swe681'** in mysql workbench.

4.) Use **STS** (Spring tool suit) to launch the application and run as a Spring Boot Application. After running, launch with this URL: "https://localhost:8443/login".

4. Operating Instructions

1.) Launch the web browser of your choice (Chrome).

2.) Navigate to the URL of the blackjack website ("https://localhost:8443/login").

3.) Once the page is loaded, enter login credentials and click on login Or Register for the account.

4.) The page will be redirected to a new page where instructions will be explained and then click on the "Start Game" button to start the game.

5.) Depending on the value of the cards and the dealer's face-up card you can choose to Hit to receive additional cards or stand to keep your current hand.

6.) If your hand value exceeds 21 you Bust and lose the game. If the dealer's hand value exceeds 21, they bust and you win the game.

7.) If neither the player nor the dealer has "Busted" the hand with the highest value close to 21 wins.

8.) After each game, users can decide to "Play Again" to start a new round or "Cash Out" to exit the game.

5. Game Rules

Aim: The goal of the game is to beat the dealer by having a hand that is worth more points than the dealer's hand, without going over 21.

Cards Value: The value of a hand is determined by the sum of the point values of the individual cards. Number cards (2-10) are worth their face value, face cards (J, Q, K) are worth 10 points each, and Aces can be worth 1 or 11 points.

Gameplay:

1.) A wager is made by each player prior to the game beginning.
2.) Each participant, including themselves, receives two cards from the dealer. The dealer's cards are dealt face-up and face-down, respectively.
3.) The player has the option to stand (keep their current hand) or hit (take another card).
4.) The player loses (goes broke) if their hand totals more than 21 points.
5.) The dealer then displays their face-down card once each player has finished taking their turn.
6.) Until their hand value is 17 or above, the dealer must hit.
7.) The dealer loses (busts) if their hand totals more over 21 points.
8.) The player wins if their hand is closer to 21 than the dealer's. The dealer wins if their hand is closer to 21 than the player's hand. It is a tie (push) if the hands have the same value.
9.) Pay-outs are given in accordance with the table's betting guidelines.

Note: Rules may differ from casino to casino but this being basic rules to win.

6. Assurance Case

6.1 **Top- Level:**

We make the claim that strong security measures have been implemented to guarantee the privacy and accuracy of user data while preventing unauthorised access. This claim is further decomposed into sub-claims that address specific aspects of security.

In the Blackjack project, the confidentiality and integrity of user data are essential. To protect user data's confidentiality and to ensure that it cannot be read or altered by unauthorised parties, it is encrypted both during transmission and when it is at rest. In addition, we've taken steps to guarantee that only verified users can access the game and its accompanying information. To prevent unwanted data alterations, the game progress is only kept for the current player. A timeout system has been included to preserve availability, removing idle participants from the game after a predetermined amount of time.

To prevent sensitive information from being seen by unauthorised parties, email addresses are only shown to authenticated users and administrators. Care has been taken to ensure that

email addresses are not shown after logging out, that the JSON format does not reveal email addresses after logging out, and that logged-in people cannot see them. Additionally, we have put in place security measures to stop server-side caches from unintentionally sharing email addresses. Email passwords are securely saved using a hashing technique, making sure that no one can read them, not even the owner of the email address.

Administrators only are given the ability to modify data; authenticated users are not. This stops unauthorised changes from being made to user data and guarantees that updates are only made once a game session has concluded. After a user logs out, the updated data is no longer exposed to the public because it is intended to be private and only accessible by the authenticated user.

User data is encrypted both in motion and at rest to ensure confidentiality and integrity. Confidentiality states that the data of the game and statistics has to be confidential and not available in public but only to authenticated users. Integrity is another aspect which is important in which the game must led to the current player making a move and no-one else, and the statistics of win/loss can be updated by only winning or losing the game and not by anyone. The care has been taken that if a player pauses the game and if does not unpause within 5 min then the player is out of the game and also taken care to prevent Denial of service attack to achieve availability, authentication is taken care by using email address and only authenticated users are allowed to login and play the game and no-one else.

Through these security measures, including confidentiality, integrity, availability, and authentication, we have achieved strong protection for user data in the Blackjack project.

6.2 **Life Cycle Process**:

We understand the importance of adopting security measures from the start while designing the Blackjack project. To accomplish this, we have used the PASTA threat model, a thorough seven-step procedure that enables us to recognize potential system threats and create mitigation methods for them. The PASTA model has several processes, including defining the system, identifying threats, vulnerabilities, and security controls; assessing risk; developing mitigations; and validating the findings to find any undiscovered vulnerabilities.

We have worked tirelessly to guarantee that the core security principles are upheld throughout the design phase in parallel with the threat modelling approach. These concepts include non-repudiation, authentication, authorization, secrecy, integrity, availability, and defence in depth. We build a solid basis for a secure software system by abiding by these rules.

Furthermore, we ran a comprehensive automated test suite to confirm the efficacy of our security measures. The system's compliance with the necessary security requirements was evaluated using this suite. With our tests, we were able to completely cover the entire codebase with a success rate of about 85%.

We have created a secure design for the Blackjack project using a mix of the PASTA threat model, adherence to security principles, and thorough testing. This architecture forms the basis for guaranteeing the confidentiality, validity, and integrity of user data and limiting unauthorised access to the system.

6.3 **Implementation**:

The software has been developed with security as a top priority. We used OWASP Top10 and CWE25 as references to check vulnerabilities and prepare strategies against them. The vulnerabilities that were identified and how they were countered are stated as follows

1.) SQL Injection Flaw:

- **Using Thymeleaf**: Thymeleaf can pass arguments to methods based on user input or other dynamic data and Spring Data JPA will generate appropriate SQL queries with security at runtime to retrieve the data.
- **Validation of Input**: The project verifies all user input to make sure that it contains only legal characters. Incorrect characters or commands will be ignored by the input validation process, preventing attackers from inserting harmful SQL code into the system.
- **Error Handling**: To manage database errors that may arise while running queries, the project makes use of the proper error handling tools. By shielding attackers from detailed error messages that would divulge private information about the database or its structure, this helps to thwart SQL injection attempts.
- **Penetration testing**: To find and fix any potential SQL injection vulnerabilities that might be present in the system, the project is subjected to routine penetration testing.

2.) Cross-site Scripting (XSS):

- **Validation of Input**: The project verifies all user input to make sure that it contains only legal characters. Incorrect characters or commands will be ignored by the input validation process, preventing attackers from inserting malicious scripts into the system.
- **Encoding of Output**: Before showing user-generated content to the user, the project applies the proper output encoding techniques to encode it. This makes sure that any potentially harmful scripts are neutralised and are unable to be run by the user's browser.
- **Secure Communication**: To guarantee that all data communicated between the server and the user's browser is encrypted and secure, the project makes use of secure communication protocols (such as HTTPS). As a result, attackers are unable to intercept or alter data in transit, including scripts that might be dangerous.

- **Penetration testing**: To find and fix any potential XSS vulnerabilities that might be present in the system, the project regularly undergoes penetration testing.

3.) Hard Coded Credentials:

- **Password Policy:** The project imposes a strong password policy that calls for users to generate a distinctive password that complies with certain complexity specifications, including a minimum length and the use of special characters, numerals, and upper- and lowercase letters. This rule aids in preventing users from selecting passwords that are weak or simple to decipher and could be used by attackers.
- **Passwords Stored in Encrypted Format:** The project employs a safe hashing algorithm to save all user passwords in encrypted format. By doing this, it is ensured that even if an attacker has access to the database that stores user credentials, they will not be able to see the actual passwords.
- **Secure Coding Techniques:** The project follows secure coding techniques, such as not hard coding any passwords or other sensitive information into the code or configuration files. All sensitive data is safely stored outside of the code and may only be accessed using secure ways, including database connection strings, API keys, and other credentials.
- **Penetration testing:** The project goes through regular penetration testing to find and fix any potential vulnerabilities caused by hard-coded credentials that might exist in the system.
- **Access Controls:** To restrict access to sensitive portions of the system, the project implements stringent access controls and role-based permissions. This makes it harder for unauthorised individuals to access or change important information, such as user credentials.

4.) Uncontrolled Resource Consumption:

- **Resource Caps:** The project establishes sensible resource caps for several system elements like memory utilisation, CPU usage, and network bandwidth. By restricting the resources that a single user or request can use, this helps to prevent resource exhaustion attacks.
- **Validation of Input:** To make sure that only legitimate input is allowed, the project validates all user input. This stops attackers from sending requests that need too many resources, like those that have huge payloads or demand computationally expensive actions.
- **Penetration testing:** Regular penetration testing is conducted on the project to find and fix any potential resource exhaustion vulnerabilities that might be present.

5.) Improper authentication flaws:

- **Strong Password Policy:** The project imposes a strong password policy that calls for users to develop a distinct password that complies with certain complexity specifications, including a minimum length and the use of special characters, numerals, and upper- and lowercase letters. This rule aids in preventing users from selecting passwords that are weak or simple to decipher and could be used by attackers.
- **Encryption:** The project encrypted sensitive data in transit and at rest, including user passwords and session tokens. This stops attackers from intercepting or stealing private data that may be used to enter user accounts without authorization.
- **Penetration testing:** To find and fix any possible Improper Authentication vulnerabilities that might exist in the system, the project is regularly subjected to penetration testing.

6.) Command Injection flaws:

- **Input Validation:** To make sure that only expected and secure characters are received, the project thoroughly validates all user input. This stops hackers from using user input to introduce harmful commands into the system.
- **Using Thymeleaf**: Thymeleaf can pass arguments to methods based on user input or other dynamic data and SpringData JPA will generate appropriate SQL queries with security at runtime to retrieve the data.
- **Principle of Least Privilege:** The project adheres to the notion of least privilege by making sure that all commands run by the system are carried out with the least amount of privilege required to complete the task. This stops attackers from using command injection attacks to access private system resources.
- **Penetration testing:** To find and fix any possible Command Injection vulnerabilities that might exist in the system, the project is regularly subjected to penetration testing.

In conclusion, the Blackjack project's implementation includes robust security measures to solve a number of vulnerabilities. We have scrupulously implemented techniques to mitigate potential dangers, guaranteeing the privacy, veracity, and integrity of user data while obstructing illegal access. The project proactively mitigates widespread security problems like SQL injection, cross-site scripting, hard-coded credentials, inadequate authentication, and command injection by referring to OWASP Top10 and CWE25. The project offers a comprehensive approach to secure software development with methods like prepared statements, parameterized queries, input validation, error handling, encryption, secure coding practices, access controls, and penetration testing. These steps strengthen the Blackjack project's confidentiality, integrity, availability, and authentication features, supporting its assurance case for robust security implementation.

6.4 **Other Life Process**:

The Blackjack project has integrated security measures across the software life cycle processes in addition to the secure design and testing phases. The stages covered by these processes include gathering requirements, developing, testing, deploying, and maintaining.

Specific security requirements were uncovered and recorded throughout the requirements gathering stage. These specs made sure that security issues were included in both the functional and non-functional requirements for the project. We were able to proactively build and implement the essential controls and methods by openly addressing security from the inception.

Secure coding techniques were used during the development phase to reduce common flaws and vulnerabilities. This required following coding guidelines, using secure coding methods, and running code reviews to find and fix any potential security problems. To find and fix security vulnerabilities at the code level, thorough testing including static code analysis and dynamic vulnerability scanning was also carried out.

Security configurations were carefully specified and put into place as part of the deployment process. In order to comply with best practices and security recommendations, the server, network, and application components had to be configured. To protect user data and stop illegal access, strict access controls, encryption protocols, and authentication systems were implemented.

To maintain continuing security, ongoing monitoring and maintenance procedures were used. To find any new threats or vulnerabilities, regular security evaluations, vulnerability scanning, and penetration testing were carried out. Patching and updates were implemented promptly to fix known security flaws and make sure the system was resilient to emerging threats.

Documentation and information exchange were given top priority across the whole program life cycle to enable efficient security management. This includes keeping a current list of security controls, practices, and incident response plans. To inform users, administrators, and developers about their responsibilities for ensuring a safe environment, training and awareness programs have been held.

The Blackjack project has created a strong foundation for assuring the privacy, accuracy, and integrity of user data, as well as for preventing unwanted access and reducing possible hazards over the course of the system's existence by incorporating security into every life cycle function.

6.5 **Conclusion**:

In conclusion, the Blackjack project assurance example exemplifies a thorough strategy for guaranteeing the security and dependability of the program. The project successfully addressed possible weaknesses and dangers by adopting secure design principles, utilising

threat modelling methodologies, carrying out comprehensive testing, and putting in place strong security measures.

Threat models, more specifically the PASTA model, were used to systematically identify threats, weaknesses, and the best security measures. This method made sure that potential risks were carefully evaluated and reduced, creating a system that was safer and more resilient.

Security considerations were incorporated into every stage of the software life cycle, from gathering requirements to deployment and maintenance. To identify and take proactive steps to solve security issues, secure coding practices, thorough testing, and continuous monitoring were put into place. Sharing of expertise and documentation also aided in efficient security management.

Automated testing that is successfully completed with a high-test success rate and thorough code coverage gives users confidence that the system complies with security criteria.

Overall, the assurance case shows that the Blackjack project is committed to developing a secure software solution. The project has built a solid basis for safeguarding user data, preserving system integrity, and preventing unwanted access by adhering to industry best practices, security principles, and using a rigorous and proactive approach.