SWE 681 Secure Software Design and Programming, Spring 2014

Dr. David A. Wheeler

Term Project Report

Secure Blackjack

Submission date - 7th May 2014

Gentian Cala | Shaeq Khan

Introduction

Blackjack or 21 is the most popular table game offered in gambling establishments. We decided to implement a web app that would allow its users to join a table and play the game along with others. A table can have up to 7 players at the same time (muti-player, synchronous). Players can -

- Create an account
- Login to start or join a game
- Bet before the start of each round in a game
- Surrender, Hit or double in the game
- View win-loss statistics of themselves and other registered users
- View game moves of completed games
- Continue a game if it hasn't timed out

The focus in this project was to make this application reasonably secure to convince users that the app can be trusted with confidentiality, integrity and availability.

In this game, players play against the dealer and bet money. The basic premise of the game is that you want to have a hand value that is closer to 21 than that of the dealer, without going over 21. Other players at the table are of no concern and your hand is strictly played out against the hand of the dealer. It's not a problem if the dealer or any of the other players at the table see the cards in your hand.

Game Rules

In blackjack, the cards are valued as follows:

- An Ace can count as either 1 or 11.
- The cards from 2 through 9 are valued at their face value.
- The 10, Jack, Queen, and King are all valued at 10.

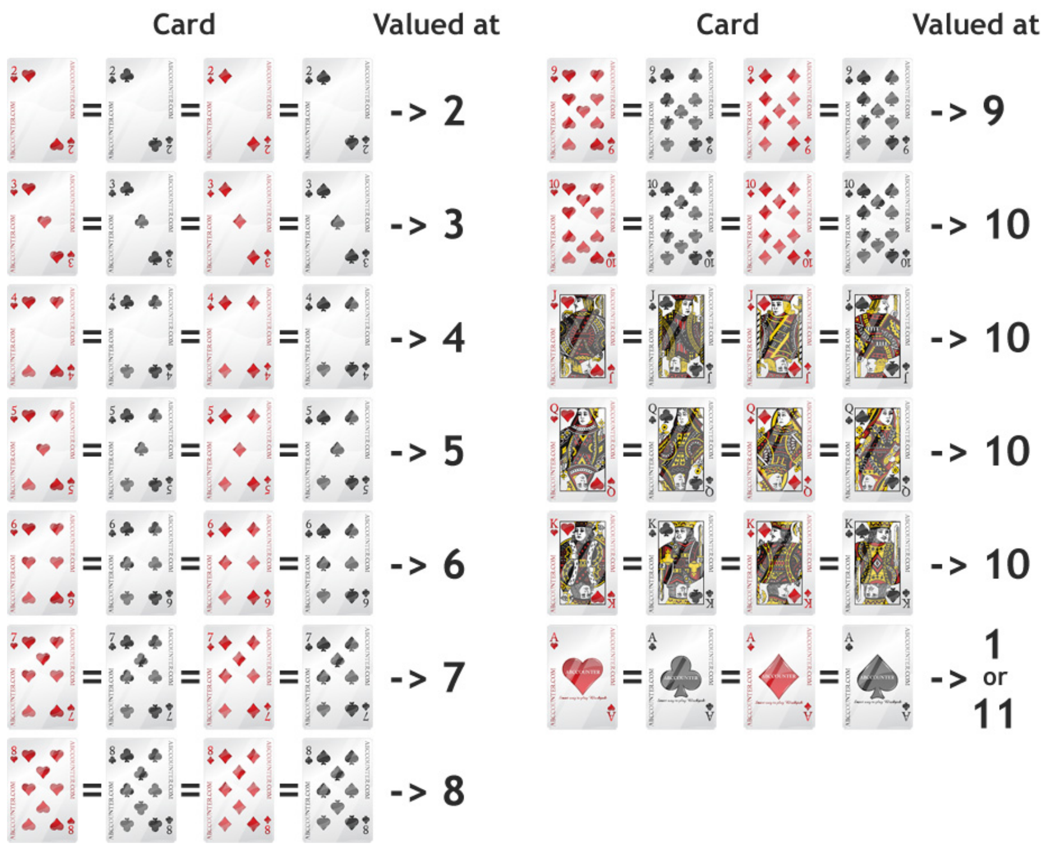

Card
Valued at
Card
Valued at
-> 2
-> 3
-> 4
-> 5
-> 6
-> 7
-> 8
-> 9
-> 10
-> 10
-> 10
-> 10
-> 1 or 11

Figure 1 - Card Values in Blackjack[source]

The suits of the cards do not have any meaning in the game. The value of a hand is the sum of the point counts of each card in the hand. For example, a hand containing (5,7,9) has the value of 21. The Ace can be counted as either 1 or 11. You need not specify which value the Ace has. It's assumed to always have the value that makes the best hand.

An example - Suppose that you have the beginning hand (Ace, 6). This hand can be either 7 or 17. If you stop there, it will be 17. Let's assume that you draw another card to the hand and now have (Ace, 6, 3). Your total hand is now 20, counting the Ace as 11. Let's backtrack and assume that you had instead drawn a third card which was an 8. The hand is now (Ace, 6, 8) which totals 15. Notice that now the Ace must be counted as only 1 to avoid going over 21.

A hand that contains an Ace is called a "soft" total if the Ace can be counted as either 1 or 11 without the total going over 21. For example (Ace, 6) is a soft 17. The description stems from the fact that the player can always draw another card to a soft total with no danger of "busting" by going over 21.

The hand (Ace,6,10) on the other hand is a "hard" 17, since now the Ace must be counted as only 1, again because counting it as 11 would make the hand go over 21.

When all the bets are made, the dealer will deal the cards to the players. Once the cards are dealt, play proceeds around the table starting at the first seat to the dealer's left, also called first base. Each player in turn indicates to the dealer how they wish to play the hand. After each player has finished their hand, the dealer will complete his hand, and then pay or collect the player bets.

A blackjack, or natural, is a total of 21 in your first two cards. A blackjack is therefore an Ace and any ten-valued card, with the additional requirement that these be your first two cards. If you split a pair of Aces for example, and then draw a ten-valued card on one of the Aces, this is not a blackjack, but rather a total of 21. A player blackjack beats any dealer total other than blackjack, including a dealer's three or more card 21. If both a player and the dealer have blackjack, the hand is a tie or push.

Player Choices

Surrender

It is one of the least common decisions and must be made before any other choice about playing your hand. Surrender offers you as a player the choice to fold your hand, at the cost of half of the original bet. You must make that decision prior to taking any other action on the hand. For example, once you draw a third card, or split, or double down, surrender is no longer an option.

Hitting/Standing

The most common decision a player must make during the game is whether to draw another card to the hand ("hit"), or stop at the current total ("stand").

Doubling Down

Among the more profitable player options available is the choice to "double down". This can only be done with a two card hand, before another card has been drawn. Doubling down allows you to double your bet and receive one, and only one, additional card to the hand. A good example of a doubling opportunity is when you hold a total of 11, say a (6,5) against a dealer's upcard of 5. In this case, you have a good chance of winning the hand by drawing one additional card, so you might as well increase your bet in this advantageous situation.

Design and Architecture

This web application was completed using the following technologies -

Adobe ColdFusion 11

This is a commercial web app development platform that makes it easier to connect simple HTML pages to a database. This platform uses the ColdFusion Markup Language CFML (compares to scripting components of JSP, PHP, ASP) which has a tag syntax similar to HTML and the script syntax resembles JavaScript.

The standard ColdFusion installation allows the deployment of ColdFusion as a WAR or EAR file for deployment to standalone application servers like Macromedia JRun and IBM WebSphere. ColdFusion can also be deployed to servlet containers like Apache Tomcat but because these platforms do not officially support ColdFusion, they leave many of its features inaccessible.

Because ColdFusion is a Java EE application, ColdFusion code can be mixed with Java classes to create a variety of applications and use existing Java libraries. ColdFusion has access to all underlying Java classes, supports JSP custom tag libraries, and can access JSP functions after retrieving the JSP page context

MS SQL Server 2014

MS SQL Server is a relational database management system developed by Microsoft. As a database, it is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet).

Adobe Dreamweaver

Adobe Dreamweaver is a web design and development application that provides a visual WYSIWYG editor and a code editor with standard features such as syntax highlighting, code completion, and code collapsing as well as more sophisticated features such as real-time syntax checking and code introspection for generating code hints to assist the user in writing code. The Design view facilitates rapid layout design and code generation as it allows users to quickly create and manipulate the layout of HTML elements. Dreamweaver features an integrated browser for previewing developed web pages in the program's own preview pane in addition to allowing content to be open in locally installed web browsers.

Wiring Diagram

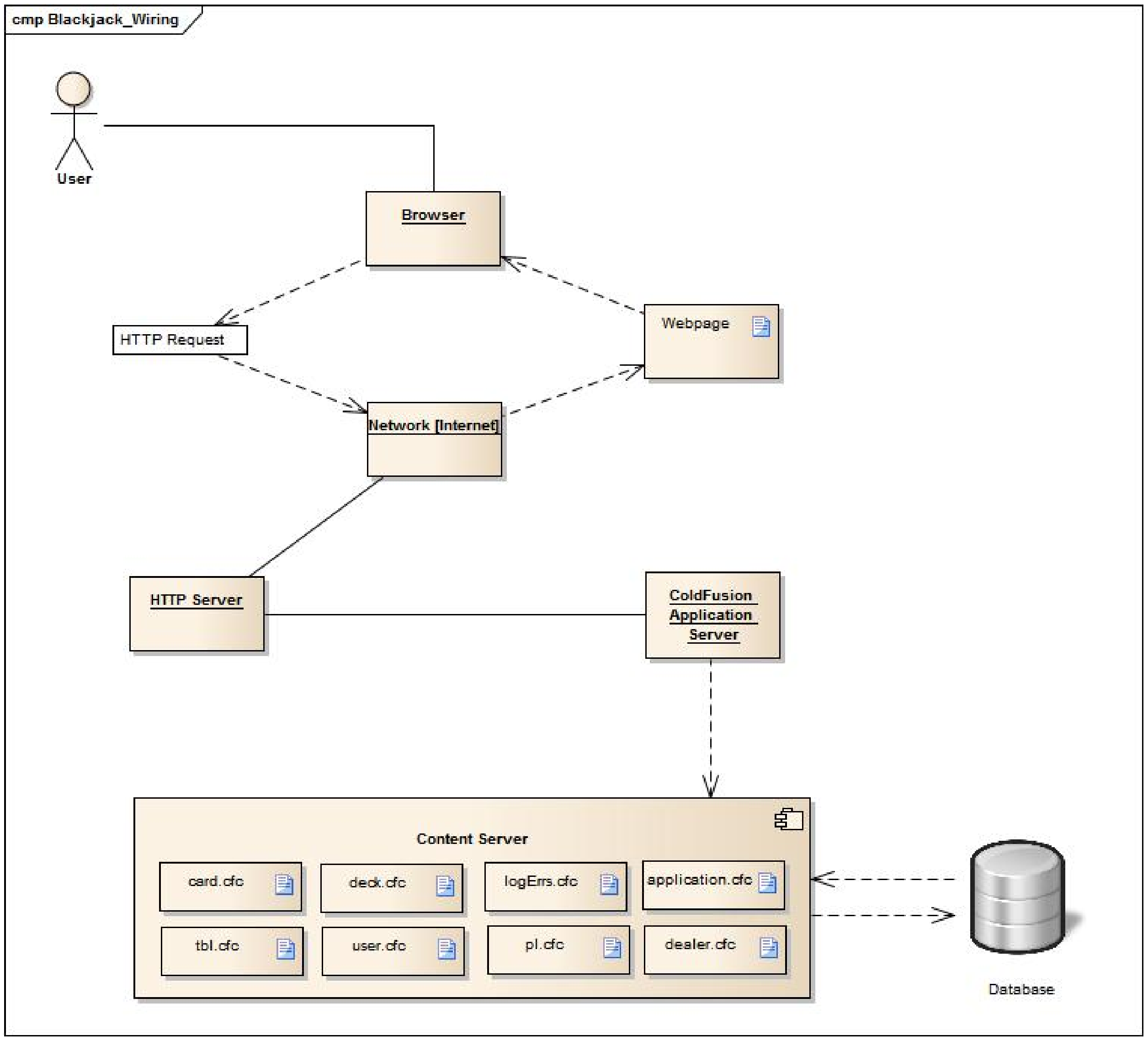


Figure 2 - Wiring Diagram

ER Diagram

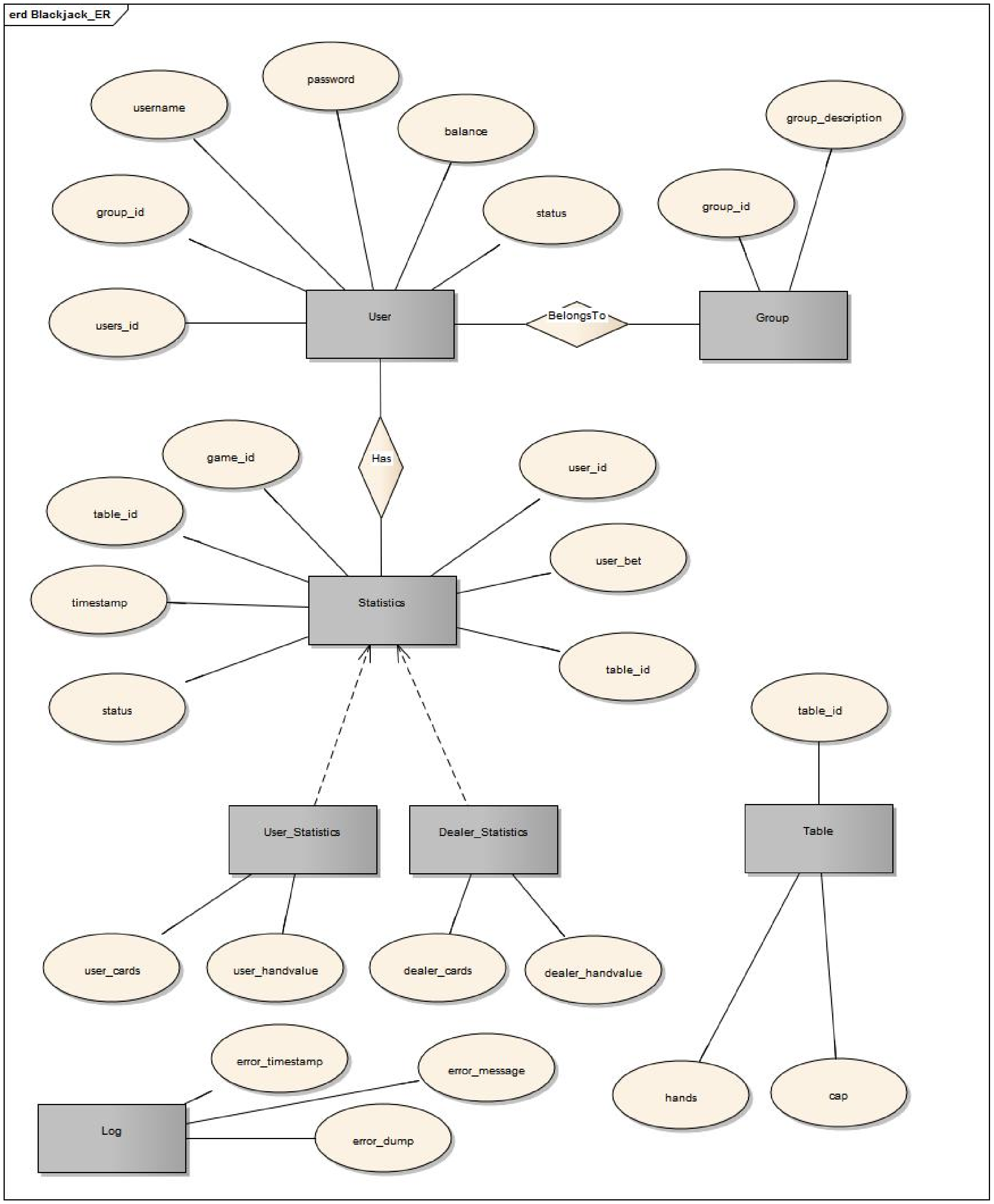


Figure 3 - ER Diagram

Installation Instructions

- You will need to install the ColdFusion 10 or 11 Server (trial version available for a month on submitting a snapshot of your gmu id via the adobe website). Refer to the documentation here to walk you through the installation process.

- Install the MS SQL 2014. Refer to documentation here to walk you through the installation process.

- Restore the database file by importing it on the MS SQL server.

- Follow this tutorial to create a DSN (Data Source Name) on the ColdFusion server. Before you can connect to a database in Dreamweaver, you must create a ColdFusion data source in ColdFusion Administrator, the server's management console.

- Deploy (copy/paste) the coldfusion files on the web server to run the web application.

Operating Instructions

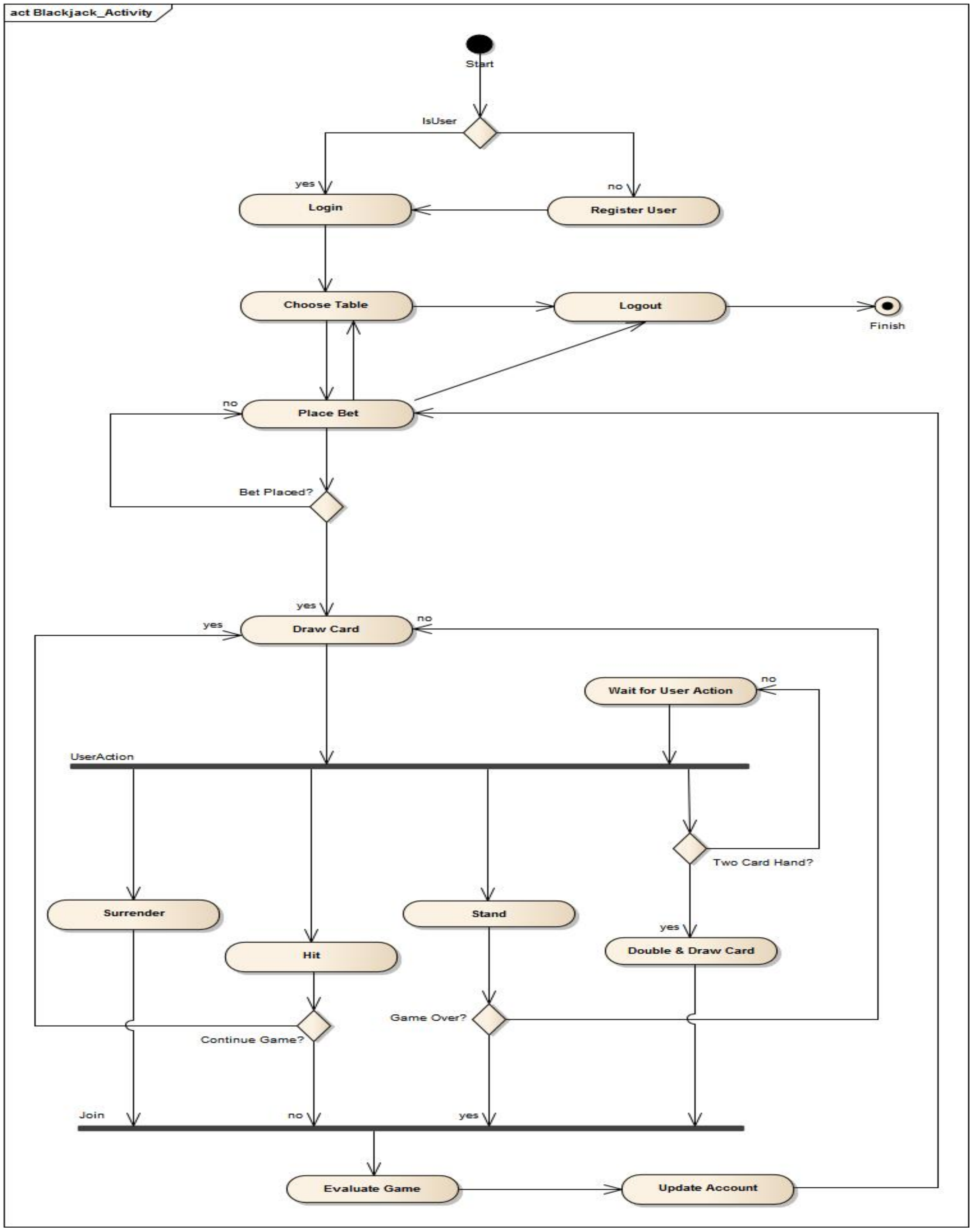


Figure 4 - User Action Flowchart

Why is this application secure?

The following *CIA* security requirements have been taken care of in this application -

- *Confidentiality* - every game move is audited, game moves are not available for view until the game is completed and the final audit is available to all authorized users.
- *Integrity* - only current players are allowed to make moves in the current game.
- *Availability* - users cannot pause the game forever, the game is forfeited if timed out to prevent DoS attacks.

Assurance Case

Claim 1 - This app is secure against low and moderate vulnerabilities

Claim 1.a - User Information is secure

Claim 1.b - Input Values are sanitized

Claim 1.c - Game state is always consistent

Claim 1.d - The application design is secure

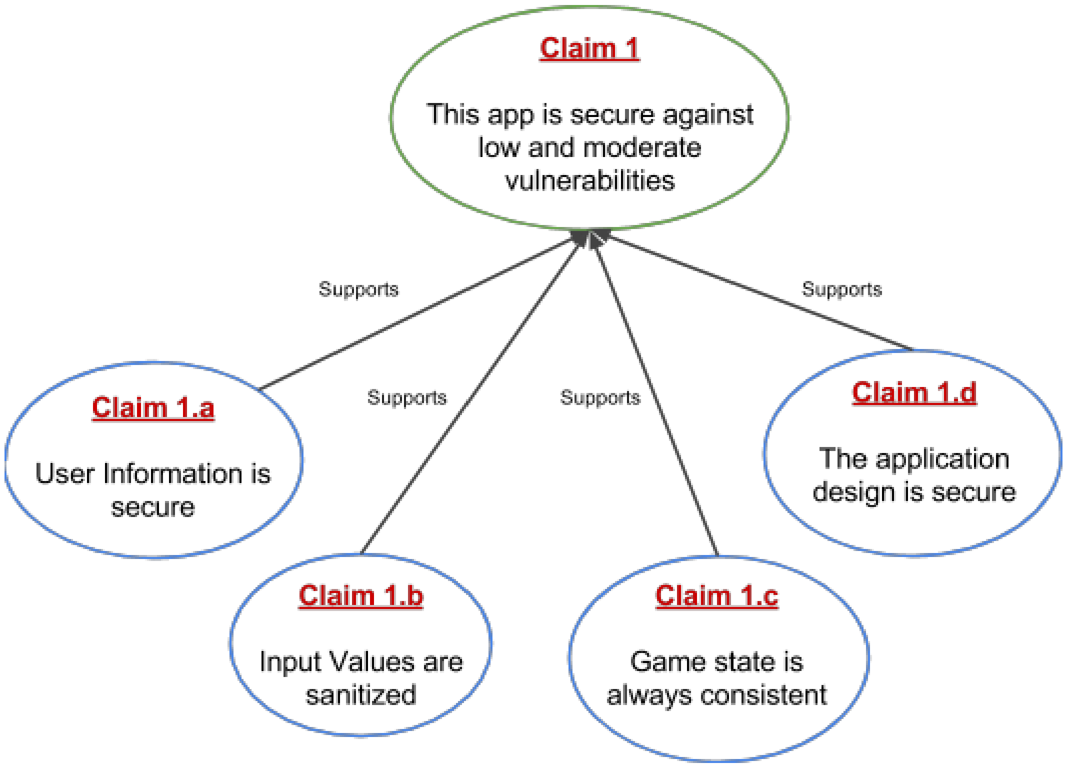

Claim 1
This app is secure against low and moderate vulnerabilities
Supports
Supports
Supports
Supports
Claim 1.a
User Information is secure
Claim 1.b
Input Values are sanitized
Claim 1.c
Game state is always consistent
Claim 1.d
The application design is secure

Figure 5 - Overall view of Assurance Case following the Claim-Argument-Evidence model

Claim 1.a - User Information is secure

Argument 1.a.i - Use of SSL for transmitting passwords over encrypted channels
Evidence - Detailed steps followed to ensure SSL is enabled in the deployment descriptor for the entire application and it is run on the server with information sent over the network between the client and server using hypertext transfer protocol secure (https).

Argument 1.a.ii - Using salted hashes to store passwords
Evidence - use of java.security.SecureRandom library to generate a Cryptographically Secure Pseudo-Random Number for a user salt. A unique salt value of 128 bits as a String generated based on the SHA1PRNG algorithm which is used as a cryptographically strong pseudo-random number generator based on the SHA-1 message digest algorithm.
A unique salt is generated on the server per every user and password without re-using a salt. The salt is prepended to the password and is hashed with a cryptographic hash function. The password is generated using the SHA-256 hashing algorithm built into the java.security library. Both the salt and hash are stored in the database.

Argument 1.a.iii - User authentication on server
Evidence - For validation, a user's salt and hash is retrieved and the salt is prepended to the entered password and hashed. The result is compared to the hashed value stored in database and validated.

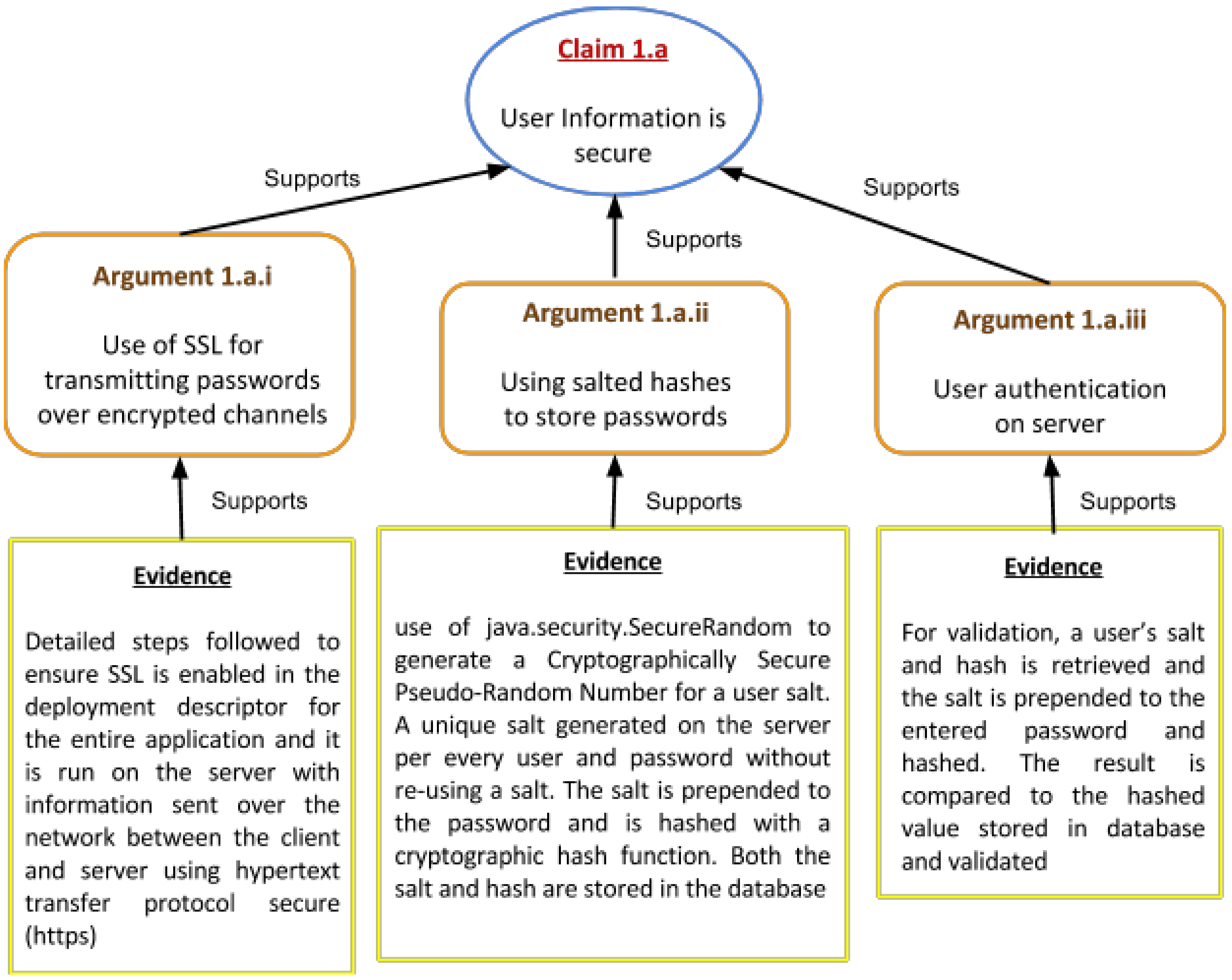

Claim 1.a
User Information is secure
Supports
Supports
Supports
Argument 1.a.i
Use of SSL for transmitting passwords over encrypted channels
Argument 1.a.ii
Using salted hashes to store passwords
Argument 1.a.iii
User authentication on server
Supports
Supports
Supports
Evidence
Detailed steps followed to ensure SSL is enabled in the deployment descriptor for the entire application and it is run on the server with information sent over the network between the client and server using hypertext transfer protocol secure (https)
Evidence
use of java.security.SecureRandom to generate a Cryptographically Secure Pseudo-Random Number for a user salt. A unique salt generated on the server per every user and password without re-using a salt. The salt is prepended to the password and is hashed with a cryptographic hash function. Both the salt and hash are stored in the database
Evidence
For validation, a user's salt and hash is retrieved and the salt is prepended to the entered password and hashed. The result is compared to the hashed value stored in database and validated

Figure 6 - Claim 1.a, arguments and evidence

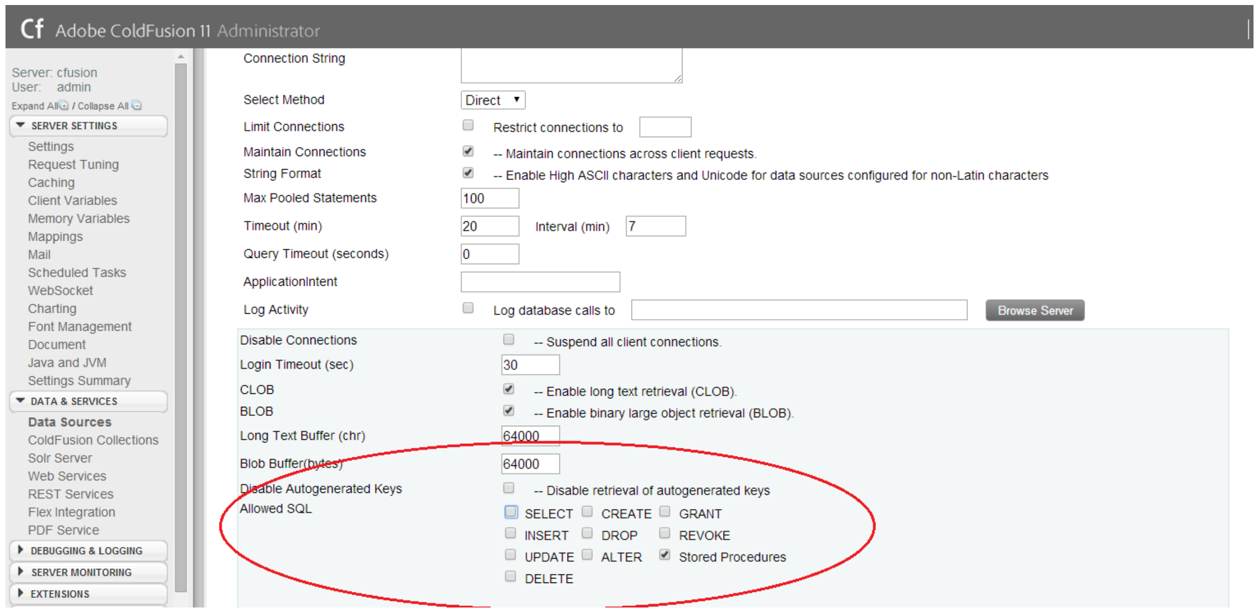
Claim 1.b - Input Values are sanitized

Argument 1.b.i - Input Validation for all possible user inputs

Evidence - Use of java.util.regex library to declare a regular expression to validate user inputs for placing bets, entering a username and password for registering account and every user action which is validated on the server. Even if you write your own front-end to use the modules on Server, every user input and action will be validated at the server.

Argument 1.b.ii - Use of Stored Procedures to avoid SQL injection attacks

Evidence - Stored Procedures are produced in MS SQL Server and called from the ColdFusion middleware. This reduces the attack surface area of the database by communicating only through these "Procedures" instead of any SQL statements. Any other SQL statements hard-coded do not execute from ColdFusion middleware which helps in preventing SQL Injections.



Connection String	
Select Method	Direct
Limit Connections	☐ Restrict connections to
Maintain Connections	✓ -- Maintain connections across client requests.
String Format	✓ -- Enable High ASCII characters and Unicode for data sources configured for non-Latin characters
Max Pooled Statements	100
Timeout (min)	20 Interval (min) 7
Query Timeout (seconds)	0
ApplicationIntent	
Log Activity	☐ Log database calls to
Disable Connections	☐ -- Suspend all client connections.
Login Timeout (sec)	30
CLOB	✓ -- Enable long text retrieval (CLOB).
BLOB	✓ -- Enable binary large object retrieval (BLOB).
Long Text Buffer (chr)	64000
Blob Buffer(bytes)	64000
Disable Autogenerated Keys	☐ -- Disable retrieval of autogenerated keys
Allowed SQL	☐ SELECT ☐ CREATE ☐ GRANT ☐ INSERT ☐ DROP ☐ REVOKE ☐ UPDATE ☐ ALTER ✓ Stored Procedures ☐ DELETE

Argument 1.b.iii - Designed with minimal surface attack

Evidence - User has a limited source of input to the application. We were successfully able to implement the concept of security in depth such that every layer validates the information exchanged between them. This helped to bottle down and rigorously test the surface area for attacks.

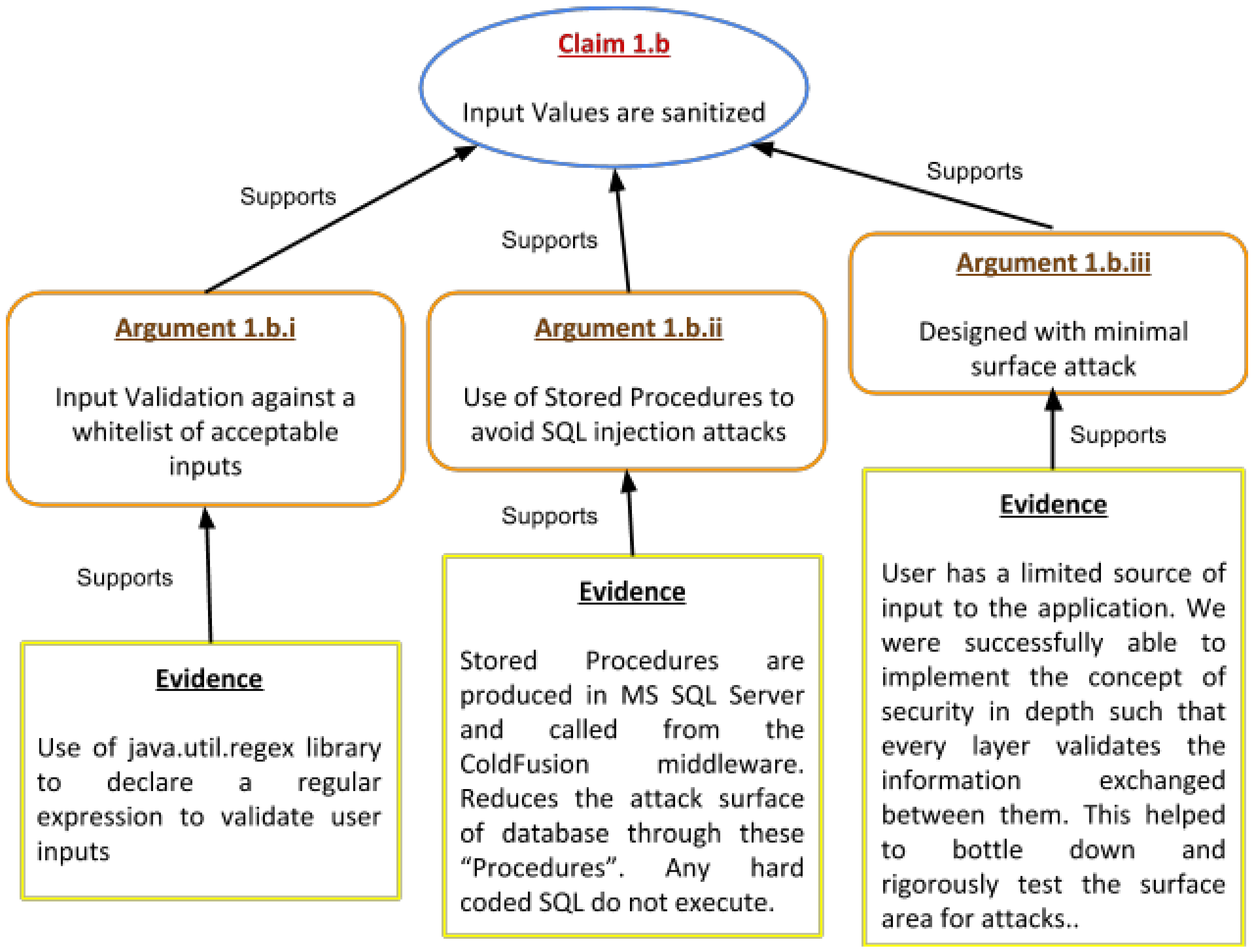

Claim 1.b
Input Values are sanitized
Supports
Supports
Supports
Argument 1.b.i
Input Validation against a whitelist of acceptable inputs
Argument 1.b.ii
Use of Stored Procedures to avoid SQL injection attacks
Argument 1.b.iii
Designed with minimal surface attack
Supports
Supports
Supports
Evidence
Use of java.util.regex library to declare a regular expression to validate user inputs
Evidence
Stored Procedures are produced in MS SQL Server and called from the ColdFusion middleware. Reduces the attack surface of database through these “Procedures”. Any hard coded SQL do not execute.
Evidence
User has a limited source of input to the application. We were successfully able to implement the concept of security in depth such that every layer validates the information exchanged between them. This helped to bottle down and rigorously test the surface area for attacks..

Figure 7 - Claim 1.b, arguments and evidence

Claim 1.c - Game state is always consistent

Argument 1.c.i - Client-Server Architecture
Evidence - Shifting security to the server side provides better security for the application. Servers have better control access and resources to ensure that only authorized clients can access or manipulate data and server-updates are administered effectively.

Argument 1.c.ii - Game state controlled at server
Evidence - Use of session ids to keep track of data coming in to the server between multiple requests from different users.

Argument 1.c.iii - Session management via session ids
Evidence - session ids are random, unpredictable and stored as cookies on client side and sent over to the server using SSL.

Argument 1.c.iv - Game timer implemented to prevent DoS attacks
Evidence - A timer is associated with every session and after a reasonable amount of time (design decisions), the game session is invalidated. Users cannot lock the game. The following behaviours are incorporated in the web application -

- The server waits for 30 seconds for a user move and then continues (to the next player or ends the game) and stands for the player.
- As a player, your session will expire in 10 minutes of inactivity or when you log out or when you close the browser window.
- In case of wireless disconnectivity, you can join the same game as long the disconnectivity did not last for longer than 10 minutes.

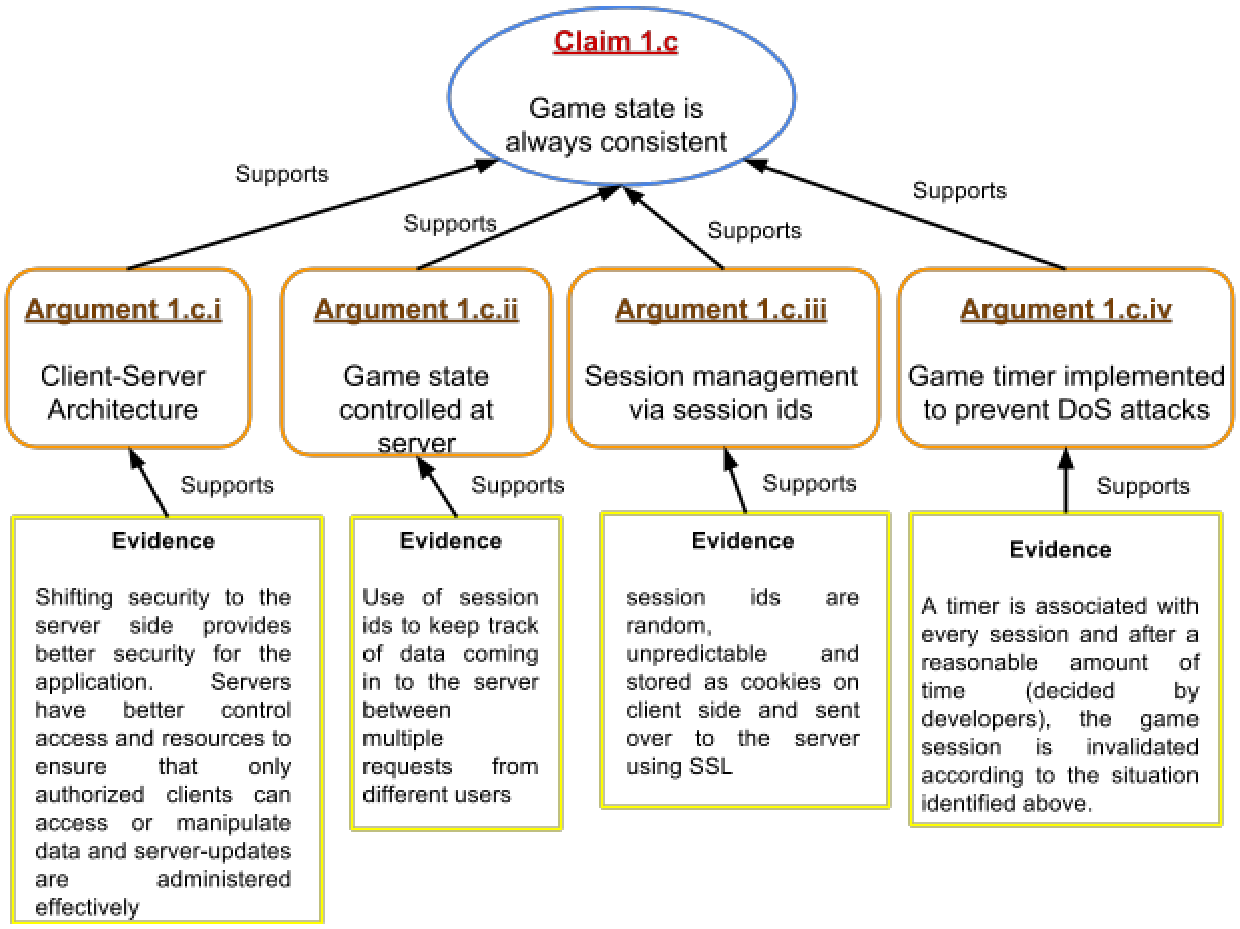

Claim 1.c
Game state is always consistent
Supports
Supports
Supports
Supports
Argument 1.c.i
Client-Server Architecture
Argument 1.c.ii
Game state controlled at server
Argument 1.c.iii
Session management via session ids
Argument 1.c.iv
Game timer implemented to prevent DoS attacks
Supports
Supports
Supports
Supports
Evidence
Shifting security to the server side provides better security for the application. Servers have better control access and resources to ensure that only authorized clients can access or manipulate data and server-updates are administered effectively
Evidence
Use of session ids to keep track of data coming in to the server between multiple requests from different users
Evidence
session ids are random, unpredictable and stored as cookies on client side and sent over to the server using SSL
Evidence
A timer is associated with every session and after a reasonable amount of time (decided by developers), the game session is invalidated according to the situation identified above.

Figure 8 - Claim 1.c, arguments and evidence

Claim 1.d - The application design is secure

Argument 1.d.i - Ensuring security patches are available for identified vulnerabilities of the technologies used
Evidence - We checked the security bulletins for patches of the technologies used to identify and understand any major vulnerability that might exists in our web application

Argument 1.d.ii - Ensuring randomness of card shuffling algorithm
Evidence - A log made with hundreds of shuffled deck revealed no visible way to predict a pattern in which the cards were being shuffled, use of the Fisher-Yates shuffle algorithm which used a random number generated from the java.security.SecureRandom library.

Argument 1.d.iii - Modular software components
Evidence - The code developed was made in modular packages in part of the complete web application so that some classes/packages can be reused.

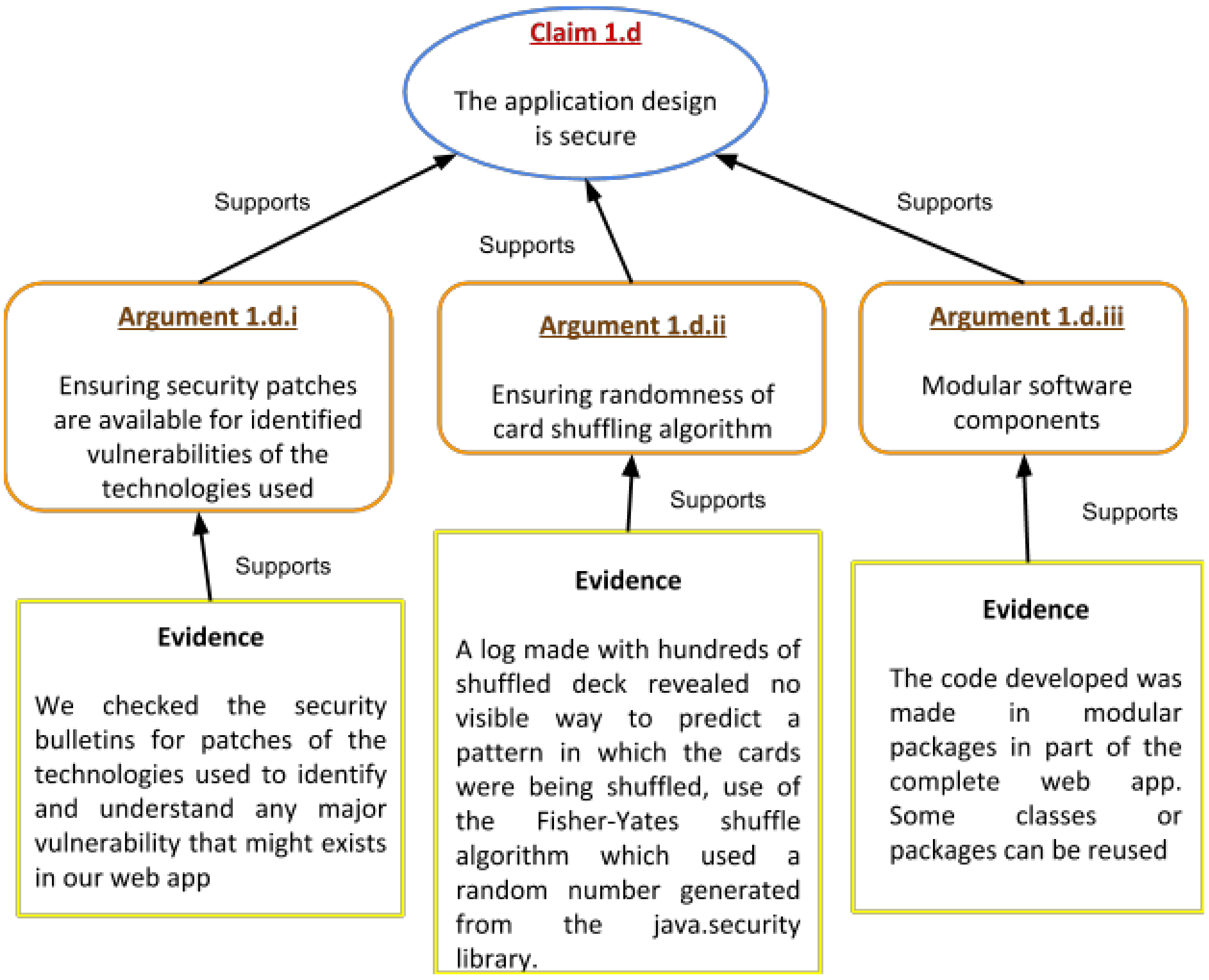

Claim 1.d
The application design is secure
Supports
Supports
Supports
Argument 1.d.i
Ensuring security patches are available for identified vulnerabilities of the technologies used
Argument 1.d.ii
Ensuring randomness of card shuffling algorithm
Argument 1.d.iii
Modular software components
Supports
Supports
Supports
Evidence
We checked the security bulletins for patches of the technologies used to identify and understand any major vulnerability that might exists in our web app
Evidence
A log made with hundreds of shuffled deck revealed no visible way to predict a pattern in which the cards were being shuffled, use of the Fisher-Yates shuffle algorithm which used a random number generated from the java.security library.
Evidence
The code developed was made in modular packages in part of the complete web app. Some classes or packages can be reused

Figure 9 - Claim 1.d, arguments and evidence

Overall Assurance Case

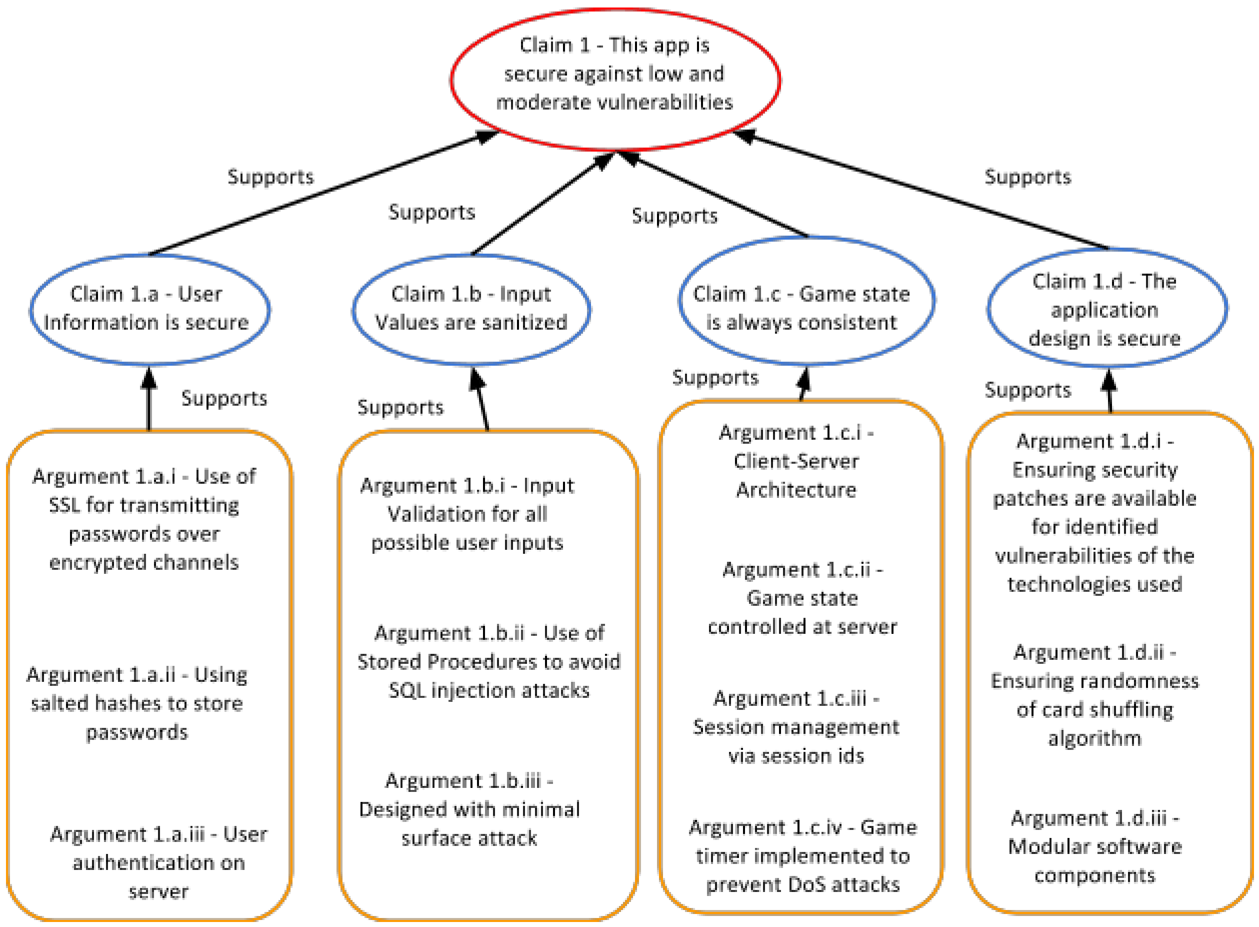

Claim 1 - This app is secure against low and moderate vulnerabilities
Supports
Supports
Supports
Supports
Claim 1.a - User Information is secure
Claim 1.b - Input Values are sanitized
Claim 1.c - Game state is always consistent
Claim 1.d - The application design is secure
Supports
Supports
Supports
Supports
Argument 1.a.i - Use of SSL for transmitting passwords over encrypted channels
Argument 1.a.ii - Using salted hashes to store passwords
Argument 1.a.iii - User authentication on server
Argument 1.b.i - Input Validation for all possible user inputs
Argument 1.b.ii - Use of Stored Procedures to avoid SQL injection attacks
Argument 1.b.iii - Designed with minimal surface attack
Argument 1.c.i - Client-Server Architecture
Argument 1.c.ii - Game state controlled at server
Argument 1.c.iii - Session management via session ids
Argument 1.c.iv - Game timer implemented to prevent DoS attacks
Argument 1.d.i - Ensuring security patches are available for identified vulnerabilities of the technologies used
Argument 1.d.ii - Ensuring randomness of card shuffling algorithm
Argument 1.d.iii - Modular software components

Figure 10 - Overall Assurance Case Model for the Blackjack web application