1

82

2. Submit to Moodle the link to your GitHub repository to mark your delivery.

Evaluation Rubric (out of 50 points)

- R1 Architecture section inadequate or missing: -10 pts
- R2 Technology section inadequate or missing: -10 pts
- R3 Data representation description is inadequate or missing: -10 pts
- R4 Coding Standards section inadequate or missing: -10 pts
- R5 (as promised) Over Inflated Story Point Estimations: -5 pts for over inflated story points. Any over inflation must be fixed before the next group assignment or incur additional penalties.
- R6 `pdflatex` fails: -10 pts
- R7 No GitHub Repository: -50 pts.

R1 3 tier

-3 main modules needed a bit of explanation

R2 I? ✓

framework ✓

persistence ✓

-3 testing framework missing

R3

~~-2 "all sorts"~~

~~-3~~ -3 see pg 7&8

R4

-2 commit rules % coverage

✓ R5

✓ R6

R7

-3 what is their v good but incomplete see pg 9

R8 missing?? oh the file exist -- need to include them

-4 the 9,43,45 screenshot is confusing/busy w/ some prose

I think this is in green? shape but w/o a narrative explanation it's hard to tell

Stories

UI Mockups missing

CS482 Go Fish Architecture, Design, and UI

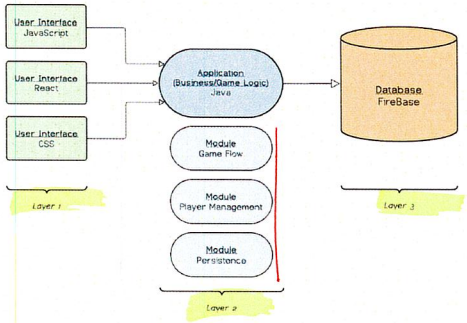
Ryland, Silas, Marley, and Chase
Client: Dr. Hoang Bui

October 2024

Contents

1 Architecture

In software engineering, the architecture refers to the high-level design that organizes a system's core elements. This includes, but is not limited to, components such as modules, packages, and programming languages. Key design decisions are involved which are essential for constructing a strong foundation to build off of, thus making it critical to have for successful future development. With this in mind, we have chosen to utilize a three-tier architecture. This will use JavaScript, React, and CSS for constructing the user interface, Java for the application, and Firebase as our database.



For our Go Fish application, we will utilize a three-tier architecture that is structured around user interface, application, and database. We have chosen this architecture because its components align with our goals in creating a card game web application that offers an enjoyable user experience in both visuals and playing the actual game. As seen in the figure above, it is made up of three distinct layers.

1.1 User Interface

Handling the display of game elements, user interaction, and input processing is essential for the frontend of a web application. We will be utilizing JavaScript, React, and CSS to do so. These languages will allow us to construct a well-designed UI to ensure a smooth and engaging experience for the user. The design emphasizes a classic card table feel with a responsive layout that adapts to various screen sizes. Cards are styled with consistent proportions, rounded corners, and a clear display for the suits and ranks of cards. The game board mimics a card table, featuring distinct player areas and a centralized deck. Interactive elements like hover effects on cards and smooth animations for card movements enhance user engagement.

1.1.1 Visual Design Draft

1. **Card Design**
   - Card Size & Proportions: Maintain a consistent aspect for the cards
   - Visual Components
     - Suit & Rank: Place the rank and suit symbol clearly in the top-left and bottom-right corners of each card
     - Centered Design: Use the center of the card for the suit symbol
     - Rounded Corners: Give the cards rounded corners for a modern look (use border-radius in CSS)
     - Border: Add a subtle border around the cards to distinguish them from the background
2. **Display for Hand of Cards**
   - Grid Layout for Player's Hand: Use a horizontal layout for cards in a player's hand
   - Hover Effects: Make the cards interactive with hover effects
3. **Playing Table**
   - Table Layout: The game area should mimic a table for cards
   - Player Zones: Each player should have a distinct area where their hand of cards is displayed
   - Center Game Elements: Ensure the draw pile or game deck is in the center of the table
4. **Visual Aesthetics**
   - Color Scheme: Use a color palette that is easy on the eyes and give it a classic card table feel
5. **Animations & Transitions**
   - Card Movement: When a card is drawn or played, add subtle animations to make the movement more engaging
   - Transition for Cards: Use smooth transitions for flipping cards or drawing them from the deck
6. **Responsive Design**
   - Ensure the game layout is responsive for various screen sizes like desktops to mobile devicesk

7. **Player Information Display**

   - Player Avatars and Names: Display player avatars or profile pictures next to their card area for personalization

Overall, the design prioritizes user experience, blending classic card game elements with modern web design principles to create an enjoyable game of Go Fish for the players.

1.2 Application

The application layer will involve several key modules that are crucial for the actual game rules. The player management module will handle everything related to the player, such as storing and managing player profile information or managing each player's hands during games. The game flow module is responsible for the actual gameplay mechanics of Go Fish in terms of ensuring each player's turn runs accordingly based on the real card game. Finally, we have the persistence module which ensures that database operations (such as saving or retrieving game states) are abstracted from other parts of the business logic.

1.3 Database

In the third layer, we will be implementing Firebase as our database for storing all sorts of data that is to be manipulated by the system. For example, once a game of Go Fish is completed, each player's information is updated in the database in terms of their wins/losses and amount of virtual currency.

Here is what this should be detailed

point to §3

2 Technology

The application will leverage a powerful stack of modern web technologies. Firebase Authenticaiton will handle user authentication, allowing players to securely sign up, log in, and manage their accounts. Firestore, Firebase's flexible and scalable NoSQL database, will store and synchronize game data in real-time. This includes player information, game states, and card decks to ensure users experience changes in real-time.

Express.js is a minimal and flexible Node.js web application framewor that will serve as our backend. It will handle tasks such as HTTP requests, routing, and API endpoints to act as an intermediary between the client-side application and Firestore.

Whereupon Node.js will power the server-side logic, enabling efficient handling of concurrent connections and real-time updates.

who wrote this

3 Persistent Data

Collections will contain documents that belong to them. Such documents will then contain attributes to describe themselves and what they contain. This layout is depicted below in the same order and will be handled properly to be stored in Firebase.

3.1 Users (Collection)

- User's Authorization ID/Email (Document)
  - Username: String
  - User Logo: String
  - Email: Boolean
  - Google Email: Boolean
  - Friends: Array of usernames
  - Friend Requests: Array of usernames
  - Games Played: Integer
  - Games Won: Integer
  - Bets: Array of past bets placed

3.2 Lobbies (Collection)

- Lobby ID (Document)
  - Owner User ID: String (creator of the lobby)
  - Lobby Type: String (can be public or private)
  - Lobby Password: String (if it is private)
  - Player Limit: Integer
  - Players: Array (User IDs)
  - AI Fill: Boolean (whether AI fills in for empty spots)
  - Status: String (either open or closed)
  - Betting Pool: Integer (collected bets for a game)

3.3 Friends (Collection)

- Friends List (Document)
  - User ID: String (when requesting friends)
  - Friend ID: String (recipient of friend request)
  - Friends: Array
  - Status: String (waiting, accepted, or declined)

3.4 Game Sessions (Collection)

- Game Session ID (Document)
  - Lobby ID: String
  - Players: Array (User IDs of players that are currently playing)
  - Current Turn: String (username of the player who has an active turn)
  - Deck: Integer (number of cards remaining in deck)
  - Player Sets: Map (User ID should be connected to an array of completed sets)
  - Game Status: String (ongoing or finished)
  - Winner: String (username)
  - Winner: String
  - Prize Pool: Map (User ID of winner should be given the pooled money form bets)

3.5 Cards (Collection)

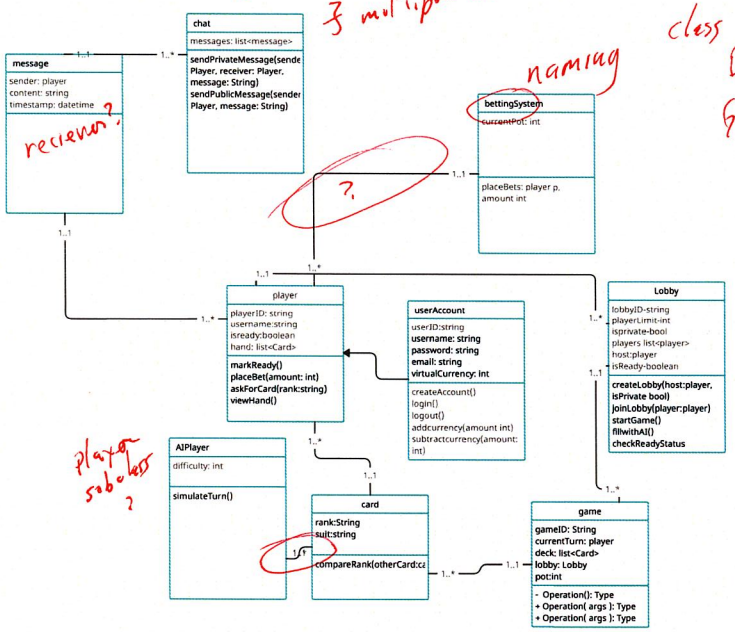
- Game Session (Document)
  - Game Session ID: String
  - Deck: Array (of Cards)
  - Player Hand: Map (connecting a User ID to a unique array of cards)

3.6 Bets (Collection)

- Game Session (Document)
  - Players: Array (of players involved in the bet)
  - Total: Integer (of the finalized pool)
  - Winner: String (User who won, prize mapped to their account and info)

3.7 Shop (Collection)

- Item (Document)
  - Item Name: String
  - Item Pic: String (hardcoded link to show the image)
  - Price: Integer
  - Featured: Boolean (for weekly new limited edition items)



In this UML diagram, we outline the architecture of our Go Fish card game application, which integrates several key components. At the center, the player class handles user actions such as betting, viewing their hand, and interacting with the bettingSystem. We've designed the Lobby to manage game sessions, enabling players to join, start games, and interact with one another through the chat system. Additionally, the AIPlayer class simulates opponent behavior with varying difficulty levels. Our design also supports account management via the userAccount class, allowing players to log in, manage their virtual currency, and participate in games.

4 Coding Standards

- Use Camel Case for naming conventions (e.g., thisExampleHere)
- Write meaningful commit messages
- Include class and method descriptions
- Add inline comments for clarity
- Follow a clear and concise document structure for Firebase
- Implement uniform error handling throughout the project
- For Firebase, prioritize authentication through secure endpoints for all database interactions
- For Java, ensure each class represents a single responsibility
- Prioritize efficiency within the code

Our coding standard follows several key principles, ranging from the simple usage of camel casing for naming conventions to ensuring meaningful commit messages.

Each class and method should include clear descriptions, and inline comments should be used for better code readability. A clear and concise document structure is required for Firebase integration, with consistent error handling throughout the project.

For Firebase, secure endpoints must be prioritized for all authentication and database interactions. In Java, each class should adhere to the single-responsibility principle. Lastly, efficiency within the code is a top priority. For example, repeated lines of code will be made into a function and variables that can be grouped will be made into objects.