0

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

92

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

MVC

-5 main module missing

R2

I ✓

-2 frameworks

✓ persistence ✓

-3 testing framework missing

R3

-5 main collection not specified

R4

-3 commit rules missing

R5 ✓

R6 ✓

R7

-5 incomplete See me

R8

-5 there will be more than one screen in your app

War Card Game
CS482 Software Engineering

Ayoposi Olu-Bamisaye, Jonathan Ramos, Brett Bonner
Client: Dr. Eric Cui

2024-10-09

1 Design

1.1 Architecture

We are planning on implementing a Model View Controller (MVC) architecture type for our game. It will be able to handle our in-game logic and how the user interacts with the game itself. The Model typically will represent the game logic, deck, and cards. The View will be responsible for representing the current state of the game to the user. Examples of this are displaying the players stats, profile and friends, having the ability to view output of cards/deck and showing tables in the lobby. We will be using a graphical feature to show this in our project. The Controller will handle user input for various commands like starting the game, ending the game, placing cards, party privacy and updating the players profile settings.

1.2 Technologies

Since we are developing a web-based version of the card game "War", we have decided to use modern web technologies to ensure we have a responsive and scale-able application. Our front end will consist of common technologies used for web development, HTML, CSS, and Javascript. HTML will provide the basic structure of the web page, and CSS will allow us to style the page according to the theme specified by Cosmic Radiance. Javascript will handle game logic on the client side. Our front-end framework will be React.js as this will facilitate with updating the game-state without refreshing the page. React will also help with creating reusable components such as the cards themselves.

Our back end will need to handle the game logic, player data, and real-time communication. Node.js will allow us to handle game logic outside of the browser. This will allow us the necessary means of handling requests such as determining winners. We will use a library Socket.io to handle real-time communication between multiple clients.

To store our user data, game history, and player stats we will need persistent storage. We have chosen Firebase as our persistent storage due to its ease of use with real-time updates. It will also allow us to have a more flexible schema as SQL databases require relations among the tables created. Firebase also includes built-in authentication, supporting Google and Facebook sign in, as well as traditional email and password. Overall, Firebase supports our needs of real-time updates, scalability, security, and rapid development.

1.3 Persistent Storage

Our long term data will be stored in documents. The data will be organized as JSON documents which are then organized into collections. Our users will all be stored in a collection. The collection will hold documents, and each document will hold information related to one player. Each document is the equivalent of one player and therefore, each player will need to have their own document to store their information. Another collection that we will implement is the leader-board. Each document in the leader-board collection will contain information regarding the player that has achieved recognition. This structure is flexible, making it scale-able which is important for an online multiplayer game. Our

NoSQL design ensures that we can continue adding more collections and documents as needed without restructuring the entire database.

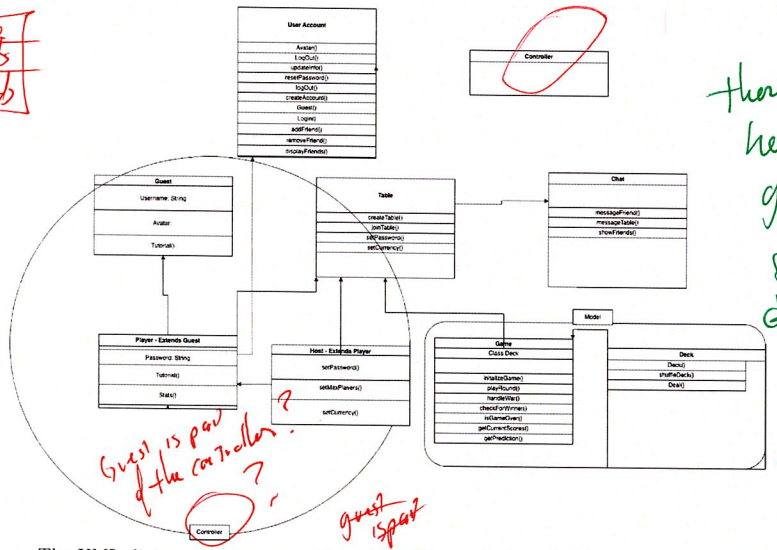
1.4 Coding Standards

Our database naming scheme will consist of singular, lowercase names for collections (user, leaderboard). Each attribute within documents will use camelCase (userName, gameStats, gameID). Variables and functions in JavaScript should also be written in camelCase.

We will format our code using two spaces for indentation. Brackets should be used on the same line for functions and other control statements. Inline commenting should be used only to explain complex logic. Multi line comments should be used to describe functions and classes. Additionally, all functions should be properly documented with a multi-line comment describing the purpose, parameters, and return value.

Testing standards will consist of test-driven development. We will ensure proper functionality by ensuring that all code commits pass necessary code tests before merging. Code tests should handle correctness, edge cases, and performance.

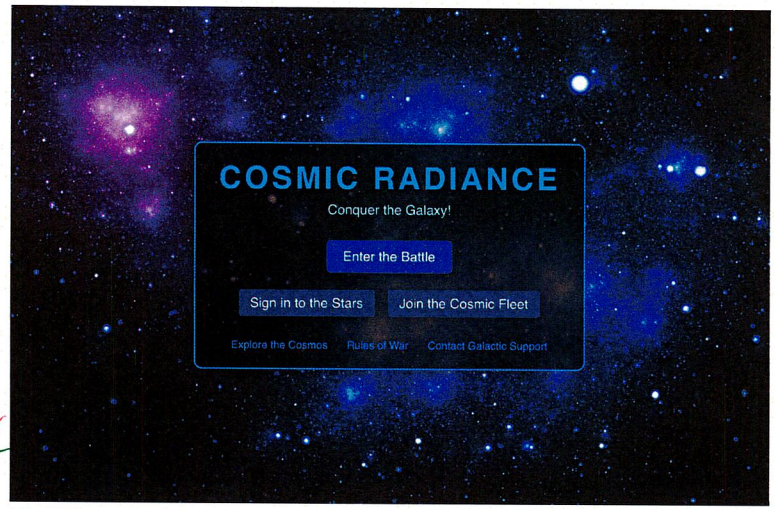
2 UML Diagram



The UML diagram creates an outline of the War card game for Cosmo Radiance. The system features a Host class that extends the player class, granting the host control over

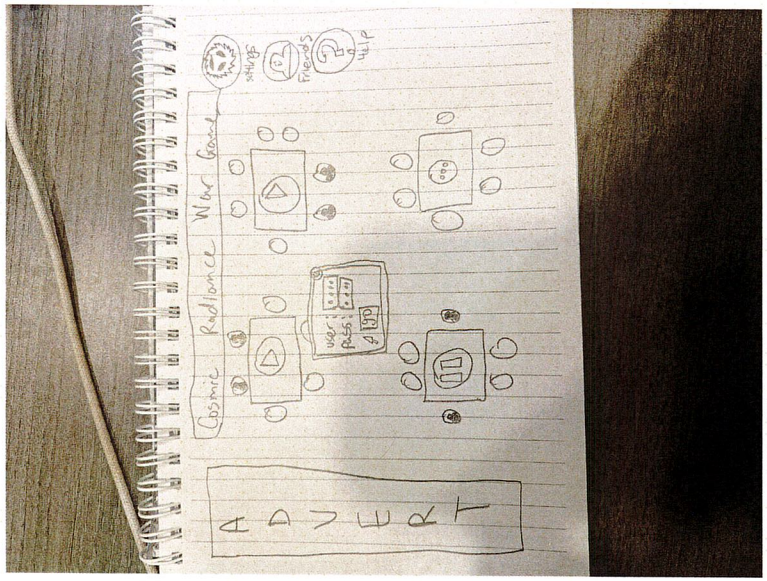
the table, including decisions on who can join. Additionally, a Game class manages the gameplay mechanics, ensuring that the rules of the game are followed. The User Account class manages all of the information of the User, from their password to avatar, to their friends. Lastly, a table class is made to allow users to join the table or create their own table.

3 UI Mockups

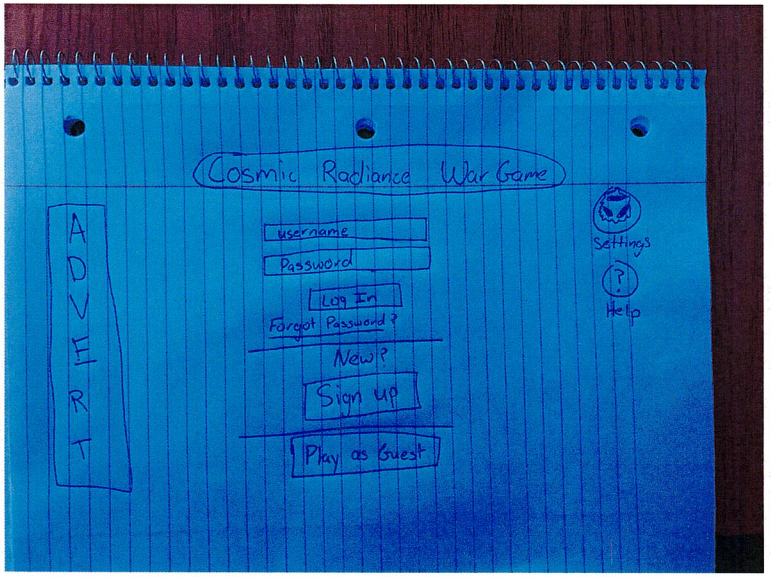


The UI design for Cosmic Radiance web app's welcome screen features a space-themed layout with royal blue tones.

0



there are missing screens
these also need a b.7- o
explanation



Login Screen for Cosmic Radiance War Game.