Team 17 Project 3 Architecture Document

Project Name: Online Card Games
Team 17 Members: Ryan Grimsley, Mya Hoersdig, Henry Hoopes, Benjamin Stonestreet, Connor Williamson, Abdelrahman Amir Zeidan

Project Synopsis:

An online platform that allows users to play card games.

Architecture:

This project aims to provide an online card gaming platform accessible through a web browser, starting functionality with Blackjack as the initial playable game, and expanding into others(UNO, Poker, War, etc) if we are able to within the project time limit. Users will be able to join multiplayer lobbies and play rounds of blackjack against the same virtual dealer using virtual in-game currency. Users will also be able to register accounts, log in, and manage their profiles. The system's long-term goal is to establish a foundation for serving additional card games to users.

The platform is going to follow a client-server architecture, with a React frontend for the user interface and displaying graphics, and a Golang backend for the logic, game management, and data handling. The frontend and backend will communicate with a HTTP API connecting the two layers. HTTP communication between the frontend and backend allows for straightforward communication through simple HTTP requests. The database that will be used is SQLite, accessed by the backend through the GORM ORM library.

The React frontend will handle the main user interface. It will perform actions like: render the game interface and player dashboards, handle user input, make HTTP requests to the backend, display backend responses and update interface accordingly. A typical user flow through the system might be:

1. The user logs in or registers through a dedicated register or log in page.
2. The frontend sends an HTTP request with their credentials to the backend.
3. The backend verifies or registers the credentials.
4. If the credentials are valid, the backend creates a session for that user, storing a unique session ID, and setting a cookie for the session/user.
5. The browser stores this cookie and includes it with every future HTTP request.

6. The backend reads the session id cookie with each request and knows which user is making the request.

7. When the user logs out, the backend deletes the session entry and the browser clears the cookie.

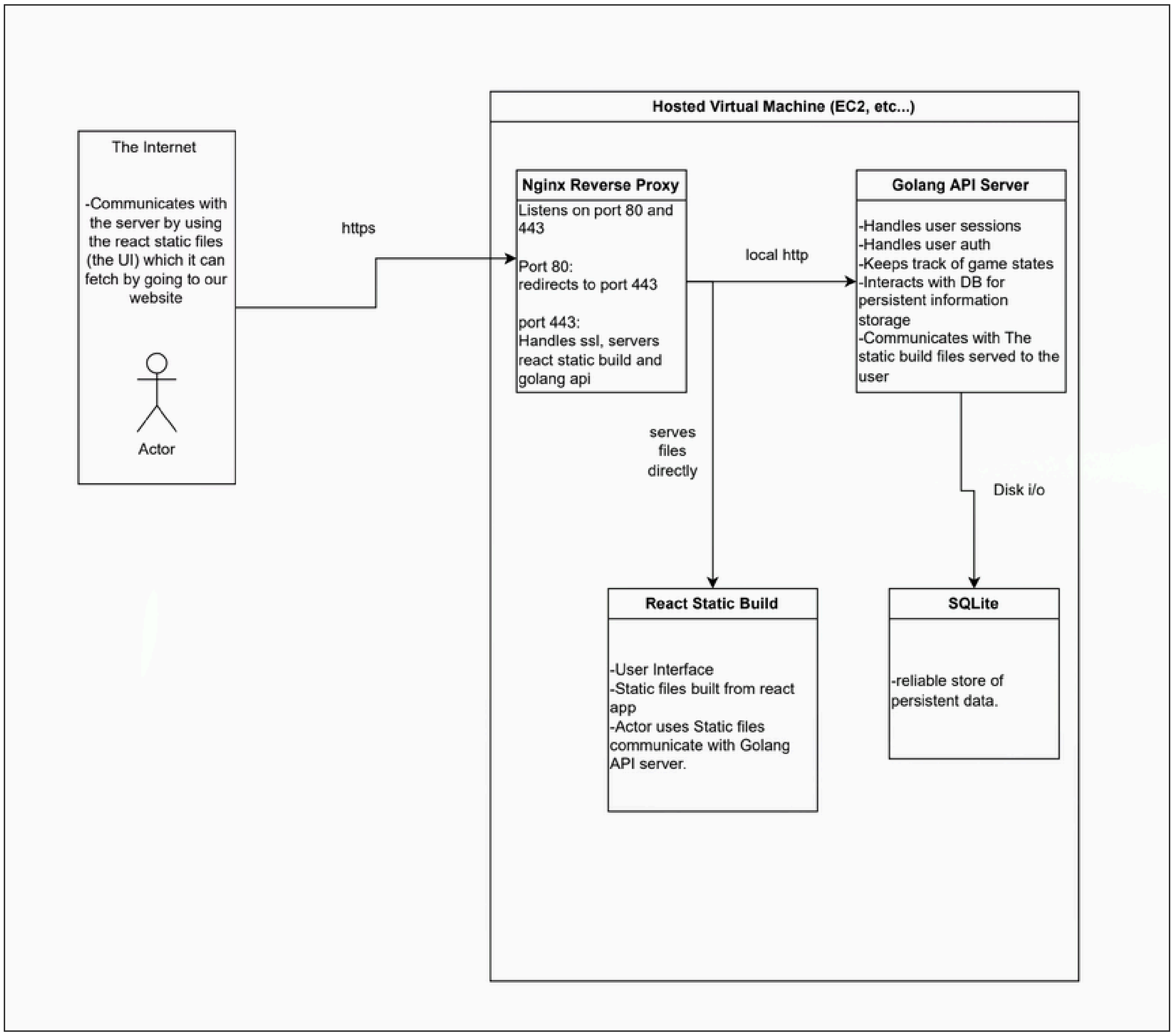
The Go backend performs the main game logic and handles user management. Its core responsibilities are: user registration, authentication, account retrieval; creating sessions and enforcing game rules; creating, listing, and handling users joining lobbies; saving user accounts, balances, and stats to SQLite database. The backend will expose different API endpoints so that the frontend is able to send different types of user input and get sensible responses from the backend accordingly.

The SQLite database stores persistent data like: usernames, hashed passwords, balances, player win/loss counts, games played. SQLite is simple and portable, requiring no external server processes, making it a desirable choice for our project. During gameplay, the system will likely follow a turn-based request model where each player’s move sends a new HTTP request to the backend, which then does things like process the move, update game state, and send a response back containing the updated game state information.

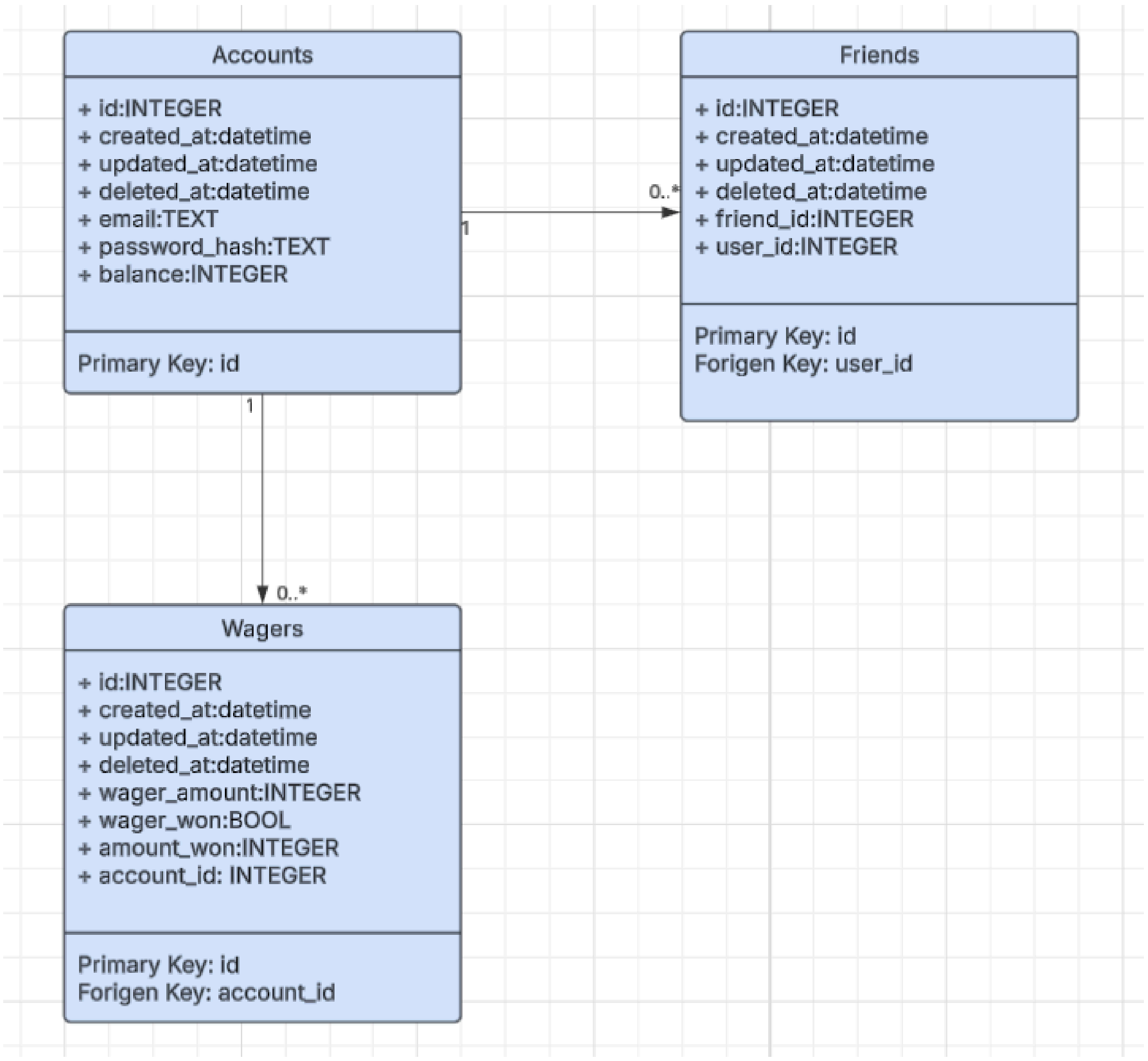
We are currently exploring options for final deployment, but considering different cloud providers or hosting it on one of our own locally owned servers.

This client-server architecture using React, Go, and SQLite should provide us a strong foundation to build our project on, allowing for solid maintainability and extensibility going forward.

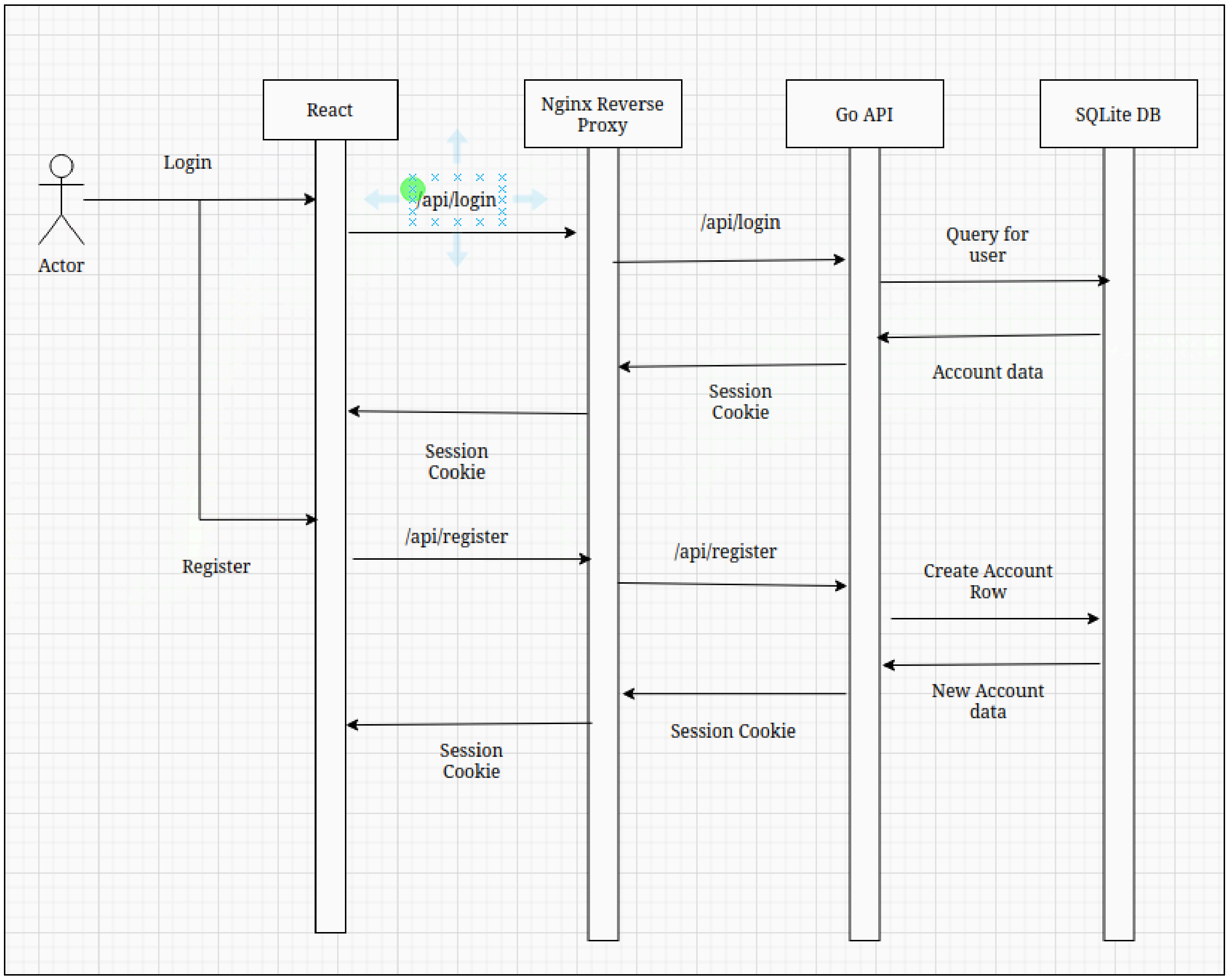
1. Architecture UML diagram.


The Internet
-Communicates with the server by using the react static files (the UI) which it can fetch by going to our website
Actor
https
Hosted Virtual Machine (EC2, etc...)
Nginx Reverse Proxy
Listens on port 80 and 443
Port 80: redirects to port 443
port 443: Handles ssl, servers react static build and golang api
local http
Golang API Server
-Handles user sessions
-Handles user auth
-Keeps track of game states
-Interacts with DB for persistent information storage
-Communicates with The static build files served to the user
serves files directly
Disk i/o
React Static Build
-User Interface
-Static files built from react app
-Actor uses Static files communicate with Golang API server.
SQLite
-reliable store of persistent data.

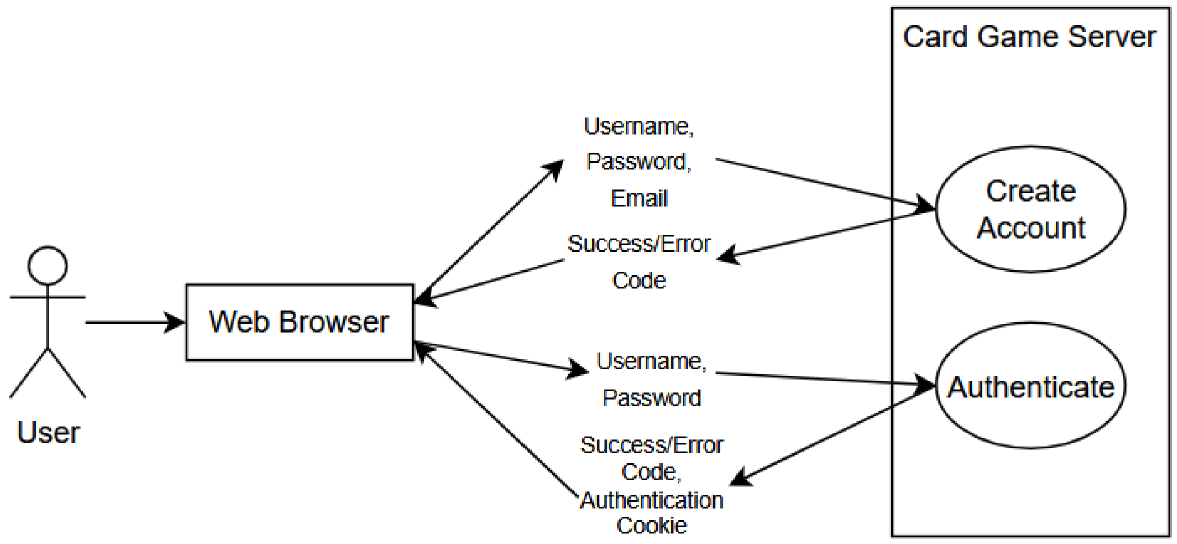
2. SQL Diagram:



3. Sequence Diagram:



4. Current API use case diagram:


Card Game Server
Username,
Password,
Email
Success/Error
Code
Create Account
Web Browser
User
Username,
Password
Success/Error
Code,
Authentication
Cookie
Authenticate