Kahoot Clone

Team: GoodbyeStuy
Haotian Gan (PM), Wen Hao Dong, Renggeng Zheng

Frontend (Next.js deployed on Vercel's CDN):

- Next.js ("The React framework for production")
    - https://nextjs.org/
    - Handles page routing
    - Created by Vercel, the company behind Next.js. Vercel allows us to deploy our frontend on their CDN
    - Provides support for deploying NodeJS serverless functions

Database:

- MongoDB
    - Collection "games" contains documents, each of which is a kahoot game

```
interface KahootGame {
  id: string; //uuid of the game
  author: string; //uuid of the author
  title: string;
  questions: Question[];
}

interface Question {
  question: string;
  choices: string[];
  correctAnswer: number; // integer index of the correct answer
  time: number; // integer seconds
}
```

NodeJS Serverless Backend API (deployed on AWS automatically by Vercel):

- /updateKahoot
    - Given a kahoot ID and a kahoot game, updates that kahoot game
    - Given no kahoot ID and a kahoot game, it adds a new kahoot game to the "games" collection
- /deleteKahoot
    - Given a kahoot ID, deletes that kahoot game.
- /getGames
    - Given a user ID, a limit, and an offset, gets that user's games from most recently created to oldest.
- /signup
    - Given username and password, creates a new user
- /login
    - Given username and password, responds with an access token.

Rust Server Backend:

- Axum server framework (https://github.com/tokio-rs/axum)
    - Asynchronous
    - Built on top of the tower library (https://crates.io/crates/tower)
    - Provides easy support for websockets
- Tokio async runtime (https://tokio.rs/)
    - Multithreaded
    - Lightweight asynchronous green-threads called "tasks," similar to Go's goroutines
- Serde (https://serde.rs/)
    - Serialization and deserialization of Rust data structures
    - serde_json for parsing and generating JSON (https://docs.serde.rs/serde_json/)

Rust Server Websocket API (GET /ws):

- Upon connection with websocket, the client must send a createRoom or joinRoom message
- Client “actions”:
    - createRoom
        - After sending a create room message, the server responds with the room id, and the client is now considered a “host”
    - joinRoom
        - If the given room id is invalid or the username is a duplicate, the connection is closed
        - Upon a successful connection, the client is then considered a “player”
    - answer
        - Sent by players to the server when they answer a question
        - If the client is not a player or they already answered the question, the message is ignored
    - beginRound
        - Sent by the host to begin the next round
    - endRound
        - Sent by the host to skip waiting for players
- Server “host events” (messages sent to the host):
    - roomCreated
        - Contains the id of the newly created room
    - userJoined
    - userLeft
    - userAnswered
    - roundBegin
        - Contains information about the question
    - roundEnd
        - Contains information about which players got the question correct and how many points they got

        - gameEnd
            - Sent when there are no more questions
    - Server "user events" (messages sent to the players)
        - roundBegin
            - Contains the number of choices in the question
        - roundEnd
            - Contains how many points the player gained (if any)
        - gameEnd

Roles:
Haotian Gan: Frontend, serverless functions
Wen Hao Dong: Rust backend
Renggeng Zheng: Frontend, Websockets functionality

Repo structure:
```
frontend/
  pages/
     api/
  public/
  styles/
backend/
  src/
    main.rs
  Cargo.toml
```

Deployment:
Since the frontend and backend are decoupled from one another, the two can be hosted on different servers (but the frontend could still theoretically be served by the backend).

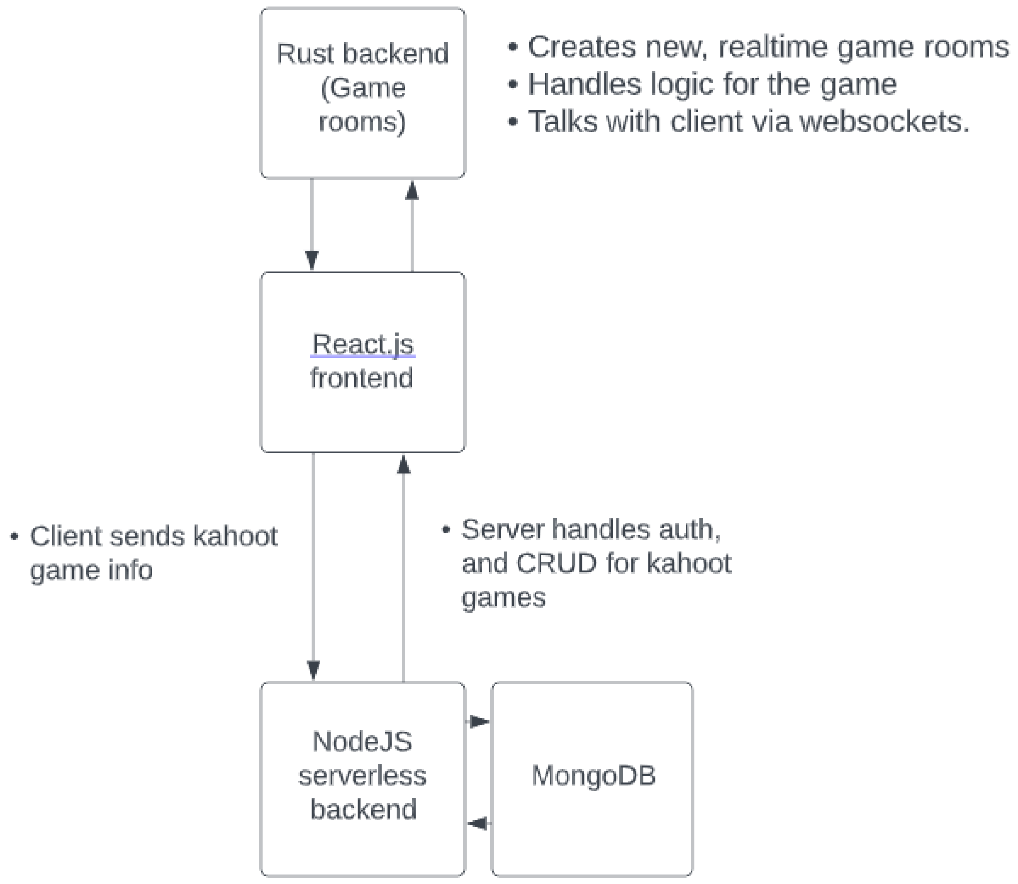
Our frontend is deployed on Vercel. It connects to the backend via the backend's API.

Javascript functions written inside `frontend/pages/api` take an http request and return an http response. Each of them are deployed as their own `/api` endpoint by Vercel on AWS. They are known as "serverless functions".

The Rust backend is deployed as an digital ocean droplet with nginx as a reverse proxy to forward the client requests to the rust server and vice versa.

The main benefit of separating the front and back end is that either one could be updated or swapped out without fear of messing up the other as long as the API stays consistent.

Component Map:


Rust backend (Game rooms)
• Creates new, realtime game rooms
• Handles logic for the game
• Talks with client via websockets.
React.js frontend
• Client sends kahoot game info
• Server handles auth, and CRUD for kahoot games
NodeJS serverless backend
MongoDB

Target Ship Date: 6/1/2022