Term Project Spring 2021

Jessica Zepeda, Diana Benavides, John Roberts

Team G

GitHub Repo:

https://github.com/sfsu-csc-667-spring-2021-roberts/term-project-team-g-dprj.git

Exploding Kittens

Table of Contents

Architecture and Implementations

Created a web application that supports a asynchronous real-time, multiplayer, online game that is capable of handling messages synchronously. The design for this project is called ***Event-Driven Architecture (EDA)***, which handles a series of events that accepts all data and then finds the correct module that corresponds to that particular action. For instance, this was accomplished by creating functions inside separate modules for each event that was generated by the user from the client side. So once the event is triggered by the user, that element informs the system, so the component performs the request. This is highly beneficial because when a request fails, then only that corresponding function in the module for the specified event is only affected, rather than the whole web application. As mentioned, the UI for the web application is ***asynchronous***, which eliminates timeouts and allows multiple applications to communicate and players can interact concurrently in real time. Another advantage for this architecture is that it is highly scalable, meaning it can handle increased workloads and has the opportunity to grow in time and provide great user experience for new users by handling more requests per min over time.

Required Functionalities:

- **Authentication**
- **Game state**
- **Chat room**
- **Arbitrary number of games**

We used ***Passport*** for authentication because it has great benefits such as easy configuration, supports persistent sessions, offers open authorization (OAuth) and provides separate modules for each strategy, among other benefits. Since Passport.js is middleware, we used Express to achieve user authentication. ***Express*** provides many features such as routing components and supports middleware so developers can build simpler web applications. For the game state the user initially waits for every player to join the room before the game begins. In general, the game state consists of four states, initial, forward, backward, and game over.

For this project we created several migrations to record multiple records of events and saved that data in the backend database ***Postgres*** and used ***Pg-promise*** to access the tables from

the database. Originally we were going to use Socket.io for the chat room but eventually switched to Pusher channels. Pusher is driven by service/API and socket io demands more monitoring in deployment. Hence pusher came out to be less work and much easier to manage. The user is able to create a game and join any existing game that is active.

Implementation Problems

The game state was not fully finished. We were able to get the initial state of the game, where each registered player joins the room and waits until every player has joined the game. Other issues that arose at the last minute were deployment, messages not working, and user authentication had minor bugs but successfully fixed the bug. In order to get the chat room working, the whole team had to get a pusher account and save the pusher keys inside the environment file (.env). However, we were not able to identify the problem.

Difficulty Discussion

We all had different schedules and only met once a week. Most tasks were not completed on time. Lack of coding experience was also an issue such that there was a lot of backend use where it was hard to grasp how the technologies were being used. The technologies were really new to our team as we were trying to learn along the way and understand how everything would connect both frontend and backend. Due to this completing tasks on time became a much harder issue and not asking enough questions. Some team members were also completely new to this game meaning that there had been issues once creating the game logic. Therefore, once developing the game it became harder to understand the aspects of shuffling the deck, considering multiple types of the same cards within the deck and what events would occur when each type of card would be played. Throughout this project there came a lot of difficulty as all these issues greatly impacted the development of our app.

Technologies

Heroku, Node.js, Express.js, Postgres, Pusher, Pug, Passport.js