Computer Science Department
San Francisco State University
CSC667
Spring 2022

Uno Game Term Project

https://github.com/sfsu-csc-667-spring-2022-roberts/term-project-code-team-6

Joseph Kois
Hira Afzal
Wenjie Ye
Ivan Cebreros

I. **Description of Architecture**

For our Uno application, we wanted to make the application as modular as possible to improve the reliability and testing of our project. By having a modular system it'll make it easier to run unit tests, and easier to modify code given a new feature, or bug fix.

In terms of the front-end development portion of this project, we decided that we wanted to utilize the clarity of handlebars in terms of syntax and the utilization of logic-less templates, alongside with css. We have separate .hbs files for different pages of our application, like the "how to play uno" page, or the login/signup page, and the game pages.

For the cards in our program, we decided to utilize the recommendation made in class to create one sprite sheet –which is a png file that displays all the cards. And then use the css styling to capture the location of the cards on this sprite image. In a way it essentially acts as a tile map for the game. And we are essentially splicing the tiles as individual components.

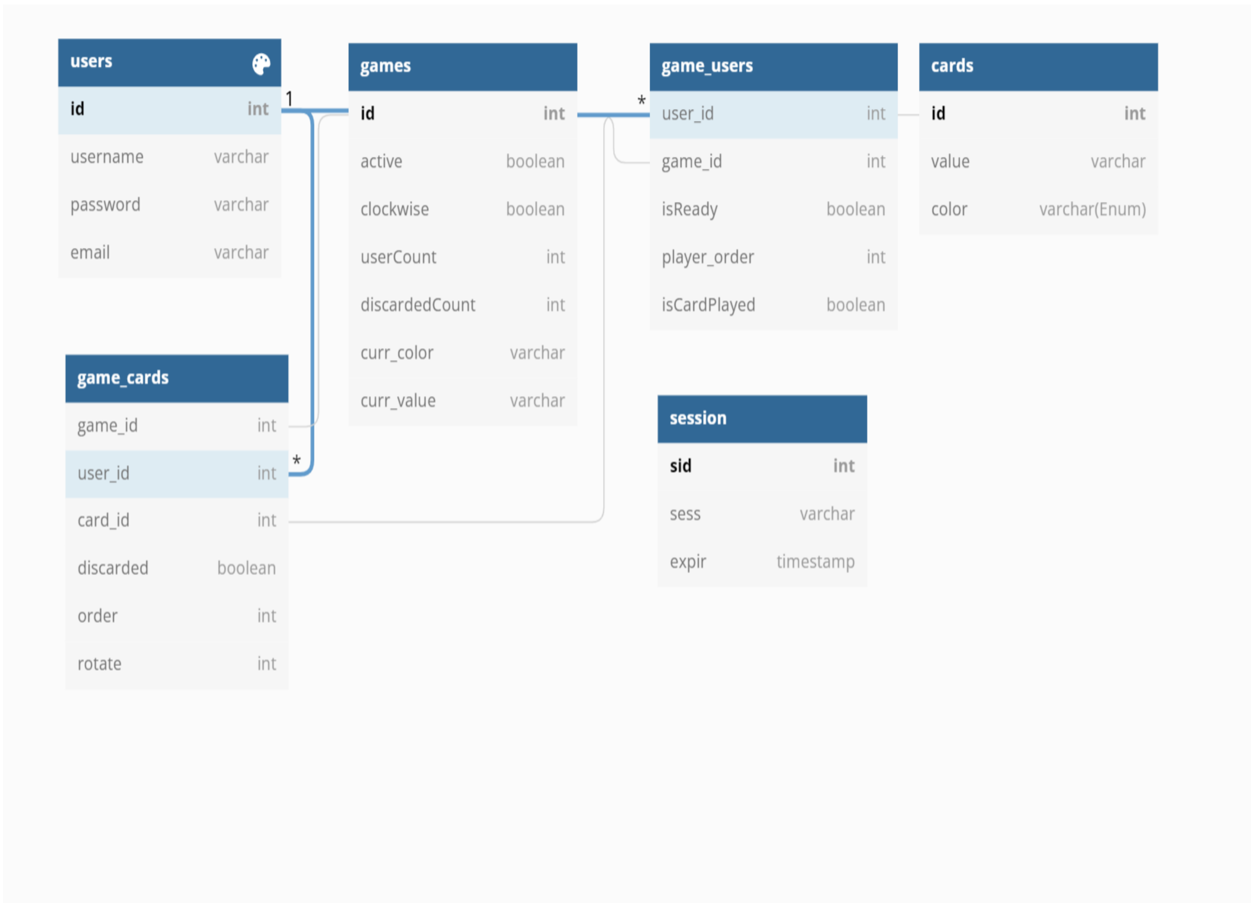
For the backend, we are utilizing ExpressJS to create our routes for the application, which consists of the following: auth.js, game.js, index.js, tests.js. Game.js encapsulates most of the game logic itself, and index.js encapsulates most of the user actions like login and signup.

In our db folder, we do have two files that are populated with async functions. In cardDao.js, this file houses a function that will draw cards for users. And gameDao.js essentially just houses a function called checkUserTurn that will take a gameId, and userId and return a boolean.

We only have one middleware file which is the authentication for the user. It is only a function called authUser which returns a response redirect and a log message.

Our socket file is titled index.js and is housed inside the sockets folder. It contains an io.on connection which encapsulates multiple socket.on calls containing different aspects of our application like lobby, disconnecting, joining room, chat messages, drawing a card and playing a card.

The design of our database was in collaboration with the class lecture and the way we designed ours is outlined below:



II. <u>**Problems during Implementation**</u>

A. Issue 1

Problem:
It is our first time working with socket.io so we need to understand how it works and how to use it in our app.

Solution:
The official docs clearly explains it and we are able to use it in our app

B. Issue 2

Problem:
We want to emit to a subset of users not to all users when using socket.io. For example, if we play a card in a game room, we do not want the card to be displayed in another game room.

Solution:
We use socket.join method to only send data to a specific game room, so that we do not receive all the data from all game rooms

C. Issue 3

Problem:
The requirement of not allowing client to send any data using socket. It will be easy to implement if we can directly send message from client.

Solution:
We rewrote the part where the client would need to emit data and we moved the logic to the server and send data to all clients from there.

III. <u>What was Difficult</u>

- Designing the database
- Understanding socket.io
-