CSC 667-02

Spring 2021

Documentation

GitHub Repository: Term Project Team JAKE

Team E

Kamelia Shaharova

Jahir Hernandez

Alexander Hall

Eanguy Eng

Table of Contents

Overview

For our term project, our features would include an authentication process where new users can create a new account with a username, email, and password that is hashed into our database. Once successfully submitted, they're redirected to the login page to submit their information to get access to our site. If someone attempts to access routes not authorized to them without being logged in, they'll be redirected back to the login page instead. From the lobby page after logging in, the user has options where they can either find games, create a new game, or log out from our site. Finding for games directs the user to game rooms where all ongoing games are displayed and their names to identify them. For creating a game, the user assigns a name for that game room along with the max amount of players to be able to join that game room.

Technology Stack

- Express.js, Sessions
- PostgreSQL
- Node
- Heroku
- Javascript, Pug, CSS

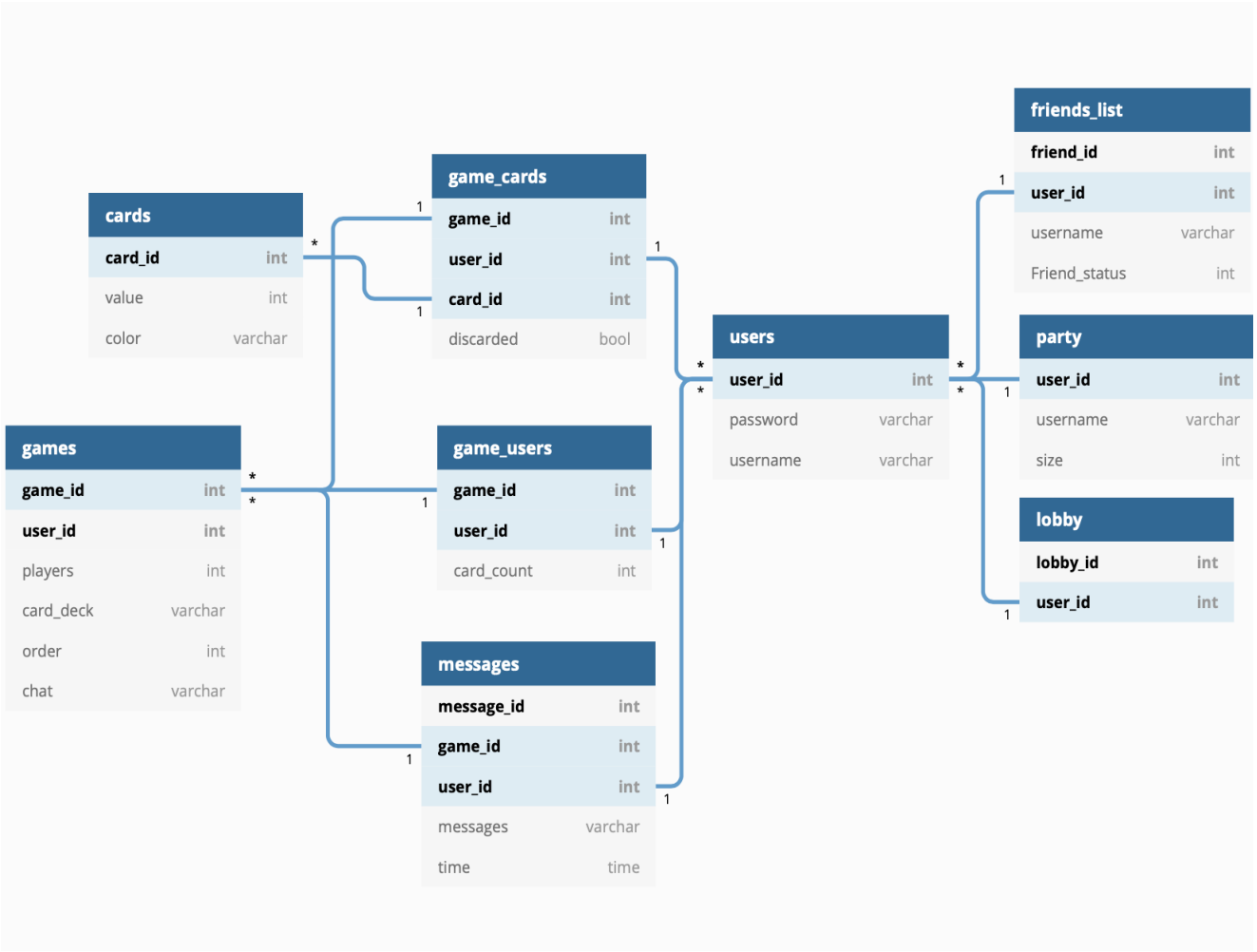
Deploying Server

1. Run `npm install` in your terminal of the project
2. Run `npm run db:migrate` to retrieve the migrations of the database tables for this project
3. In the .env file, you'll want to enter the credentials for linking your database to have an established connection
4. To run the website on your machine, type `npm run start:dev` into your terminal of the project. The port runs on 3000, so you will type localhost:3000 into your browser.
5. Navigate through our site

Migrations

1. Run `npm run db:rollback` in your terminal to delete migrations
2. Run `npm run db:migrate` in your terminal to run the migration

Database Structure



Database Description

Stored Data & Constraints

Our database needs the following entities:

- **Games**
  - game_id PRIMARY KEY, NOT NULL, UNIQUE)
  - players (int)
  - user_id (int)
  - card_deck
  - order (int)
  - chat (varchar)
- **Users**
  - user_id (PRIMARY KEY, NOT NULL, UNIQUE)
  - password (VARCHAR, NOT NULL)
  - username (VARCHAR, UNIQUE)
- **Lobby**
  - lobby_id(int)
  - user_id (int)
- **Party**
  - usernames (varchar)
  - user_id (int)
  - size (int)
- **Game_users**
  - games_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - users_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - card_count (int)
- **Cards**
  - card_id PRIMARY KEY, NOT NULL, UNIQUE)
  - value (int, NOT NULL)
  - color (VARCHAR)
- **Game_cards**
  - game_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - card_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - user_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - discarded (BOOL)
- **Messages**
  - message_id (PRIMARY KEY, NOT NULL, UNIQUE)
  - game_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - user_id (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE)
  - messages (VARCHAR[200])
  - time (time)

Deploying to Heroku

Deployed on Heroku at earlier stages, but recent overhaul of changes caused errors and issues to push to Heroku master

Assumptions

Kamelia: Most of the technology we used was new to me, such as: Postgres, Pug, Heroku, etc. It was a bit difficult setting up Postgres, as well as figuring out my way around Pug as it it's a little sensitive towards things like indentation, redirecting which would cause errors. I enjoyed working on frontend, im very familiar with prototyping/mockup tools like figma and proto.io so using that on my own time to create efficient designs was fun and a good learning experience.

Implementation Discussion

- **Games**
  - Contains information of the game session, such as the players who are currently in one session with a unique id to separate itself from other game sessions, the amount of players and the order in which they are going, and the randomized deck of cards.
- **Users**
  - Registered users will hold a unique username, their own unique id along with their encrypted password.
- **Lobby**
  - Pregame lobby of the current user that displays their own properties, like their own username and their friends list.
- **Friends_list**
  - Each player will have their ID used to determine their list of friends to be displayed at their lobby screen so that they can add those friends to their party.
- **Party**
  - Record of members and the size of the party in the pregame lobby before entering a game and current game session the users are in.
- **Game_users**
  - This is the specific game session the player's in, it will hold information on the current game and track whose turn is next.
- **Cards**
  - Each card is assigned a unique id from the other cards that will be assigned a numerical value and a color for the card.
- **Game_cards**
  - This will hold information about the individual cards and which player currently holds what in their hand, where they make the selection to discard a matching card from their hand to deposit into the discarded pile.
- **Messages**
  - A display window of messages with timestamps from each player in a current game session or lobby

Requirements

	Requirement	✔ or ✘
User Authentication	Can create a new account	✔
	Previously registered users can log in	✔
	Can log in and log out	✔
	Only registered and logged in users can access lobby and a game instance	✔
Chat	Chat is enabled in the lobby for all users.	✘
	Chat is enabled in each game room for those users participating in an instance of a game.	✘
Lobby	Users can create a new game.	✔
	Users can join a game.	✘
Game Lobby	Users can join a game room and enter a number of players.	✔
	Users can leave a game.	✘
	Users can go back to the lobby	✔
Game Logic		✘
Data Flow	Uni-directional data flow.	✔

Code Quality	Well organized, clean, single responsibility principle, and well-formatted and readable code.	✔
	Classes, methods, and variables are meaningfully named	✔
	Methods are small and serve a single purpose	✔
Documentation	PDF is submitted with team members, link to github repository, problems we encountered, and discussion of challenges.	✔

Code Structure:

.idea	revert merge	3 hours ago
authorization	Auth updated, sessions implemented	2 days ago
bin	removing from myapp folder	2 months ago
config	removing from myapp folder	2 months ago
db	links for game rooms	4 hours ago
migrations	Auth updated, sessions implemented	2 days ago
milestones	Milestones 1,2,3, and 4	4 hours ago
models	Auth updated, sessions implemented	2 days ago
node_modules	Registration finished	22 days ago
public	Merge pull request #11 from sfsu-csc-667...	3 hours ago
routes	revert merge	3 hours ago
views	Merge pull request #11 from sfsu-csc-667...	3 hours ago

Challenges

Miscommunication:

- Uncoordinated team meetings
- Time scheduling
- Conflicts with branch merging

Code:

- Roadblocks, since members had assigned roles and different things to work on, if a member got stuck on a problem, it's difficult to ask another member for help as they would be unfamiliar with their structure and logic.
- Learning how to structure and link with queries successfully and get the necessary information needed.

What we learned:

We learned a lot about Postgres, Pug templates, managing GitHub, as well as the importance of creating some sort of team management/order. Postgres was new for most of us, so installing it was a little hard. Pug templates were also a bit new to some of us, so it was difficult implementing that. For backend, figuring out sessions, establishing connections, and authentication was difficult to navigate. We feel a little more knowledgable on all of the technology stacks we used though, although we can definitely become more familiar with them.