CSC 667 Spring 2018 Term Project Uno Documentation

Team-B Members: Alex Bautista, Alfred Quezada, Luis Almeida
Github Heroku Repository: https://git.heroku.com/term-project-team-b.git
Github Team Respository:
https://github.com/sfsu-csc-667-spring-2018/term-project-team-b.git

App Link: https://term-project-team-b.herokuapp.com/

Table of contents

Project Introduction and Overview

This program was meant to be a multiplayer, real-time, online game that should support any number of games simultaneously. Our game was to implement the game of Uno. Technical work was to add user authentication, chat, games states saved in database, and to support an arbitrary number of games.

Development Environment

Node – Open source JavaScript run-time environment that executes JavaScript code server-side. Designed for build scalable network applications.

Nodemon – utility for Node that monitors changes in your source and automatically restarts server.

Express – Open source web application framework for Node. Designed for building web applications and API's.

PostgreSQL – Open source object-relational database that uses the SQL language for data workloads.

Passport – Authentication middleware for Node. Designed to authenticate requests.

Socket.io – JavaScript library for web applications, that enable real-time, bi-directional web clients and servers.

Sequelize – For supporting PostgreSQL, and a promise based ORM for Node v4 and up.

Pug - Express template engine for using static template files on your application.

Scope of Work

Authentication - Users must be able to create accounts, login, and logout. Users can only access pages that they are authorized to.(***completed***)

Chat - Must be enabled in a lobby or page after login for all users. In a gameroom and each instance of a game. Observers allowed.(***partially completed***)

Static Pages - Implement static pages and files for application UI.(***partially completed***)

Game States - Stored in database. Dropped user can reload saved game when logging back in. Games states should also be updated in real time.(***not completed***)

Game lobby - Authorized users directed to this page to join games.(***not completed***)

Game room - Users start a game of Uno in game room.(***not completed***)

Arbitrary Number of Games - Support any number of concurrent games. Also, user can participate in multiple games at the same time.(***not completed***)

Command Line Instructions to Compile and Execute

-Install Node and PostgreSQL

-Navigate to src folder of project in terminal, or Open Terminal from src folder:

Setting up nodemon for automatically restarting server after every edit to local code:

npm i --save-dev nodemon

-Add this to script section in package.json

For quick way to start application in development mode. (Don't delete start script!)

"start:dev": "DEBUG=APP_NAME: * nodemon ./bin/www"

Dotenv

Set up dotenv to enable user specific environment variables in your development environment. Dotenv makes use of a `.env` file that stores key/value pairs of environment variables. This file must not be checked into the repository, as we do not want to expose potentially sensitive password strings in our repository - we accomplish this by adding the `.env` file to `.gitignore`:

```
npm i --save dotenv
echo ".env" >> .gitignore
touch .env
```

We will make use of the `.env` file later on in our setup.

To make use the environment variables defined in the `.env` file in our development environment, we add the following to `app.js` (towards the top of the file, before the app is instantiated):

```
if(process.env.NODE_ENV === 'development') {
  require("dotenv").config();
}
```

This takes care of loading the environment variables defined in `.env` whenever the `NODE_ENV` is `development` (Heroku sets `NODE_ENV` to `production` in the deploy environment).

-Create Database

createdb database_name

Now that we have created the database, we can add the `DATABASE_URL` environment variable to our `.env` file:

```
echo DATABASE_URL=postgres://`whoami`@localhost:5432/DATABASE_NAME >> .env
```

Migration Setup (sequelize-cli)

We want to be able to handle versioning of our database schema without a lot of work. Migration frameworks are built to allow us to write code to handle schema changes. For this setup, we will be using the Sequelize CLI (which also serves as an ORM, but we will not explore these features for this milestone).

Install the sequelize and sequelize CLI packages, and then initialize:

```
npm i --save sequelize
npm install --save sequelize-cli
node_modules/.bin/sequelize init
```

Now, run the migration:
`npm run db:migrate`

-Finally, enter next two commands in terminal to run

export NODE_ENV=development

npm run start:dev

Assumptions

Everything was perfectly clear.

Implementation discussion

GUI

Front end UI was implemented using the Pug template engine, that is default to Express. Along with Bootstrap and JavaScript, the static pages were created. Plain html and css with relative positioning was first attempt at making gamepage view, but was beyond the grasp with the given time, and resorted to constructing it using Bootstrap row and column technology. Lots of time spent figuring out how to handle card images dynamically onto a screen for a game. Researched many solutions, and testing them, until JavaScript was learned to print an array of card images onto the game page screen dynamically by passing an image through the DOM to create image elements. Was stuck for a long time, until realizing pictures have to be in public folder for app.js to access.

Authorization

In order to implement authorization into our app we went with passport and our strategy was base on passport-local. In order to save our user data we used postgress, authorization proved to be very tricky. The first step was to install passport using npm install. For hashing passwords we used bcrypt so that not even admins would be able to steal your password. The passport-local strategy is based on a username and password field to authorize users into a session. For our user name we used email, then we used a function we labeled look up that took a email and password, it would find the user in the database by email and then compare the password with the hashed password if it passed it would pass the user object into a callback function labeled done. This all happens in the auth/index.js file, the way we looked up the user in the database was in the db/users.js file. In order to use passport we had to implement a serialize and deserialize user function. The purpose of these functions was first to serialize the user into the session, meaning we would have a user object that would be accessible by the client throughout the web app during the session, and second

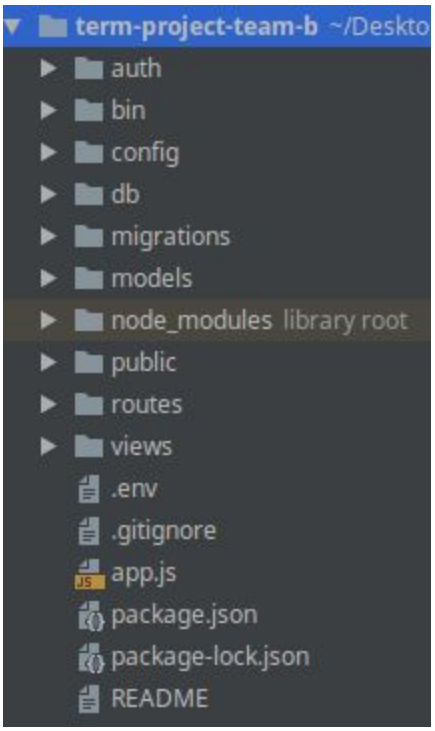
the deserialize function was used to get that info after initial serialization. Serialization and deserialization are simple functions inside the db/users.js file and we looked at the passport js website for instruction on how to implement them. In order to restrict access we created another file auth/requireAuthentication.js with a function that checked request.isAuthenticated() which is a passport function to check if a user has been serialized into the session and returns a boolean value. We call this function from routes that need authorization to access.

Chat

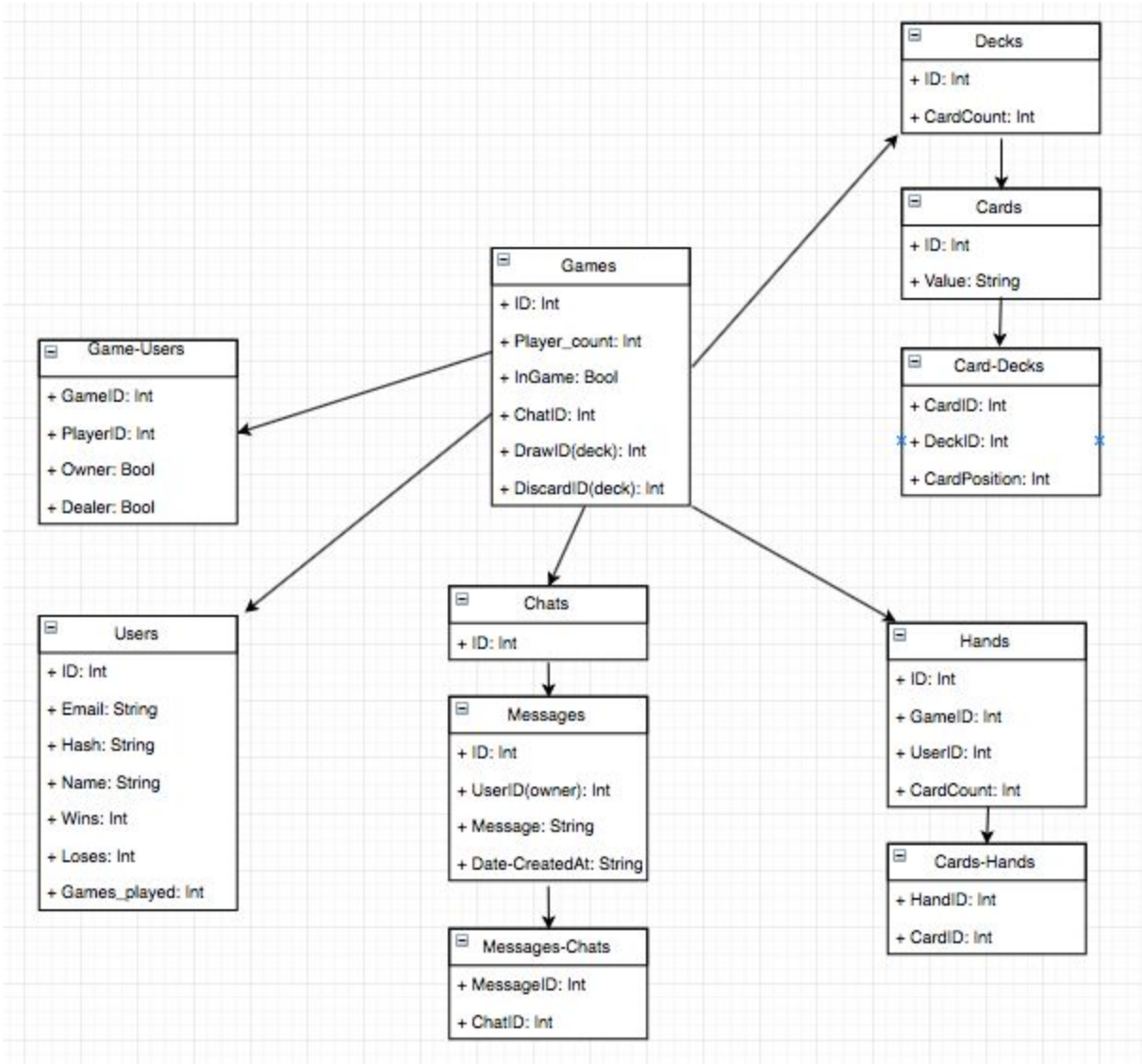
The goal of the chat was to create a live chat room for current players inside the game room and lobby(waiting area). We came across many issues during the development of this functionality, since it took a lot of time to get familiarized and learn about sockets which is something we haven't learned before. Sockets is a very confusing and difficult topic to understand. We used pug to get the input text and chat dialog area. We created a chat table inside the database in order to store messages and as well as the time the messages were sent. While sending messages on gamepage, for testing, messages were supposed to display on the terminal to make sure messages were being sent and we had no results. We tried connecting the socket server with a client but somehow there was still no messages being displayed on the terminal. I tried doing as much research as I could and get as much help as possible but I couldn't figure out the issue.This functionality still requires some how in order to have a fully working chat room.

Package structure

Contains directories named by Express. All JavaScript code are organized into subfolders named for their purpose. Main file is ('app.js'). Public folder has static pug files and images so app.js can access it. Migrations folder holds code to be run on the database for each migration, which should be everytime you deploy to server.



Database Diagram



Results and Conclusions

Besides a little background in HTML and CSS, every technology used for this project was new for all of us in our group. Thus, everything that we were able to get done, is what we learned. Along with many many challenges, that we still have yet to figure out. Having a small team, splitting up the tasks between front-end and back-end did not seem to work. An attempt at solving this, everyone tried to help out a little bit everywhere. Resulting in everyone learning a bit of everything. We got to briefly learn how to use Express, passport, socket.io, PostgreSql, and using Node. Front end guy learned JavaScript, pug, and bootstrap, and figured out how to print cards onto the bottom of a screen with a button. Everyone had to learn how Express works and how routing worked. Back end had to deal with setting up passport, and socket.io. Another big challenge, none of us were wizards at github, so a lot of

time was spent on solving git issues. Lots of challenges on getting things to work, which leaves a lot of future work to be done.