README

Uno Project

Team D

Alex Pena, Tahar Touati, Chris Riddle, Peter Hu

Heroku link: https://fast-beyond-10302.herokuapp.com/

Table of Contents

1. Overview
2. Technology Stack
3. Starting the development server
4. Migrations
5. Database Structure
6. Deploying to Heroku
7. Requirements
8. Challenges

Overview

This was a group effort to recreate the game of Uno in real-time multiplayer online. Our web app allows users to register, create, and join multiple games. A user can join many games and play multiple games concurrently. We store the game state in the database which allows users to join, leave and resume games without losing the state of the game. The game requires four players and starts automatically when the fourth player joins the game.

Technology Stack

- Database: Postgres
- Database Migrations: Sequelize
- Web Server: Node.js
- Web Framework: Express
- Front End: Vanilla javascript and CSS
- Sockets: Pusher.js

Starting the development server

First run `npm install` ,

Then before starting the development server, you must create a .env file in the root with the following fields:

```
DATABASE_URL=postgres://user@localhost:5432/uno
SESSION_SECRET=secret
```

Where the DATABASE_URL gives the postgres credentials that allow sequelize and pg-promise to access the database. The SESSION_SECRET is a passphrase required for passport to use the sessions securely.

To run the server locally, use the command `npm run start:dev` The server listens on port 3000. We cannot use the `npm run` command locally as it will break. This script is reserved for use by Heroku.

Migrations

To run the server, first you need to have an empty database created and that it matches the specificed database in the DATABASE_URL environment variable. Then to run the migrations to create the tables, run the command:

```
npx sequelize db:migrate
```

To undo all the migrations run

```
npx sequelize db:migrate:undo:all
```

The migrations will create several tables in the database, and it will seed some fake users and create a fake game.

Deploying to Heroku

To deploy to Heroku, first install the heroku-cli and login. Then after selecting your heroku project in the command line run:

```
git push heroku your-local-branch:main
```

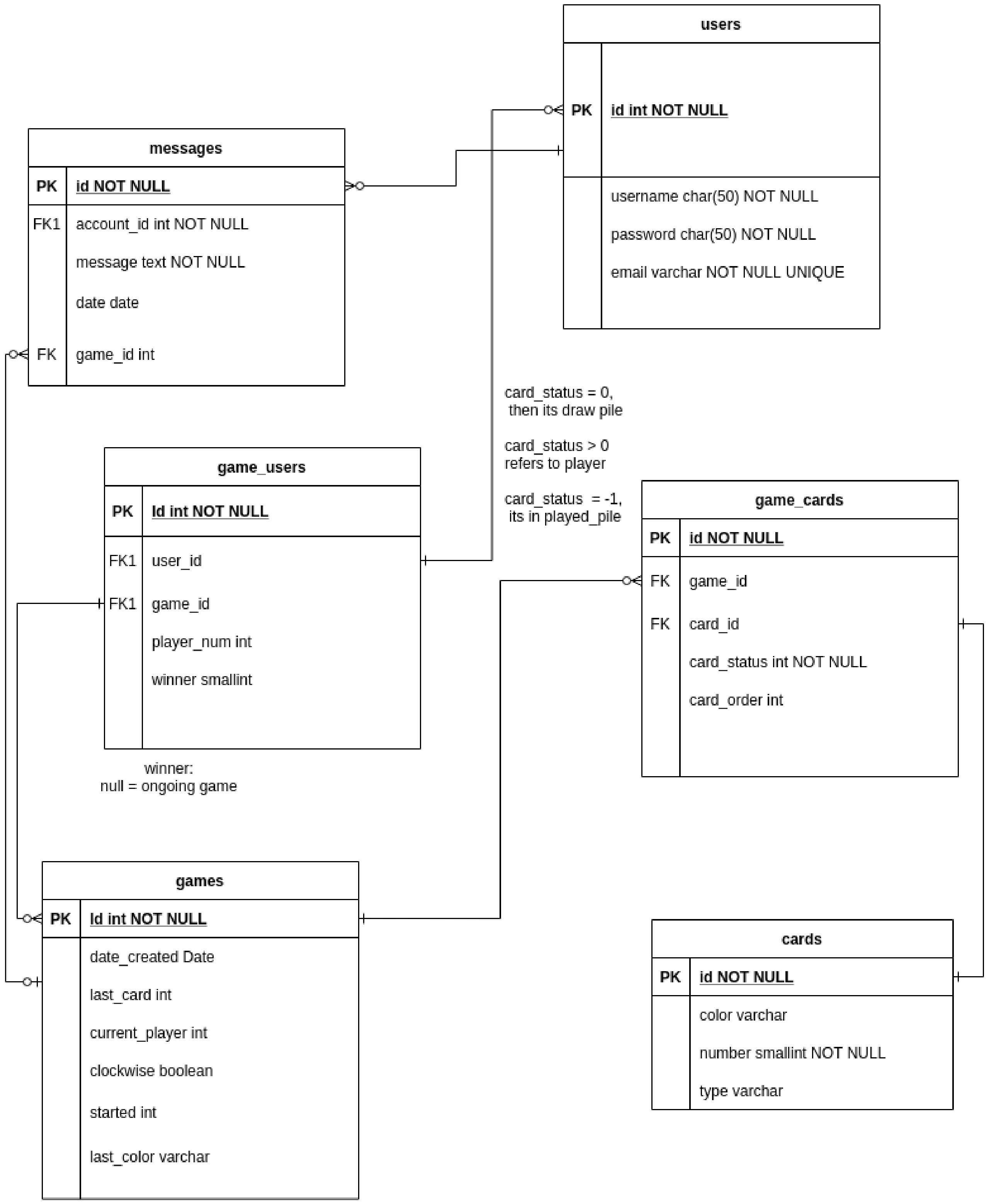
Heroku then runs the sequelize migrations automatically, but sometimes you need to undo the migrations and rerun them. In that case you need to run:

```
heroku run sequelize db:migrate:undo:all
```

then

```
heroku run sequelize db:migrate
```

Database Structure



Requirements

Category	Requirement	Completed
Registration	Users can create an account	X
	One account can be created with one email	X
	Passwords are hashed	X
Login	Registered users can login	X
	Login requires email and hashed password	X
Lobby	Users can create a game	X
	Users can join a game	X
	Users can resume a game	X
	Game list is updated in real time	X
Game Lobby	Users see number of players in game lobby	X
	Users can leave game	X
	Users can go back to the lobby	X
	Users are updated on who joins or leaves the game lobby	X
	Game starts automatically for everybody when fourth player joins	X
Chat	Users can chat globally in the lobby	X
	Users can chat privately within the game lobby and game to players	X
Game Logic	Player can draw a card only on their turn	X
	Player can play a card if it is valid and their turn	X

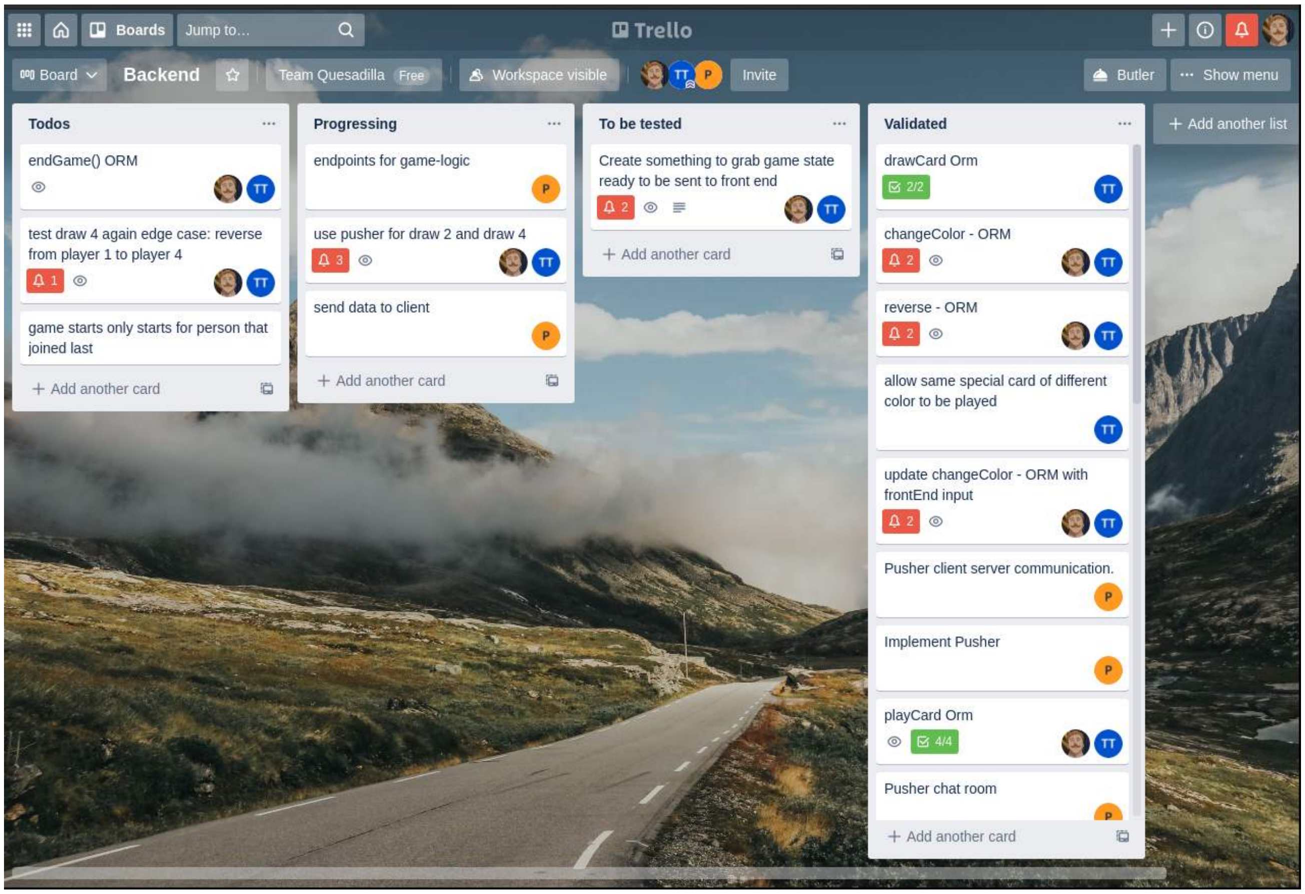
Category	Requirement	Completed
	Implemented special card effects eg. Draw 2, skip, reverse	X
	Game ends when a player has zero cards	X
Data Flow	Uni Directional Data Flow	X
Code Quality	Organized Routes	X
	Database Separated By Tables	X
	Built and maintained high performance, reusable, and reliable code	X

Challenges

Non-Technical Challenges

Colloborating efficiently: What was challenging for us was figuring out how to collaborate and split up work between the team. We had to learn each others technical background and also coordinate depending on how each team member wanted to contribute to the project. This was challenging because nobody on the team was an expert on creating a web server using this technology stack.

Communication (Trello): To coordinate tasks we used Trello. Trello helped us overcome task distribution so that our team members were aware of what other team members were contributing and also prevent two team members from incorrectly working on the same feature.



Todos
- endGame() ORM
- test draw 4 again edge case: reverse from player 1 to player 4
- game starts only starts for person that joined last
- Add another card

Progressing
- endpoints for game-logic
- use pusher for draw 2 and draw 4
- send data to client
- Add another card

To be tested
- Create something to grab game state ready to be sent to front end
- Add another card

Validated
- drawCard Orm 2/2
- changeColor - ORM
- reverse - ORM
- allow same special card of different color to be played
- update changeColor - ORM with frontEnd input
- Pusher client server communication.
- Implement Pusher
- playCard Orm 4/4
- Pusher chat room
- Add another card

Technical Challenges

Github: We urged each team member to utitilize branching and pull requests to avoid merge conflicts.

Migrations On Heroku: We had to learn how to run migrations on Heroku. We learned that Heroku runs migrations automatically when you push to Heroku, but the migrations were old versions so we had to manually rerun the migrations within Heroku so that it will have the correct database.

using socketio then using pusher: We had trouble implementing sockets using socketio, then we switched to pusher. Pusher was able to manage the sockets for us so we did not have to worry about creating and tieing sockets to users.

implementing change color card: This card was challenging to implement because it required user input within an interesting part in our game logic. To solve this we created four buttons in the UI that allowed a player to choose the color they want to use.

Testing: We used a special route that did not require login authentication so we could test routes quickly and efficiently with Postman.