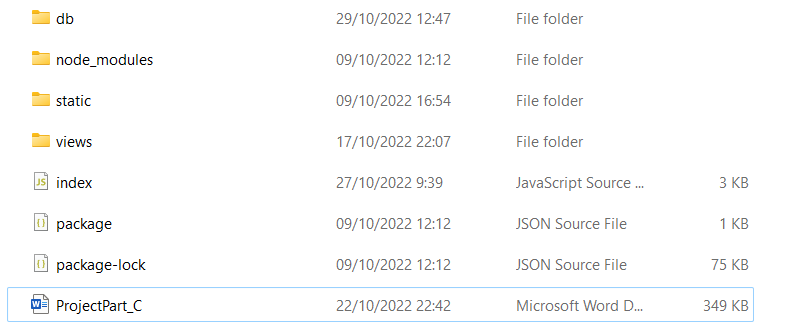
Project Requirements:

1. Project Folder structure



The root folder is as seen above. In the root folder there is the server (index.js) in which the packages and app routes are used. Both JSON packages are in the root folder which are used to transform and use the data in the DB. The DB configuration and CRUD functions are located in the db folder along with the csv files and CreateDB file to initiate the DB. The views folder is used by the view engine (PUG) to render the various html pages. In the static folder there is the CSS file and JavaScript client-side functions. Lastly there is the node\_modules folder used for all the MySql Server, express, csv, and node packages.

Packages needed to install:

npm init

npm install express mysql2 body-parser pug csv --save

npm i – save csvtojson

1. Handling Customer Requests

Client-side functions are located in bgride.js file. This file includes form validation functions, geo-location functions attained from the browser, timestamp function attained from the browser, and functions used in CSS. Both the geo-location and timestamp functions are used as input for the users form that is sent to the DB and stored.

1. Connection to DB and SQL Queries

**\*\*Please note that in order to allow the application to be checked the function updateDB() is disabled in the code. The responsibility of this function is to delete old records (more than 24 hours) from both the hitchhikers and drivers tables. UpdateDB() is a vital part of the functionality of the site due to the fact that we don’t want to match irrelevant users. In order to validate that the application matches according to the parameters described above please update the timestamp in the csv file to your current time. The records in the csv file have different geolocations to validate the geolocation constraints. \*\***

**Please initiate DB by creating tables and filling them by filling the URL in the following order:**

1. [**http://localhost:3000/CreateTable1**](http://localhost:3000/CreateTable1)
2. [**http://localhost:3000/CreateTable2**](http://localhost:3000/CreateTable2)
3. [**http://localhost:3000/Insert1**](http://localhost:3000/Insert1)
4. [**http://localhost:3000/Insert2**](http://localhost:3000/Insert2)
5. **Go back to** [**http://localhost:3000/**](http://localhost:3000/) **and begin**

Both DB configuration and connection are located in the files located in the db folder. Each Driver or Hitchhiker that registers to the app via the browser inserts their information and is added to the DBs after clicking submit. Important to note that the two DB tables(Hitchhikers and Drivers) are updated every time a user enters the main page according to a records timestamp(updateDB()). If the record is more than a day old the record is removed. This is done in order to ensure that the rides are authentic and allows the user to enter the DB more than once.

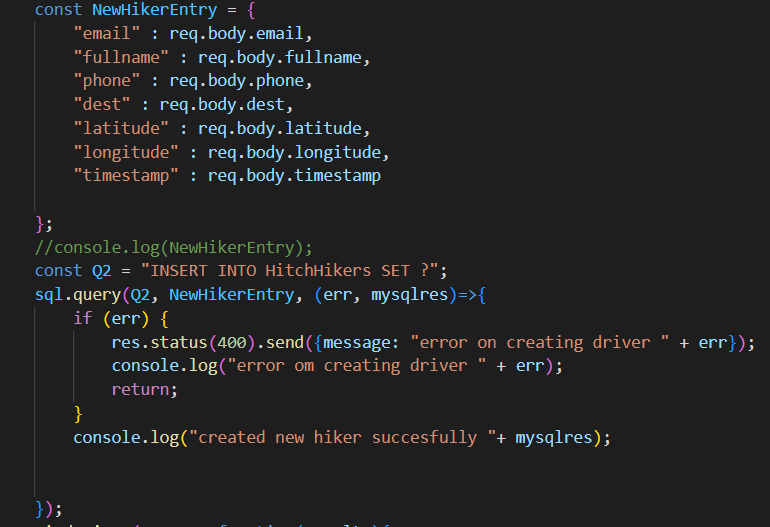
SQL Queries:

Delete: deletes old records in both tables

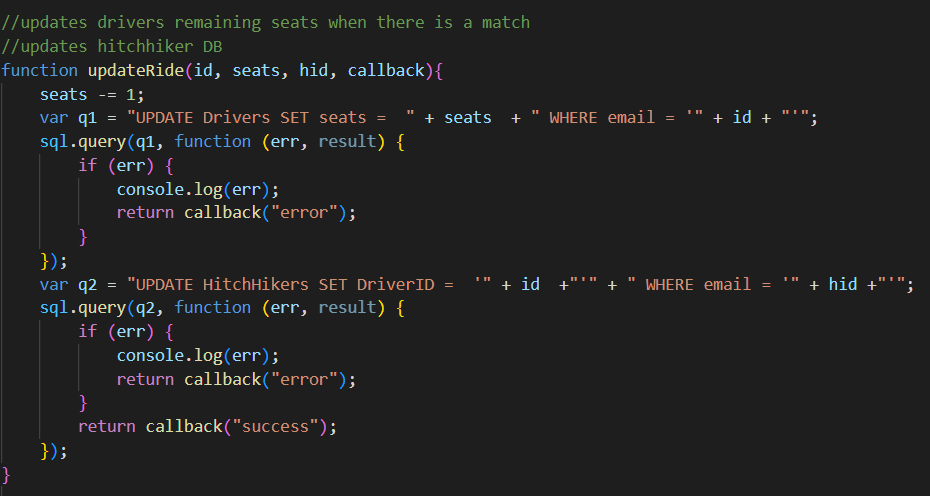
Text

Description automatically generatedSQL Queries:

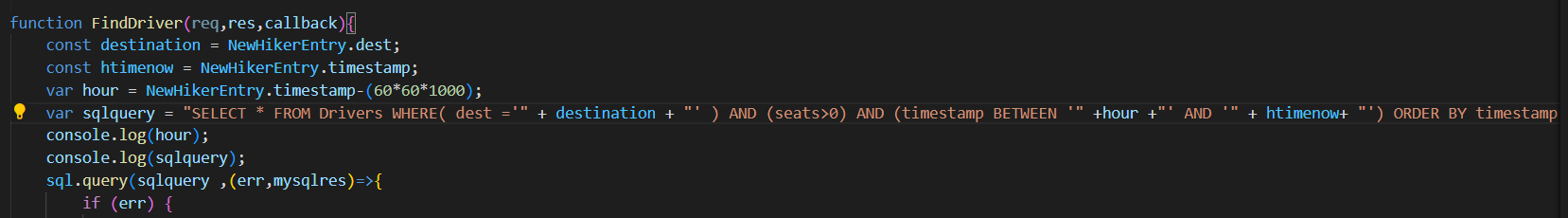
Insert: inserts hitchhiker information to DB, does the same for Drivers



Update: updates hitchhiker DB by adding the matched driver to DriverID field, updates Drivers remaining seats



Select: finds Drivers for Hitchhiker according to timestamp (within a hour) and final destination, later this list is transferred to function that filters relevant drivers according to their geo-location

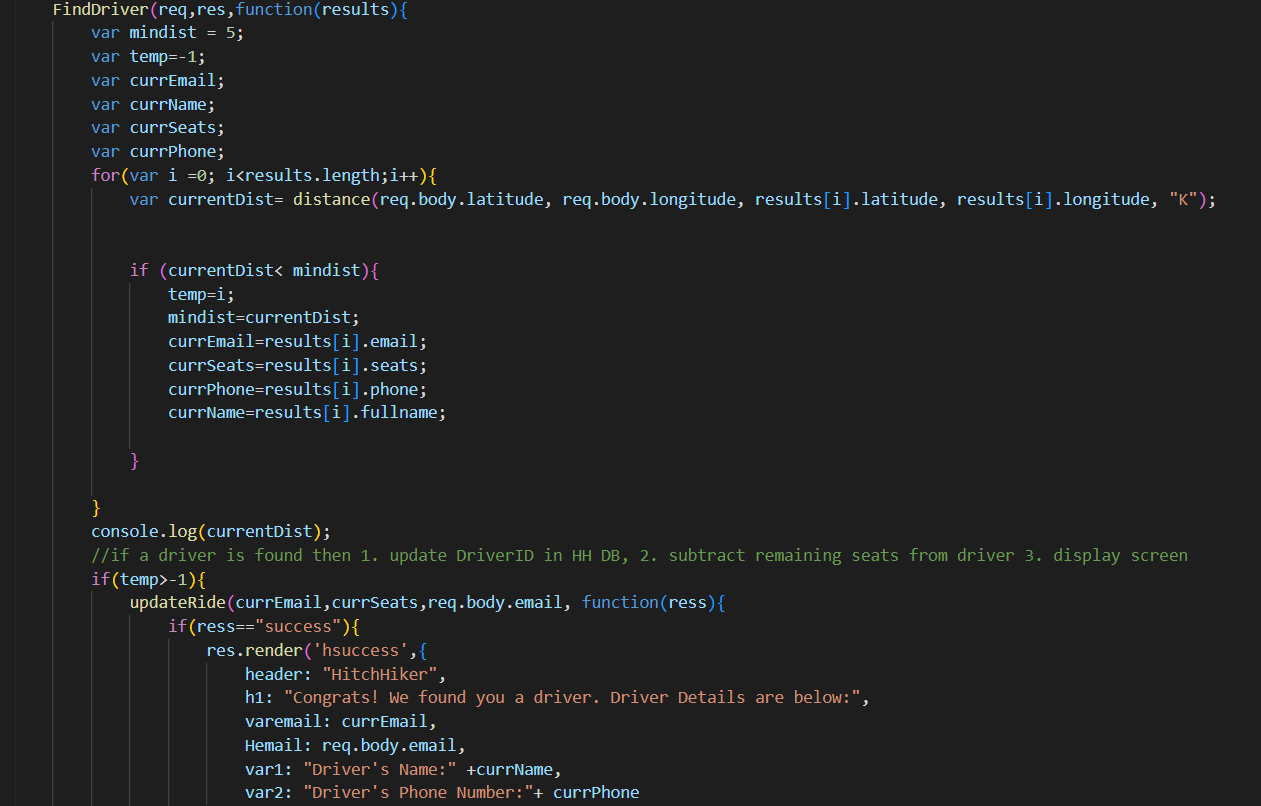


1. Implementation of forms

There are 3 forms in the website. One driver form and two Hitchhiker forms. The driver form is used to enter the new Driver’s information into the DB which will later be match to a hitchhiker based on parameters described above. Once the form is submitted the action changes the url and the server renders a html page that notifies the driver that his/her information has been saved and hitchhikers will contact. The other form pages are the sign in form and sign up form associated with hitchhikers. In the sign up form the hitchhiker must register for a ride and enter their information. In the sign in form the hitchhiker’s information is already stored in the DB but there is no driver linked. On submit for both forms the server calls the CRUD function to locate a driver if it exists. In the case that no driver is found then the hitchhiker is asked to sign in/try again later. In the case that there is a match the driver’s contact information is displayed with a option to cancel the ride. In the event that the hitchhiker cancels the ride the drivers seats is updated and the hitchhikers record is deleted.

1. Implementation of information processing functionality

As seen below one parameter for a hitchhiker to match a driver is based off of geo-location (latitude and longitude coordinates). The ‘distance’ function calculates the distance between the hitchhiker and the potential drivers. If the driver that is closest to the hitchhiker and under 5 km will be selected and matched to the hitchhiker. The latitude and longitude coordinates of the hitchhiker are taken from the browser and transferred to the server to calculate the distance.



Text

Description automatically generated

1. Assumptions
   1. The geo-location of a user is the starting point of the ride for both users
   2. Hitchhikers geo-location is based on the sign up form(sign-in doesn't update geo-location of hitchhiker or timestamp)
   3. Hitchhikers are responsible to contact the driver after each match, the driver is a passive user
   4. Drivers can have one ride per day, Hitchhikers can have more than one ride if they cancel ride (record in DB=Ride)
   5. Users are matched by a 5 km radius of current location
   6. Rides/matches have an expiration time of 1 hour
   7. Validation of student status is done by student email + bgu email validation
   8. The app has no monetary purpose/policy affiliated with its users and rides