Xpress Backend API

Technologies Used:

1. Node JS
2. PostgreSQL

Dependencies:

1.body-parser( to convert the custom json body objects which we get through the https get and post calls )

2. cors ( for cross origin support)

3. dotenv(For the environmental variables for the application which we can access them throughout the application.)

4.es6-promise ( to handle the promises returned)

5. express( a middleware for routing made easy )

6. isomorphic fetch( to use fetch api)

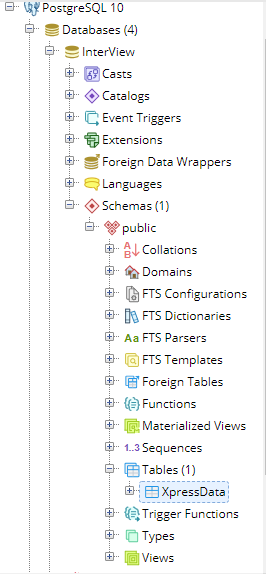
7.merge-json( to merge multiple json objects)

8.pg( to connect to the PostgreSQL database and execute queries on the database.

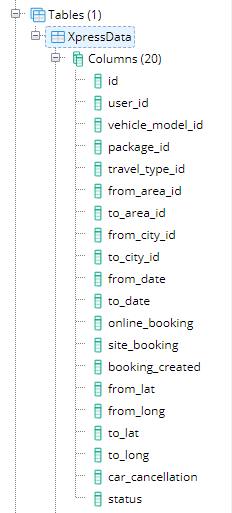
9. Winston (to log the server processes)

Setting up the Database for Testing:

Creation of table in the database:



Create a database and then a table in any schema of the created database. The structure of the table should be as follows:

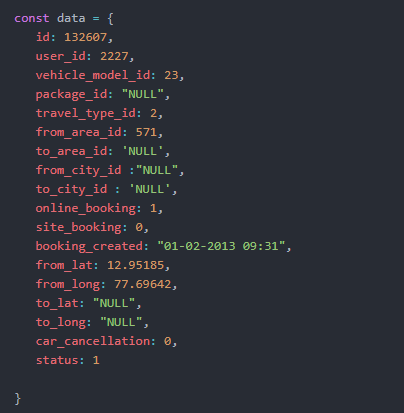


I have added a column “status” to track the status of the booking.

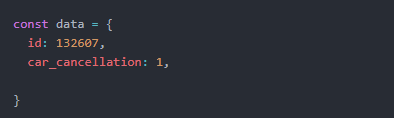
So I have divided the booking into four stages:

1. Booking created( status value is 1)

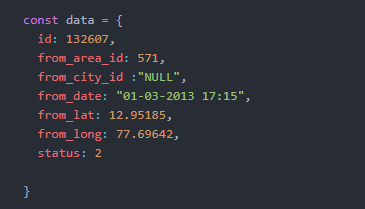
Data we put up on the database during this action is as follows:



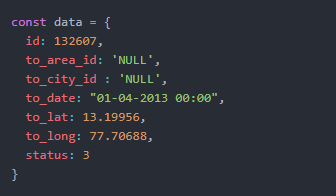
1. Ride started( status value is 2)



1. Ride Ended ( status value is 3)



1. Booking cancelled ( status value is 0)

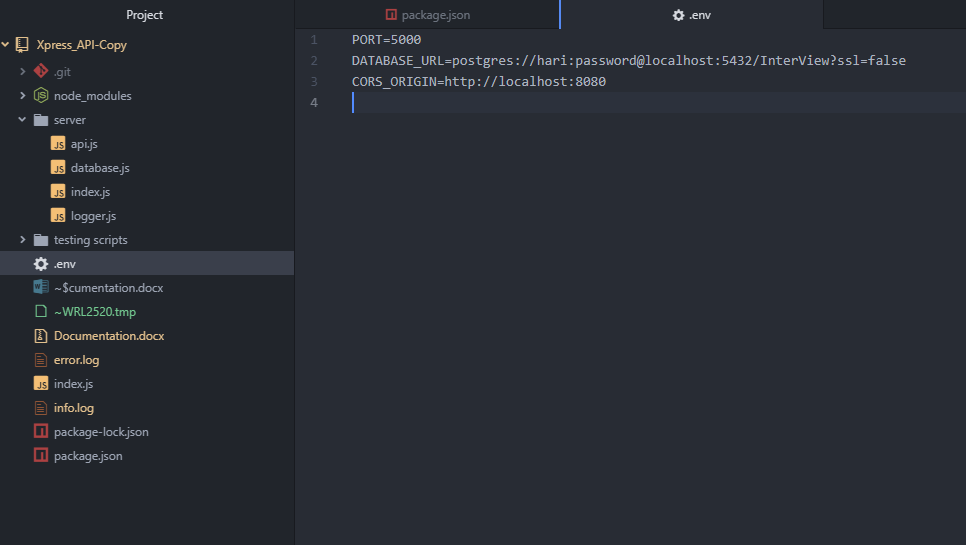


So, at a given point of time we can track the status of the booking with this new additional column.

The column “id” is the primary key for this entire table so its value is not null and unique and has limit of 6 characters.

Once the table is successfully created. We are good to test the API.

With few changes in the DOTENV(.env) File in the project directory.

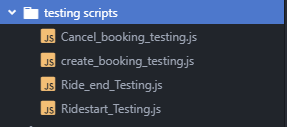


Choose the PORT on which you want to run the local server.

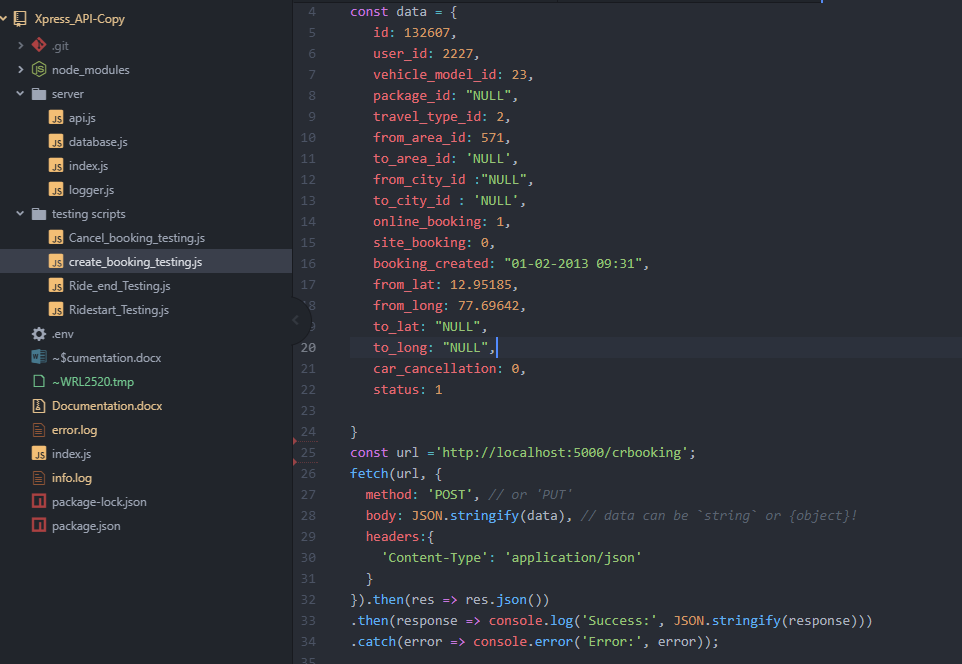
Replace the DATABASE\_URL with the username of the PostgreSQL database hari in my case. Also if your user doesn’t have a password let the password part be password itself or if you have a password replace the password with your own password and replace the Interview part with your database name which you have created earlier.

Now we are good to test the API.

Open the terminal or command prompt in the project directory and run $node index.js and boom the local server is up and running. Now Open and another Terminal or command prompt in the project directory and direct to the testing scripts folder with $cd testing scripts. There you can see 4 scripts in which each of the condition is simulated and a request is sent to the local server which is already running.



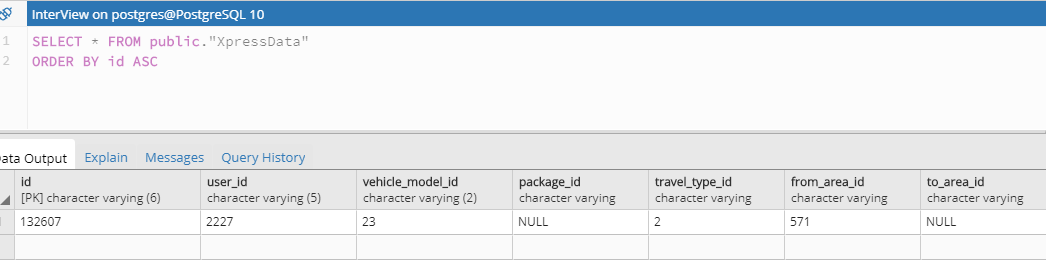
The four scripts are simulating four different type of requests to the API with four different end points. If we see the create\_booking\_testing.js



The data such as seen in the screenshot can be served during the time of booking creation. We then with the help of fetch API we are send a post request to the local server running at 5000 port and hits the end of crbooking with url <http://localhost:5000/crbooking>.

Then run $node create\_booking\_testing.js

Then a booking is created on to the database as you can see.



Then run on the other scripts $node Cancel\_booking\_testing.js which changes the status to 0.

Also you can run other scripts $node Ridestart\_testing.js and $node Ride\_end\_Testing.js and you can see the data being sent to the database and being updated through the fetch api requests with the end points of the API and they are following.

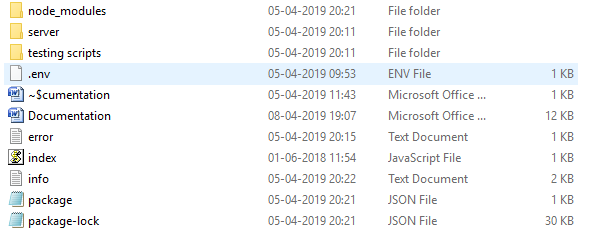
1. <http://localhost:5000/crbooking> at creation of booking.
2. <http://localhost:5000/canbooking> at booking cancellation.
3. <http://localhost:5000/ridestart> at start of the ride.
4. <http://localhost:5000/rideend> at end of the ride

All the data is sent in the form of json objects attached to the body of the each request.



In the snippet of the code above we can see how the data is attached to the body of the fetch api request to the url and the response is printed.

Understanding the project structure:

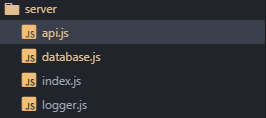


Index.js is the start of the application.

Node\_modules is the folder where all the packages are installed.

Testing scripts folder is where all the testing scripts are present.

Server is the folder where all the server scripting done



This folder has 4 files where index.js is the starting of the server file where the express server is setup, the routes are mounted, cors origin is configured, body parser is attached, Log requests and error handlers are attached and finally the server is started.

The other file logger makes uses of the Winston package to define the configuration of the logs and errors of the server.

The file api.js is the one where all the routes and end points are setup and the methods on the Database are called from here when a end point is hit.

The database.js file is where all the background action of querying database is done prior to that postgres client is initialised and connected to the database.

In a nutshell to Test this API:

1. Create a postgres database and a table with the same exact names of the columns following the same structure mentioned above.
2. Change the credentials of the database and the postgres user and also choose the port if you want to
3. Install all the dependencies in the package.json file this can be done with simple command “$node install “ from the project directory.
4. After installation of all the dependencies run “$ node index.js”

The server should be up and running.

1. Now open an another terminal or command prompt run “$node *file\_name.js”* the file\_name is to replaced by whatever testing script you want to excute.

Also other end points such as fetching of all the details of the booking by id or all the bookings are added later.

The nodejs can easily pull of the requirement of 200 requests per minute. Since single Epress.js process can handle more then 7000 requests per second.