**SERVER DOCUMENTATION**

Setting up the server: -

1. In the root directory, create package.json file using following commands:

npm init -y

1. We are using express, body-parser & cors as the basic package of this server. Now installing all the node packages:

npm install express body-parser cors mqtt sqlite3 swagger-ui-express

1. In package.json, edit the ‘scripts’ fields with following:

"start": "node server/index.js"

1. Create the server folder in the root and make a index.js file in it.
2. “Index.js” contain all the important routes to different api’s(end-points).

* ‘/api/registration’:- It contains the CRUD api of the all the users in registration.js file in ‘./routes/api/registration.js’.
* ‘/existing’:- It handles the post request from the OBD device to write vehicle data to the database. It is referenced to existingUser1.js file in ‘./routes/api/existingUser1.js’
* ‘/existing/topics’ :- It gives you all the subscribed (or active) topics.
* ‘/alerts’ :- It gives you all the alert message with their respective registration number (or topics) when it is sent to the server.
* ‘/api-docs’ :- It contains the swagger api documentation of CRUD api created at ‘/api/registration’. It is referenced to swagger.json file in server folder. The swagger doc can be edited from here.

1. The index.js is the entry point of the server resides in the “/server” folder. Such that server can be started using:

* node server/index.js
* nodemon server/index.js

1. The ‘registration.js file’ handles all the CRUD api’s in the following manner:-

* The Get api “/api/registration” returns all users registered with our System.

It can be accessed on http://localhost:2000/api/registration

* The Get api “/api/registration/{userId}” returns the registered users by their userId with our System.

It can be accessed on http://localhost:2000/api/registration/{userId}

* The Post api “/api/registration” posts a user in the user table with our System.

It can be accessed on http://localhost:2000/api/registration

* The Put api “/api/registration/edit/{userId}” posts a user in the user table with our System.

It can be accessed on http://localhost:2000/api/registration/edit/{userId}

1. We have created “db.js” in server folder which is used to initialized the database. Our database in placed in the ‘./server/db’ as registrationDB.db. Currently It has only user table for storing car data of respective user.
2. We are using “swagger-ui-express” npm package for creating the api docs for user registration.

We have created the doc in the “swagger.json” file which is placed in the server folder.

It can be accessed on “http://localhost:2000/api-docs”

1. Since our server should allow connection from remote machine (i.e obd device) so we have to set the ip in following manner:

app.listen(2000,'0.0.0.0').

How MQTT Works with our server & OBD Device

1. All our mqtt logics resides here: ‘./server/routes/api/existingUser1.js’
2. As soon as car enters the petrol pump the cc3200 device(obd) will send a post request on ‘http://localhost:2000/existing’ with a JSON body with registration number of that vehicle

{

“REG\_NUM”: “45114”

}

This will subscribe to this number as a topic of that user after verifying that the user is in the database.

1. Now, you can send mqtt message on this topic as JSON string like:

{

“fl”: “15%”, “tc”: “12L”, “rpm”: “15”, “speed”: “90”, “maf”: “20”, “ctemp”: “55”, “milage”: “55”, “alert”: { “type”: “high”, “source”: “hardware”, “state”: “true” }

}

These keys of mqtt message is explained below:

|  |  |
| --- | --- |
| KEYS | MEANING |
| fl | Fuel level |
| tc | Tank capacity |
| maf | Mass air flow |
| ctemp | Coolant temperature |

1. This data is updated regularly on the database of that user. It can be observed on the api.

http://localhost:2000/registration?reg\_num=45114

1. Now the timer for this topic is started as soon as we have a successful post hit on ‘/existing’ api. This is the time after which topic will be unsubscribed if no mqtt message is received in this period. This is defined as ‘period’ variable in ‘./server/routes/api/existingUser1.js’.

You can adjust this variable to set after how much time your topic get unsubscribed.

1. The alert object in the mqtt message is pushed to ‘/alerts’ api if we get the state=true.
2. The active topics are the topics whose timer are still running can be seen on the console in following format:

Topic1 45

Topic2 33

Topic3 11

These topics are pushed to the ‘/existing/topics’ as json object in real time.