

Node-RED Software System PoC

1. What is Node-RED?

Node-RED is an IoT framework, a visual programming tool for wiring and integrating hardware devices, APIs and online services easily. It runs on a wide range of edge, container, cloud, or premise platforms. It is flexible and powerful because it is built on JavaScript and Node.js, also because the programming part happens in a browser. We can create flows by wiring, dragging and setting nodes (internal or external) from a palette, some of them can be set by clicking and giving simple text as input, for example a message, others accept JavaScript code and some even HTML. The palette of nodes can be easily extended by installing new nodes created by the community and the flows we create can be easily shared as JSON files. With a single click, the application is deployed, and the result can be seen on the dashboard.

2. Troller – PoC

What is my application meant to do and why is Node-RED suitable for it?

In order to prove the usefulness of this framework, I have decided to create an application for renting kick scooters. I know that it is possible to connect to this type of vehicle with a Raspberry Pi, and there is a Github repository (<https://github.com/AntonHakansson/m365py>) which contains some python scripts for connecting to nearby located kick scooters, and retrieving data about them, which means all one would need is a monitor connected to the Raspberry Pi, to be able to see and use my application. This could be used for example by a company, which wants to make its vehicles available for the employees and handle the renting in an automated way. Also, to motivate people to use them, the application has a page with statistics showing why is this beneficial not just for them but even for our planet.

Application setup

In order to make the connection between the node-red part and the database easier, I have created a docker-compose file with a container for node-red and one for a mysql server. It is not the optimal version, it contains database credentials, but it is a quick way to start and stop the whole application.

The database has a table for *kickscooters* (*id, kickscooterName, stateId*), *states* (*id, stateName*), *users* (*id, usingKickscooter, roleId*), *roles* (*id, name*) and *rents* (*id, userId, kickscooterId, rentDate, returnDate, currentlyRented*) and because I did not have access to some kick scooters, I inserted manually a few rows in the tables.

```
docker-compose.yml
1 version: '3'
2 services:
3   mysql:
4     image: mysql:5.7
5     container_name: mysql
6     restart: always
7     environment:
8       MYSQL_ROOT_PASSWORD: root
9       MYSQL_DATABASE: Trolling
10      MYSQL_USER: trolling
11      MYSQL_PASSWORD: trolling
12    volumes:
13      - ./mysqlVolume2:/var/lib/mysql:rw
14      - ./database/create_tables.sql:/docker-entrypoint-initdb.d/create_tables.sql:ro
15  node-red:
16    image: nodered/node-red
17    ports:
18      - "1880:1880"
19    links:
20      - mysql
21    volumes:
22      - ./node_red:/data/
23  volumes:
24    node-red:
25      mysqlVolume2:
26
```

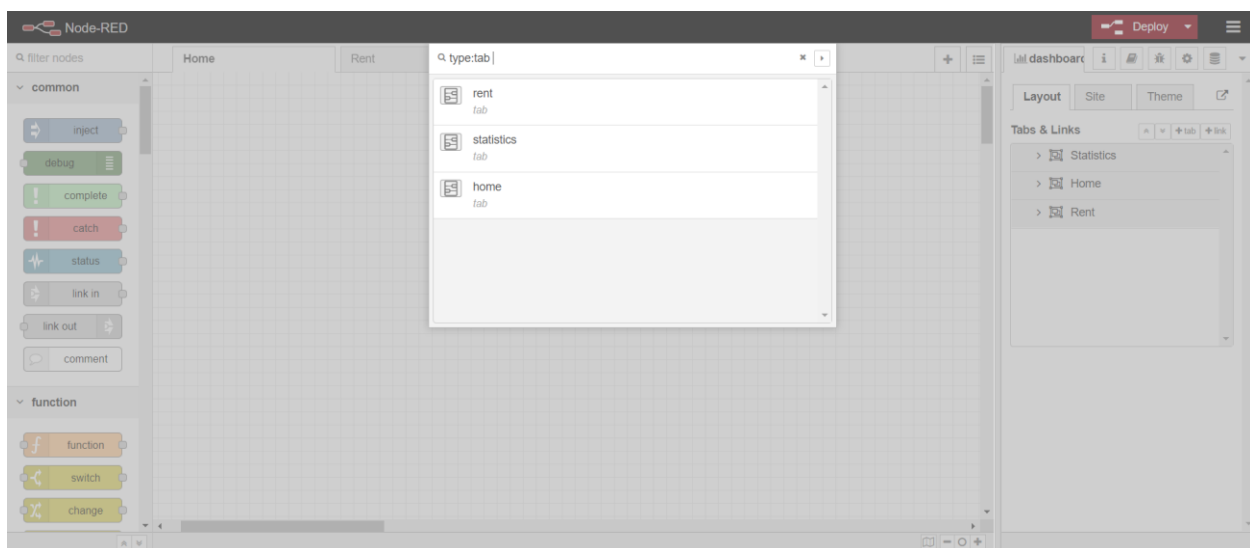
Used node sets

On the node-red part, in addition to the internal nodes, I have also installed some external ones. I used the following nodes:

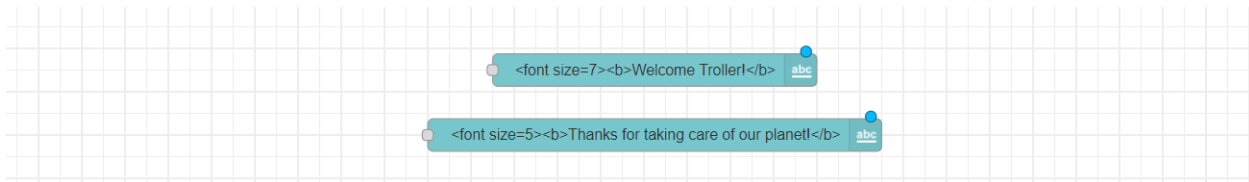
- *node-red*
- *node-red-dashboard*
- *node-red-contrib-dashboard*
- *node-red-node-mysql*
- *node-red-node-ui-list*

Tabs

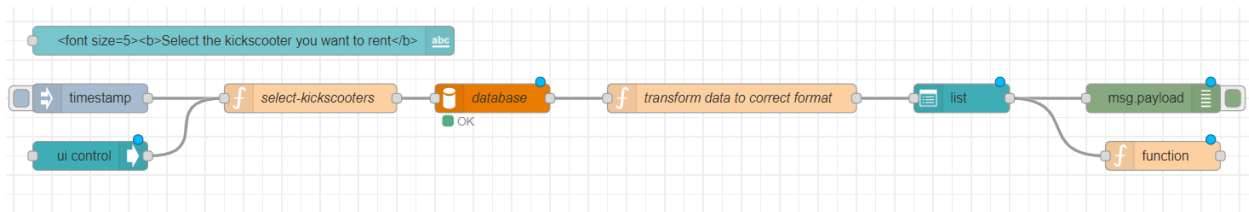
There are three tabs, one for the home page, one for the renting functionalities and one for the statistics.



The flows are grouped as functionality sets. The first group is for the **Home** page, which actually is not exactly a flow, the tab contains only two text nodes with HTML for welcoming the user. If we double click on one of them, a window with settings will open.



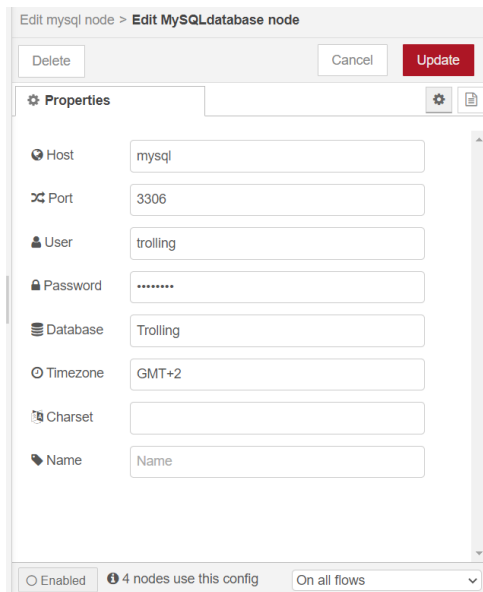
The **Rent** tab has four flows. The first one is for listing the available kick scooters.



First of all, we have a text node for a title. Two triggers are available for listing the kick scooters, a timestamp-based trigger which can be set to fire repeatedly, but in this case, it works by clicking on the little square which is in front of the node. The second trigger is a *ui control* node which is activated when we go to the Rent page or when we refresh it. The *select-kickscooters function node* contains JavaScript code with a query command for retrieving the available vehicles. This is sent to the database by the following node.

```
1 //SELECT Query
2 msg.topic = 'SELECT kickscooters.id, kickscooterName, stateName FROM kickscooters ' +
3             'JOIN states on states.id=kickscooters.stateId where stateName="available"';
4 // msg.topic = 'SELECT * FROM states;
5 node.warn(msg)
6 return msg;
```

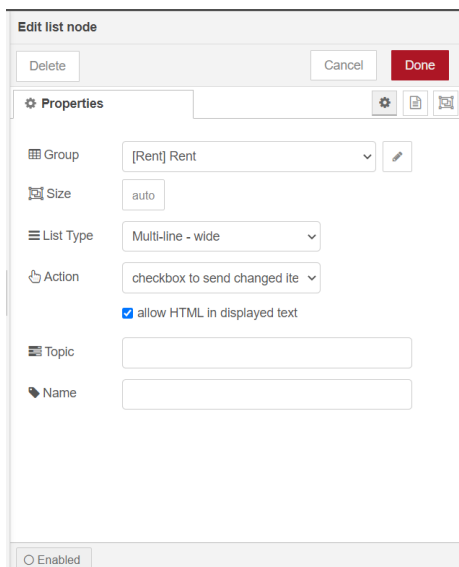
The database node has to be configured by giving information about the used mysql server and the credentials.



The next one is again a function node for transforming the received data into a form which is acceptable by the list node and into a more visually pleasant one.

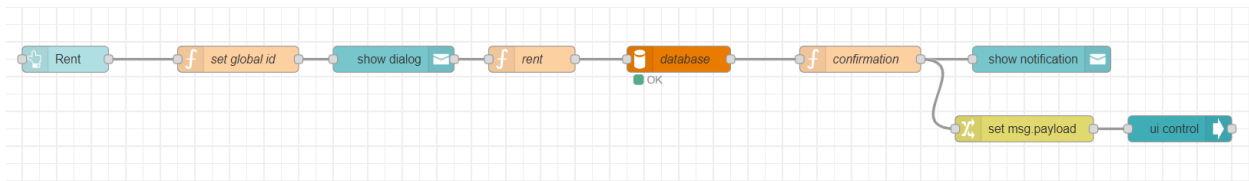
```
1 options = []; //create empty array
2 for(let i = 0; i < msg.payload.length; i++) {
3   let row = msg.payload[i]; //get the row
4   let opt = {}; //make new opt object
5   opt["id"] = row.id;
6   opt["title"] = "<font size=5 color='green'><b>" + row.kickscooterName + "</b>";
7   opt["description"] = "<font size=3><b>state: " + row.stateName + "</b><br/><b> id: " + row.id + "</b>";
8   options.push(opt)
9 }
10 msg.payload = options;
11 return msg;
```

The list is customized as follows:

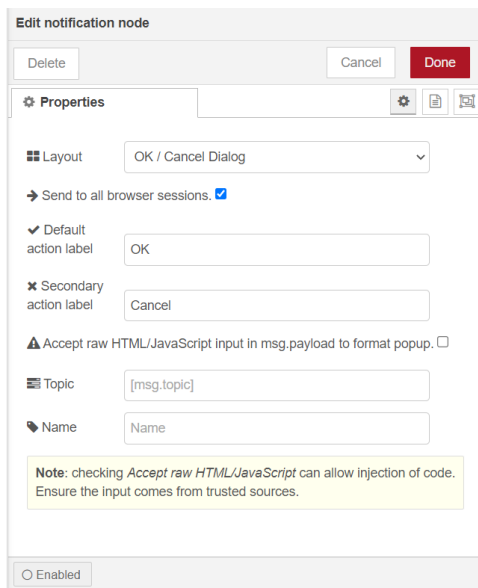


One branch from the end, the green one is just for debugging purposes, and the other one, when an item from the list is checked, sets as global variable the vehicle's id.

The next flow is for renting the selected kick scooter.

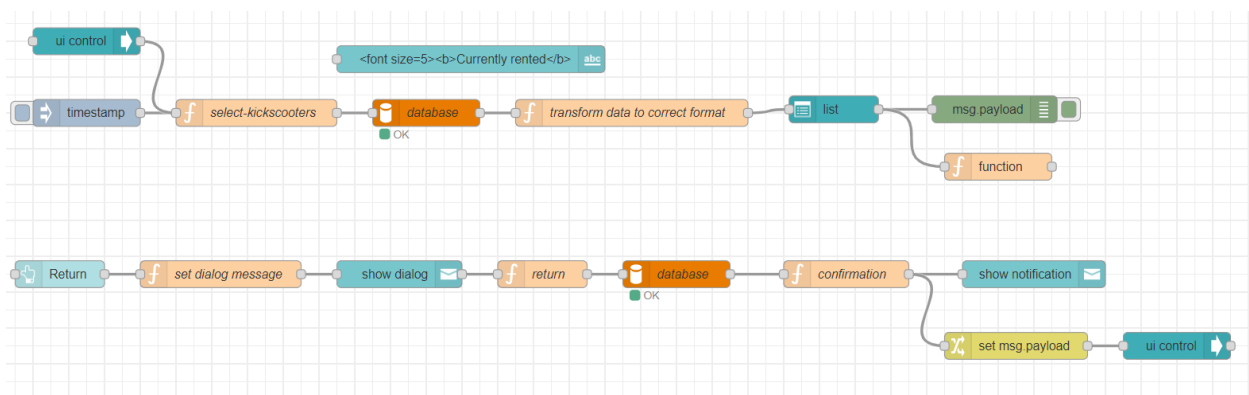


First we have a button node, which after is clicked, triggers a function node which sets the *msg.payload* to a question related to confirmation, which will be showed by the next node, the *show dialog* one. This will generate a popup window with OK/cancel buttons.

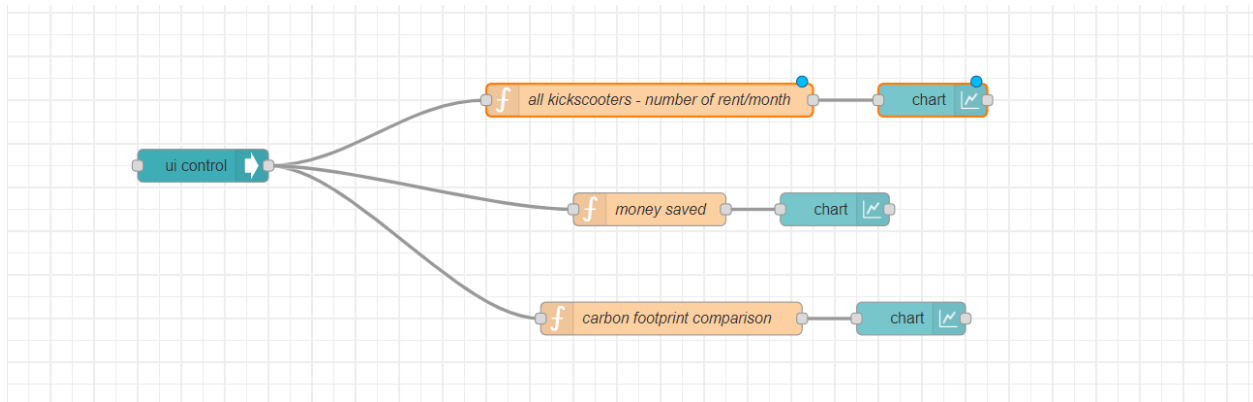


If the answer is OK, the *rent* function node will send a SQL command to the database in order to be able to rent the vehicle. The *confirmation* function sets the *msg.payload* to a message which will be shown as to the user in a popup window. The *msg.payload* will be then set to "Home", and the *ui control* node will take us to that page.

The next to flows are almost the same, except instead of the available kick scooters, the rented ones will be shown, and one will make possible the return.



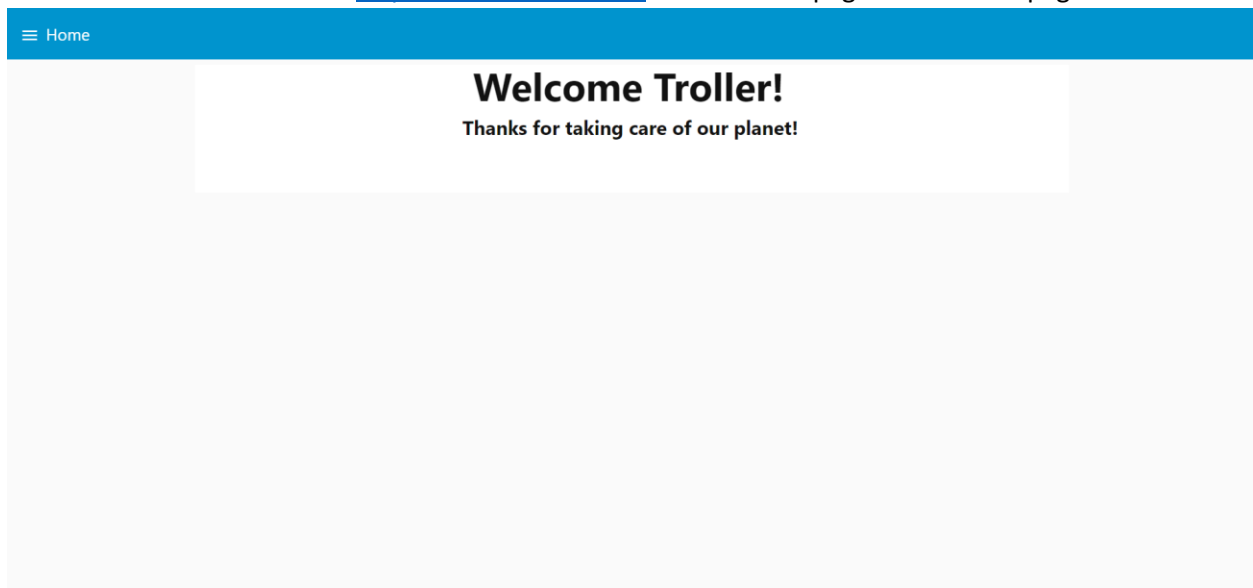
On the **Statistics** tab we can find a flow for three charts.



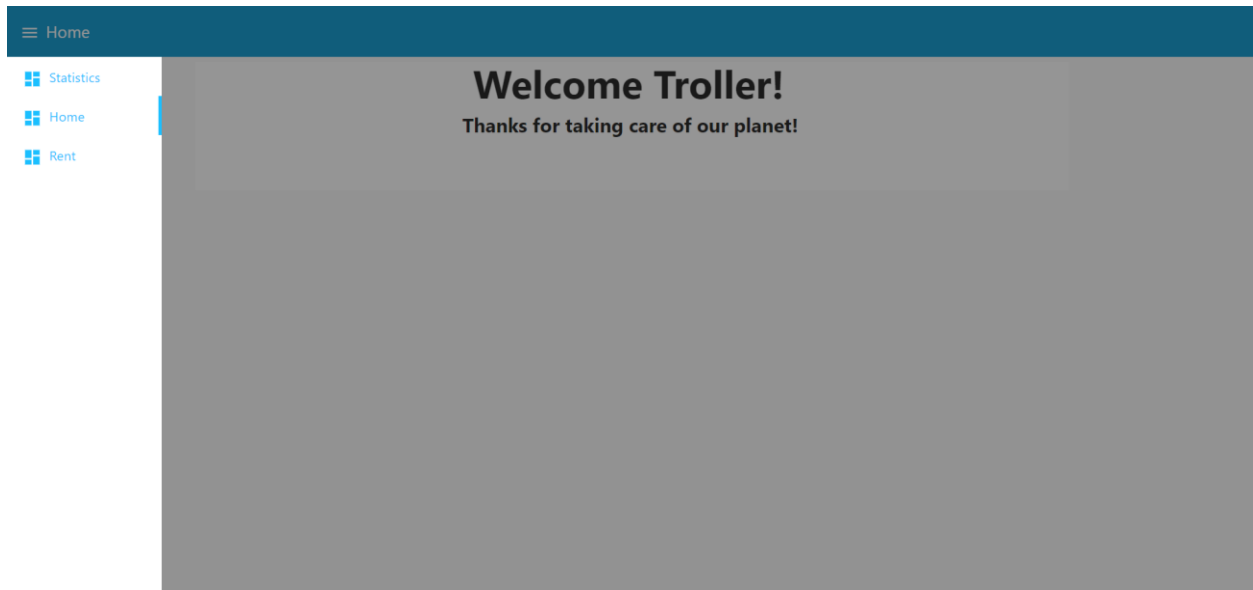
One for the number of rented kick scooters per month, one for how much money we have saved comparing to if we would've travel with bus, taxi or personal car, and one for carbon footprint comparison regarding different types of fuels and vehicles. Since I do not have real data, the function nodes contain only hardcoded values.

Result

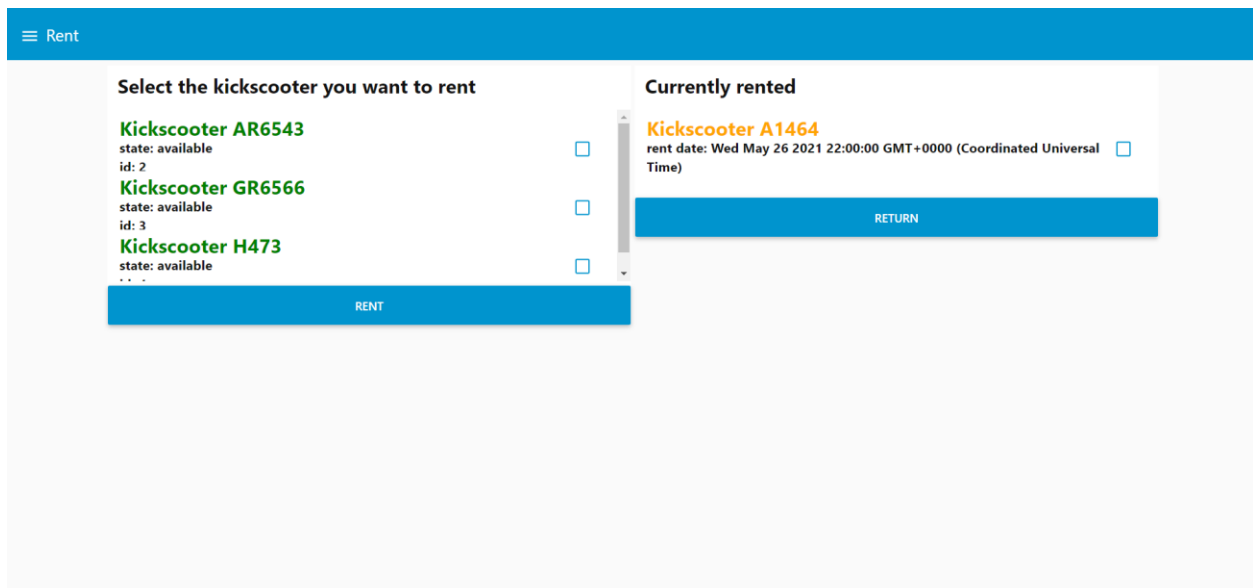
We can see the result on the <http://localhost:1880/ui> link. The first page is the Home page:



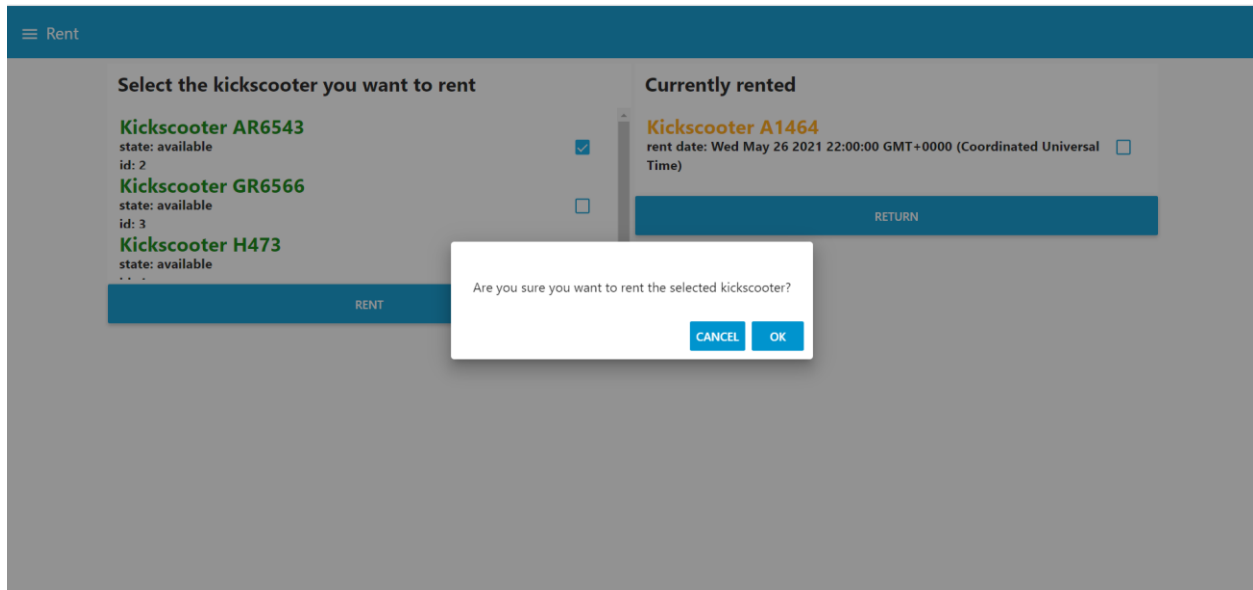
On the left upper part it's a menu button for navigation:



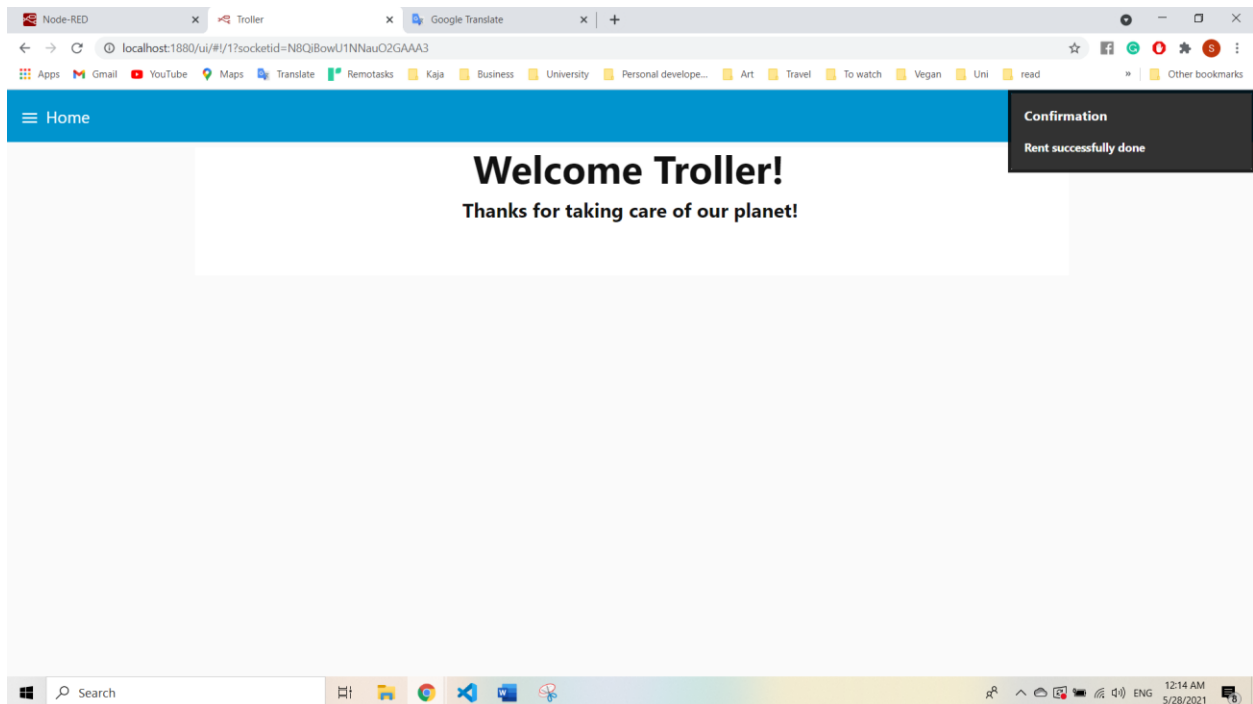
If we go to the Rent page, one list will be shown with the nearby located available kick scooters, and one for the rented ones.



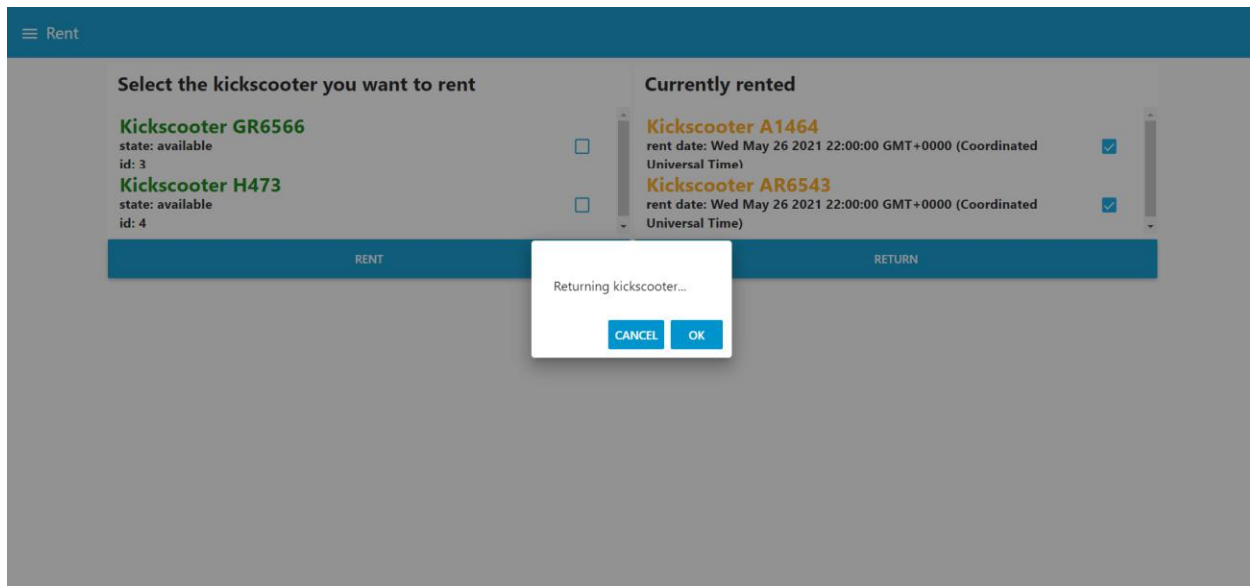
If we select a kick scooter and click on the RENT button, a confirmation question will pop up.



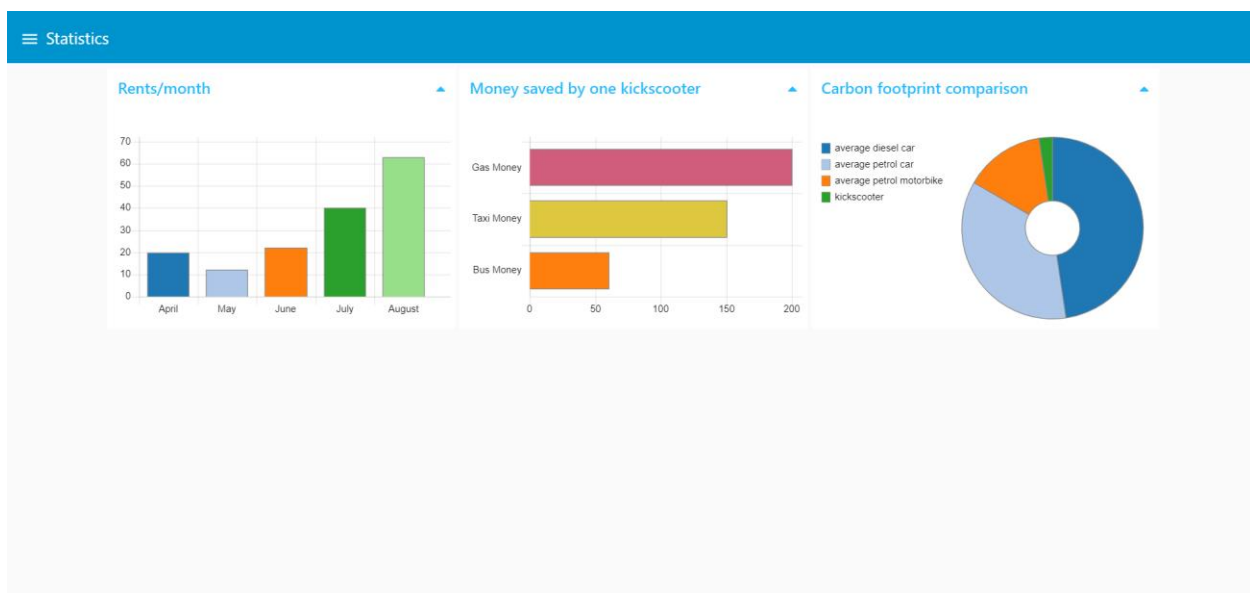
If we click on the OK button, a confirmation pop up will be shown while we will be taken to the home page.



If we want to return a vehicle, the procedure will be similar, only with different messages.



And finally, if we go to the Statistics page, we will be able to see three charts.



JSON

The flows can be saved in JSON format, and which is even nicer is that we can connect directly to a repository from Node-RED and push the changes.

Further ideas

In order to make the process of renting completely automated, an idea would be to lock the kick scooters until they are rented from the application. This can be done because they have an option for this which can be set via the Raspberry Pi. One other idea is that the application should have different user roles, and for example in the case of an administrator, one could manage and check the state of the

vehicles. He/she could see the battery capacity, battery voltage, trip distance etc. And last but not least, further statistical charts could be added.