

CS5200 Project Report

Bike Sharing

Team 21

Aakanksha Duggal, Lakshmi Priyanka Chapala and Shubhi Saxena

Abstract

Bike Sharing is the modern approach for traditional bike rentals that eradicates the entire process. From the membership, picking up the bike, dropping and rental everything is monitored using the application. This is a very simple way of renting a bike where the user can pick up the bike from the nearest dock and can drop it to any other available location across the city. There are more than 500 bike sharing apps around the world and own thousands of bikes. This not only allows one to get affordable bikes, but also has a major role in reducing the traffic and keeping the environment safe. Besides being so user and nature friendly, the features of the dataset for bike sharing systems are very interesting to research on. Contrary to the transport services like the local trains or buses, the time taken from destination 1 to 2 is explicitly recorded by these systems. This special quality of the bike sharing systems turns it into a Virtual Sensor Network that can be further used in monitoring the mobility across the city. Also, many important events can be detected using this data.

Introduction

Inspired by the idea of bike sharing, its application and significance in the modern world, we try to build a simple application of bike sharing. To achieve this we follow the MERN stack (Mongo, Express.js, React.js, Node.js). Each technology in this stack, serves our purpose to build an application as follows:

- MongoDB : Document based database (NoSQL)
- Express : A web Application framework Node.js
- React : A javascript front-end library for building user interface.
- Node.js : JavaScript run-time environment that executes JavaScript code outside of a browser (such as a server).

In addition to this, we use a database hosted on cloud using MongoDB Atlas.

The dataset used for jumpstarting our project is available publicly on Kaggle. Since the dataset is about 5Gb in size, we decided to use only 10% of it to perform simple CRUD operations in our application.

The user roles are defined as per project guidelines provided, with an admin user which can manage CRUD operations for all users. All other users are either customers or subscribers.

Dataset

The dataset has been taken from [here](#). This dataset has been shared by Divvy. There are a total of 23 columns in this dataset:

1. trip_id : ID attached to each trip taken
2. year : year
3. month : month
4. week : week No.
5. day
6. hour
7. usertype : "Customer" is a rider who purchased a 24-Hour Pass;
8. "Subscriber" is a rider who has a Subscription Plan - Monthly / Yearly / Quarterly
9. gender
10. start time : day and time trip started, in CST
11. stop time : day and time trip ended, in CST
12. trip duration : time of trip in minutes
13. temperature
14. events
15. from_station_id : ID of station where trip originated
16. from_station_name : name of station where trip terminated
17. latitude_start : station latitude
18. longitude_start : station longitude
19. dpcapacity_start : number of total docks at each station
20. to_station_id
21. to_station_name
22. latitude_end
23. longitude_end
24. dpcapacity_end : number of total docks at each station

Methodology

1. UML Diagram

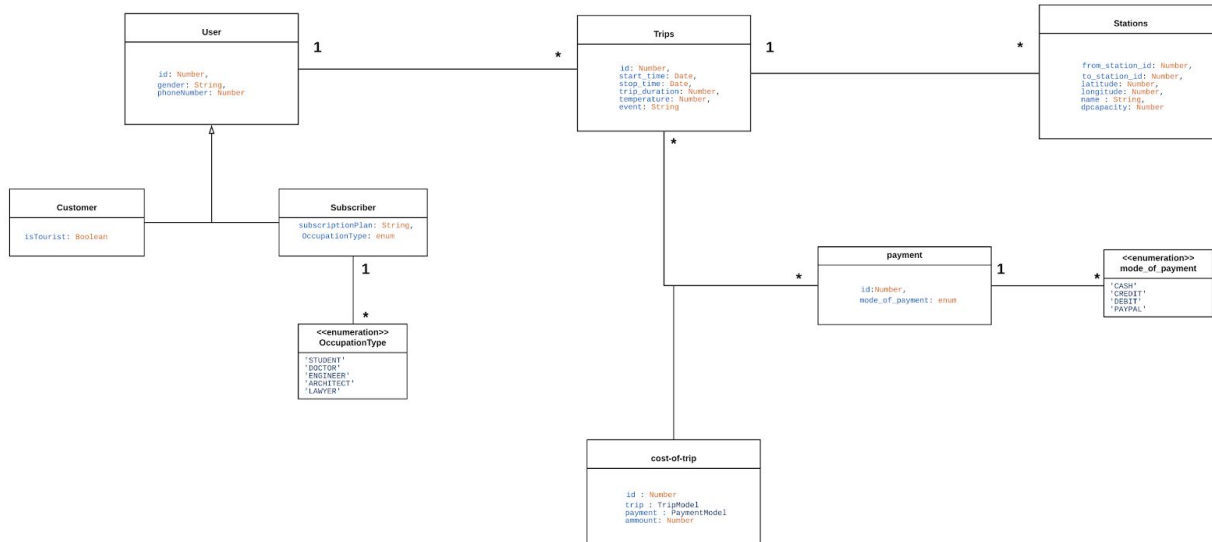


Figure : UML Diagram for Bike Sharing System

The Uml diagram above, defines the relationship between various entities of our schema. The diagram was refined to the above state taking into consideration various types of relationships like inheritance, aggregation, composition etc. We follow the above uml diagram to build our schema. Reiterations on the Uml diagram were made to get rid of redundant associations, entities and attributes. The cardinality between participating relationships were given special attention, to capture the essence of the bike sharing application.

2. Modelling Database (MongoDb)

To start off with the project, the first part is to create a schema. We create here for our project a new database called 'project' and deploy this as a cluster on Mongo Atlas. MongoDB Atlas is a cloud database to provide scalability and availability for applications. The connection links to hosted database are

Username: priyankachapala

Password: 2004Priyanka

1. Connect with Mongo shell
./mongo "mongodb+srv://project-ssy8s.mongodb.net/test" --username priyankachapala
2. Connect with Application
mongodb+srv://priyankachapala:2004Priyanka@project-ssy8s.mongodb.net/test?retryWrites=true
&w=majority

3. Connect with Mongo Compass

mongodb+srv://priyankachapala:2004Priyanka@project-ssy8s.mongodb.net/test

As suggested in the UML diagram, we need to create various models. The following are the models that have been created :

- Users : users.model.server.js , users.schema.server.js
- Subscribers : subscribers.schema.server.js
- Customers : customers.schema.server.js
- Trips : trips.model.server.js , trips.schema.server.js
- Payment : payment.model.server.js , payment.schema.server.js
- Stations : station.model.server.js , station.schema.server.js
- Cost-of-trips : cost-of-trip.model.server.js , cost-of-trip.schema.server.js

After creating our schemas and models, we broke down our Daos into smaller daos, to maintain modularity and simplicity in our code. The Daos help us perform CRUD operations and maintain abstraction in our project. The following Daos were created :

- Cost-of-trips.dao.server.js
- Management.dao.server.js
- Payments.dao.server.js
- Stations.dao.server.js
- Trips.dao.server.js
- user.dao.server.js

3. Express.js

Since we use Express.js for REST APIs, it is the most popular Node web framework. It is easy to configure and customize. It includes various middleware modules which can be used to perform additional tasks on requests and responses. In the next step we work on creating Controllers to manage GET/POST mapping between server and database. The following controllers were created:

- Costoftrips.controller.server.js
- Stations.controller.server.js
- Trip.controller.server.js
- User.controller.server.js

We also create the server.js file using Express.js to maintain all routings to different urls.

4. React.js / Front End

Being a Component based framework javascript library, we use it to create our basic frontend (Single Page Application). Since react is only considered for rendering the DOM (Document Object Model), it is used with other libraries to maintain states like redux or react router. React creates a virtual DOM in memory data-structure cache, computes the resulting differences and then updates the browser's DOM efficiently.

As per the UML , we created various pages to access the dashboards for various userTypes.

Subscriber, Customer and Admin.

The pages directory contains folders specifically related to each function.

- 1) Sign In -- Homepage
- 2) adminDashboard -- Admin Dashboard
- 3) Customer -- Customer Dashboard
- 4) Subscriber -- Subscriber Dashboard
- 5) Stations -- Stations Dashboard
- 6) Users -- CRUD functionalities of Users
- 7) NewUser -- registering as a new user

Each folder contains an index.js file which displays the dashboard.

To route the pages in a particular order we have used a routes/index.js file which helps for proper navigation of pages. This page further imports `react-router-dom` library from routes/Route.js file for navigating the HTML pages in proper order.

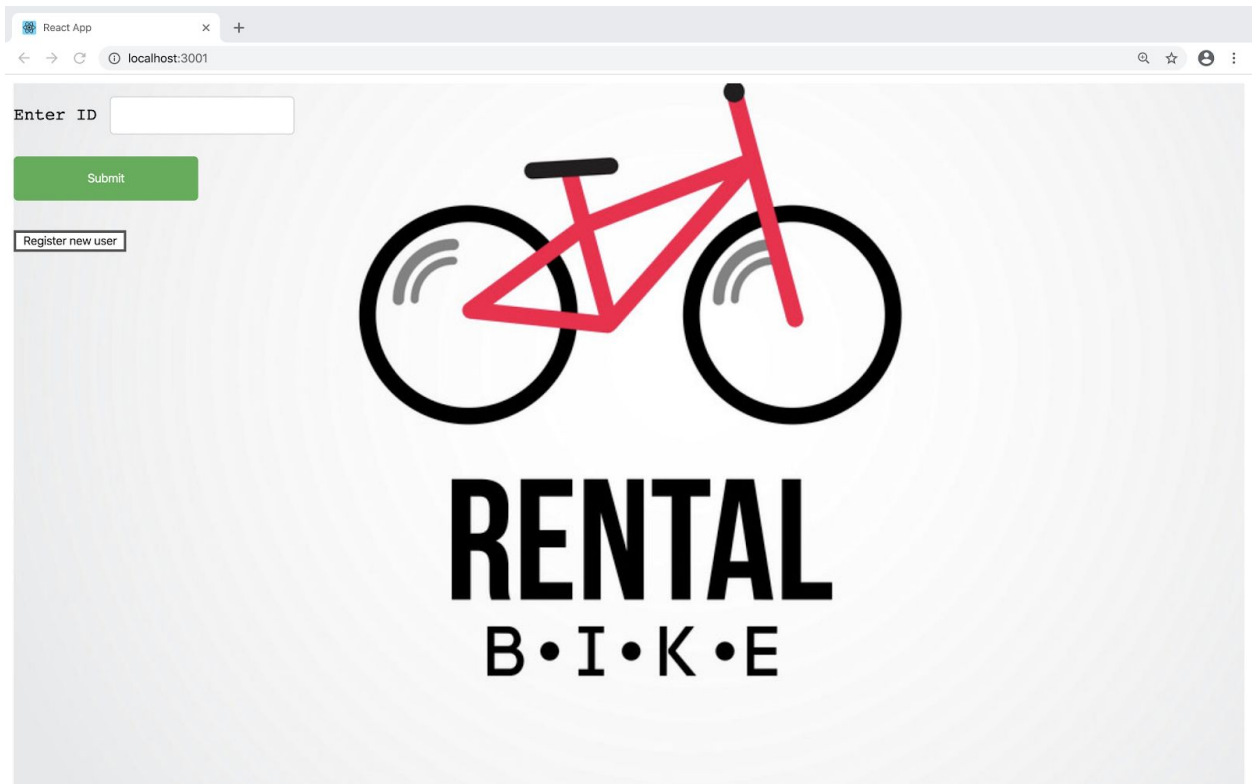
Data is fetched in the React from the database and hence displayed in the HTML page with a simple UI.

Results and Observations

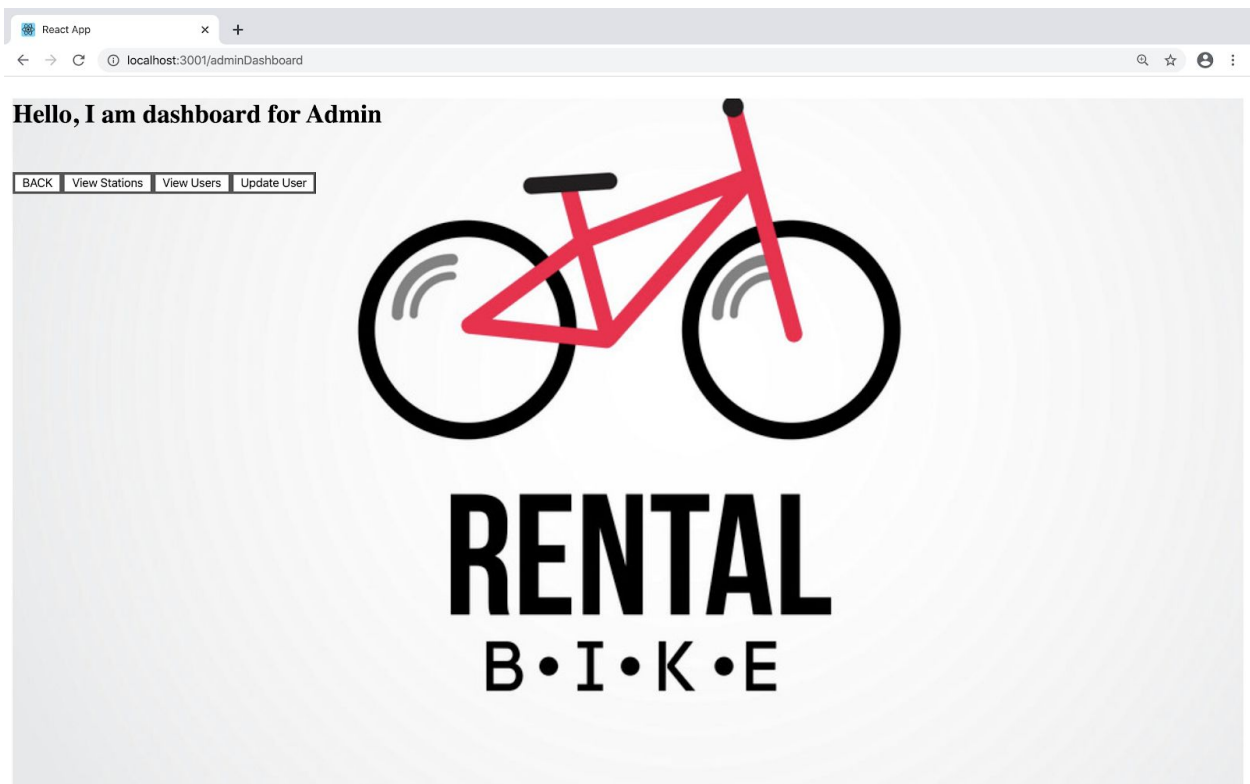
Using the concepts of React.js, Mongodb, Node.js taught in the class, we were able to create a basic application of Bike Sharing with approximately 10.3k instances of data. The application can perform all basic CRUD operations and depending on the type of user roles, CRUD operations can vary according to the user. MERN stack (Mongo, Express, React and Node) was an appropriate choice to build this kind of application since it was a single page application.

After completing the project , these are some results that we have obtained:

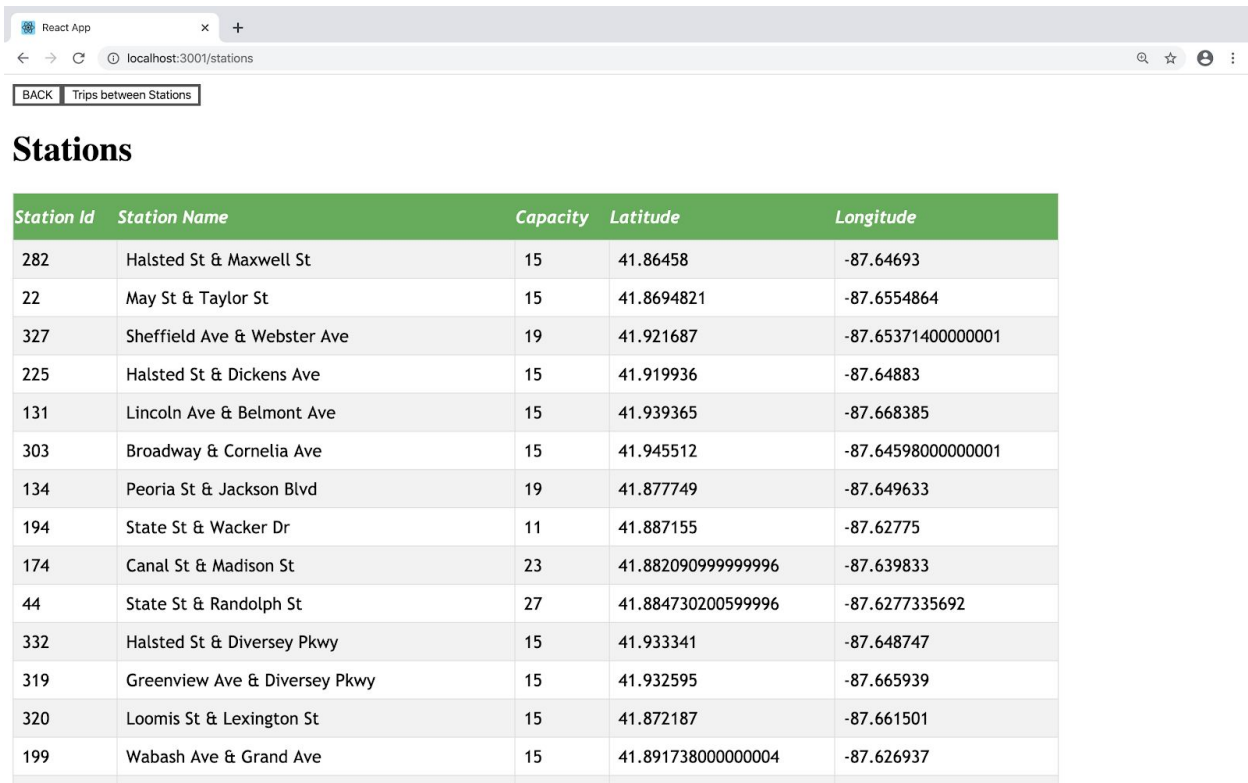
- Index page : This is how the first page would look like. As per the id specified by the user, for it be customer, subscriber or admin, the next upcoming menus will vary.



- Index page of Admin



- View Station Page : An admin can view the details of each station. Each station will show an updated capacity according to the bike usage.



Station Id	Station Name	Capacity	Latitude	Longitude
282	Halsted St & Maxwell St	15	41.86458	-87.64693
22	May St & Taylor St	15	41.8694821	-87.6554864
327	Sheffield Ave & Webster Ave	19	41.921687	-87.65371400000001
225	Halsted St & Dickens Ave	15	41.919936	-87.64883
131	Lincoln Ave & Belmont Ave	15	41.939365	-87.668385
303	Broadway & Cornelia Ave	15	41.945512	-87.64598000000001
134	Peoria St & Jackson Blvd	19	41.877749	-87.649633
194	State St & Wacker Dr	11	41.887155	-87.62775
174	Canal St & Madison St	23	41.882090999999996	-87.639833
44	State St & Randolph St	27	41.884730200599996	-87.6277335692
332	Halsted St & Diversey Pkwy	15	41.933341	-87.648747
319	Greenview Ave & Diversey Pkwy	15	41.932595	-87.665939
320	Loomis St & Lexington St	15	41.872187	-87.661501
199	Wabash Ave & Grand Ave	15	41.891738000000004	-87.626937

- Trips Between stations : An admin can also view different trips between any two stations.



Enter From Station ID

Enter To Station Id

- Results of Trips Between Stations : This is where the above submit button will take the admin. Given below is a result of various trips between Station 24 and 329.

```
localhost:4000/trips/between/stations
[
  {
    "id": 2281864,
    "start_time": "2020-04-17T19:11:26.000Z",
    "stop_time": "2020-04-17T19:11:26.000Z",
    "trip_duration": 25.3,
    "user": 100,
    "from_station": 24,
    "to_station": 329,
    "cost_of_trip": 29585,
    "v": 0
  },
  {
    "id": 2252723,
    "start_time": "2020-04-17T19:11:34.000Z",
    "stop_time": "2020-04-17T19:11:34.000Z",
    "trip_duration": 21.65,
    "user": 57984,
    "from_station": 24,
    "to_station": 329,
    "cost_of_trip": 45671,
    "v": 0
  },
  {
    "id": 2355135,
    "user": 1,
    "from_station": 24,
    "to_station": 329,
    "cost_of_trip": 3482842689,
    "v": 0
  }
]
```

- View Users Index Page : For a user this will be the index page after entering the id.

React App

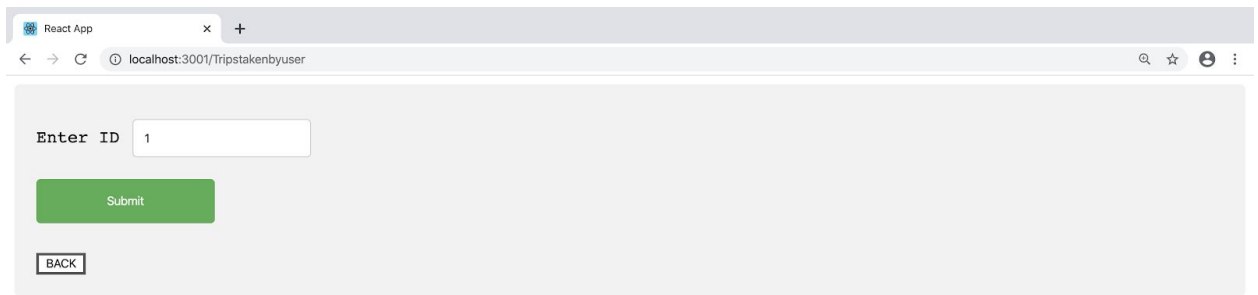
localhost:3001/users

[BACK](#) [Trips Taken By User](#) [Amount paid by User](#) [Total Amount](#)

Users

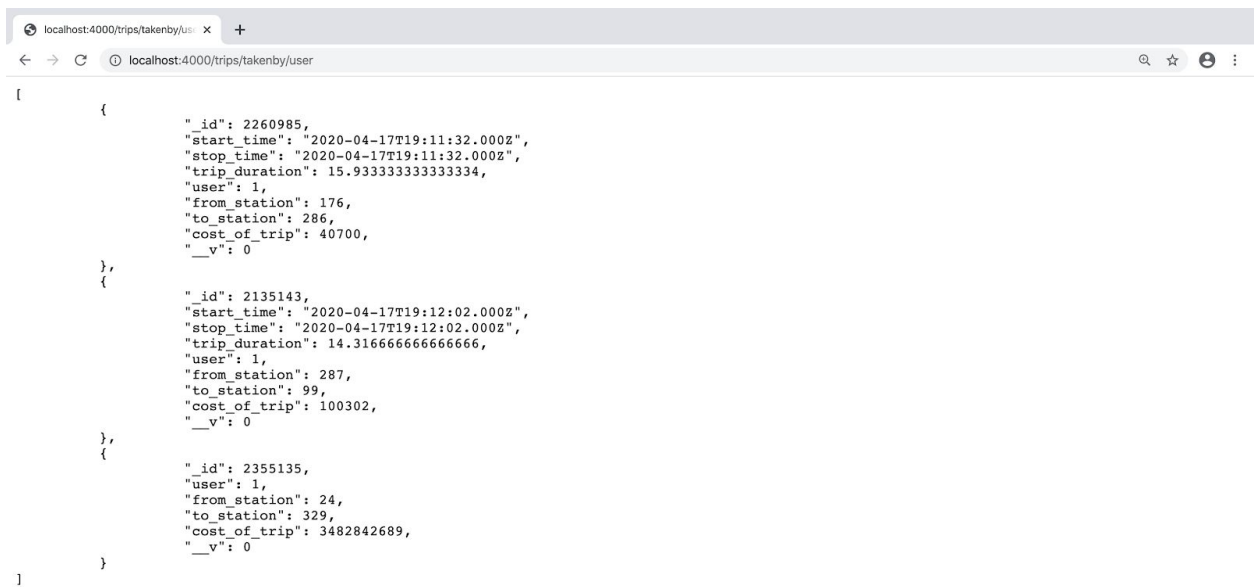
User Id	Gender	UserType
1	Male	Subscriber
4	Male	Subscriber
7	Male	Subscriber
8	Male	Subscriber
9	Male	Subscriber
10	Male	Subscriber
11	Female	Subscriber
12	Female	Subscriber
16	Male	Subscriber
18	Male	Subscriber
15	Male	Subscriber
20	Female	Subscriber
19	Male	Subscriber

- Trips Taken By User



The screenshot shows a web browser window with the address bar displaying 'localhost:3001/Tripstakenbyuser'. The page contains a form with the label 'Enter ID' followed by a text input field containing the number '1'. Below the input field is a green 'Submit' button. At the bottom left of the form area is a 'BACK' button.

- Results of Trips Taken By User



The screenshot shows a web browser window with the address bar displaying 'localhost:4000/trips/takenby/user'. The page displays a JSON array of three trip objects. Each object contains fields for _id, start_time, stop_time, trip_duration, user, from_station, to_station, cost_of_trip, and __v.

```
[
  {
    "_id": 2260985,
    "start_time": "2020-04-17T19:11:32.000Z",
    "stop_time": "2020-04-17T19:11:32.000Z",
    "trip_duration": 15.933333333333334,
    "user": 1,
    "from_station": 176,
    "to_station": 286,
    "cost_of_trip": 40700,
    "__v": 0
  },
  {
    "_id": 2135143,
    "start_time": "2020-04-17T19:12:02.000Z",
    "stop_time": "2020-04-17T19:12:02.000Z",
    "trip_duration": 14.316666666666666,
    "user": 1,
    "from_station": 287,
    "to_station": 99,
    "cost_of_trip": 100302,
    "__v": 0
  },
  {
    "_id": 2355135,
    "user": 1,
    "from_station": 24,
    "to_station": 329,
    "cost_of_trip": 3482842689,
    "__v": 0
  }
]
```

- Admin-update Subscriber Page: An admin can view and update the subscriber's information.

React App x +

localhost:3001/updatesubscriber

Please Enter your UserId and Fields you wish to update

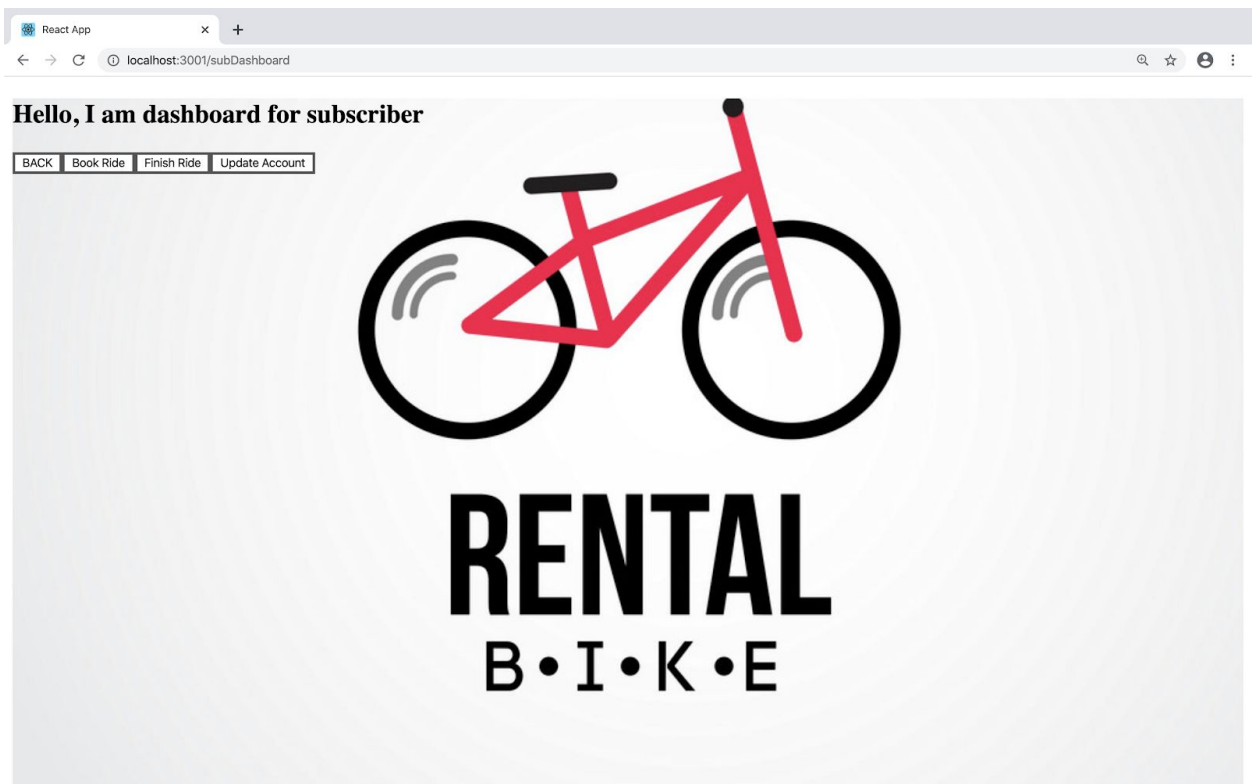
Enter UserID

Choose a userType:

Choose a Subscription Plan:

Choose Occupation:

- Index Page For Subscriber



- Book a Ride

A screenshot of a web browser showing a form titled 'Book a Ride'. The browser's address bar displays 'localhost:3001/book/ride'. The form contains three input fields: 'Enter User ID' with the value '1', 'Enter from Station Id' with the value '24', and 'Enter to Station Id' with the value '329'. Below these fields is a green 'Submit' button and a 'BACK' button.

- Finish Ride

A screenshot of a web browser showing a form titled 'Finish Ride'. The browser's address bar displays 'localhost:3001/finish/ride'. The form displays a confirmation message: 'Hi, Your Ride is Confirmed and Bike is reserved at the Station' and 'Please Click the Finish Button, once you reach destination'. Below this, there is an 'Enter ID' field with the value '1', a 'Choose a Payment Option:' dropdown menu with 'Credit' selected, a green 'Finish' button, and a 'BACK' button.

- Result of Finish Ride page

A screenshot of a web browser showing the result of the finish ride. The browser's address bar displays 'localhost:4000/finish/ride'. The page content is 'Thanks For Riding'.

After the ride is booked, it can be viewed by the admin.

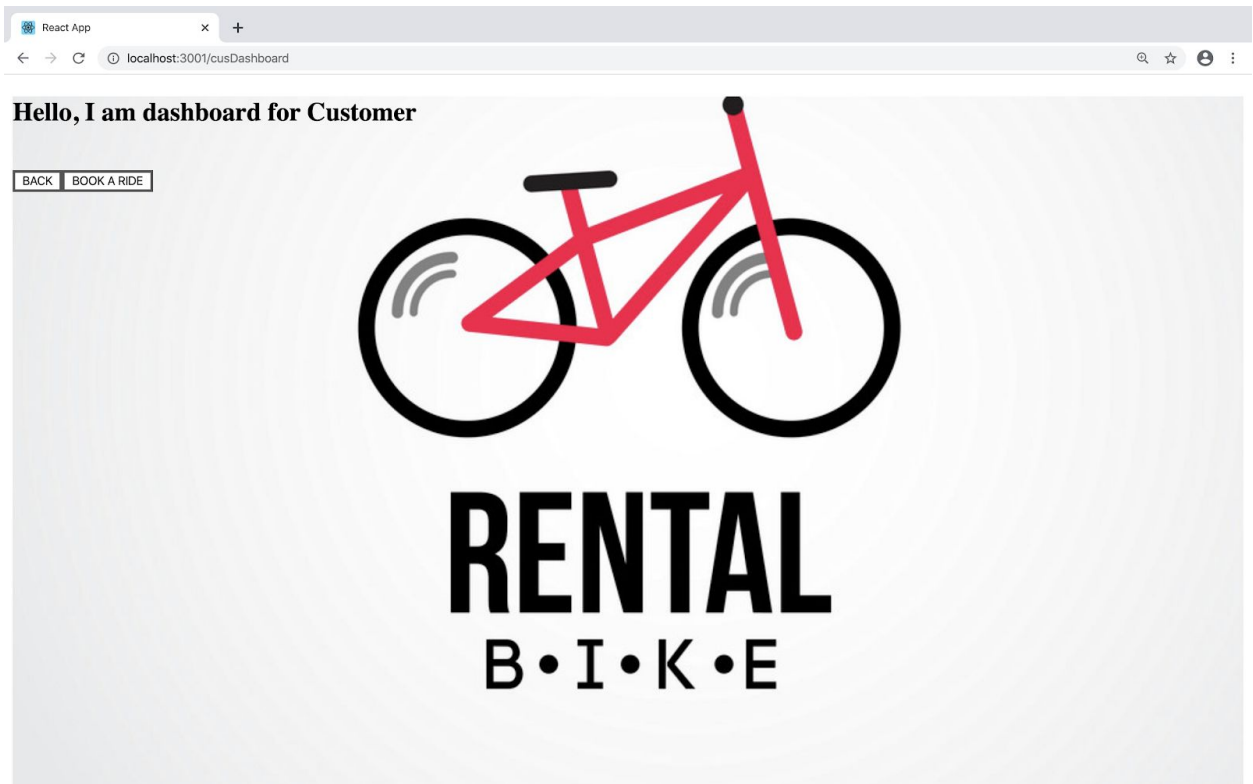
- Result after booking a ride user =1 (Results/User-1_book_new_ride)

```
localhost:4000/trips/takenby/user
[
  {
    "id": 2260985,
    "start_time": "2020-04-17T19:11:32.000Z",
    "stop_time": "2020-04-17T19:11:32.000Z",
    "trip_duration": 15.933333333333334,
    "user": 1,
    "from_station": 176,
    "to_station": 286,
    "cost_of_trip": 40700,
    "__v": 0
  },
  {
    "id": 2135143,
    "start_time": "2020-04-17T19:12:02.000Z",
    "stop_time": "2020-04-17T19:12:02.000Z",
    "trip_duration": 14.316666666666666,
    "user": 1,
    "from_station": 287,
    "to_station": 99,
    "cost_of_trip": 100302,
    "__v": 0
  },
  {
    "id": 2355136,
    "user": 1,
    "from_station": 24,
    "to_station": 329,
    "cost_of_trip": 3770449349,
    "__v": 0
  }
]
```

- Result of Amount Paid by User

```
localhost:4000/ammount/paidby/user
{
  "sum": 47.25,
  "result": [
    {
      "id": 1,
      "gender": "Male",
      "phoneNumber": 2725829902,
      "userType": "Subscriber",
      "subscriber": {
        "id": "5e9f4ef60ff6af9e21735309",
        "subscriptionPlan": "MONTHLY",
        "OccupationType": "ENGINEER"
      },
      "__v": 0,
      "trip_info": {
        "id": 2260985,
        "start_time": "2020-04-17T19:11:32.000Z",
        "stop_time": "2020-04-17T19:11:32.000Z",
        "trip_duration": 15.933333333333334,
        "user": 1,
        "from_station": 176,
        "to_station": 286,
        "cost_of_trip": 40700,
        "__v": 0
      },
      "price": [
        {
          "id": 40700,
          "trip": 2260985,
          "payment": 1,
          "ammount": 15.933333333333334,
          "__v": 0
        }
      ]
    },
    {
      "id": 1,
      "gender": "Male",
      "phoneNumber": 2725829902,
      "userType": "Subscriber",
      "subscriber": {
        "id": "5e9f4ef60ff6af9e21735309",
        "subscriptionPlan": "MONTHLY",
        "OccupationType": "ENGINEER"
      },
      "__v": 0
    }
  ]
}
```

- Index page for Customer



- Index page for registering as a new user

A screenshot of a web browser displaying the New User Registration page. The browser's address bar shows 'localhost:3001/registernewuser'. The page has a light gray background. It contains several form fields and a submit button. The fields are: 'Enter User Name' (text input), 'Enter Gender' (text input), 'Choose a userType:' (dropdown menu with 'Subscriber' selected), 'Choose a Subscription Plan:' (dropdown menu with 'YEARLY' selected), and 'Choose Occupation:' (dropdown menu with 'STUDENT' selected). A green 'Submit' button is located to the right of the 'Choose Occupation' dropdown. Below the form fields, there is a message: 'If you choose to be a Subscriber , Please Enter the Below Details'. At the bottom of the page, there is a 'BACK' button and a message: 'Please note down your UserId after clicking the Submit Button' and 'Thank you for Registering. Hope You have a safe ride.'

Future Work

Since the application built can perform all functionalities of a bike sharing application, a need for an aesthetical UI emerges. Since the designing and building of UI was out of the scope of this course, we relied on creating simple HTML pages to display our functionality of the application to the user. The future work would hence definitely include creating an intuitive UI/UX for the end user.

We also ran into performance issues with handling our requests from the server, since the size of the data being sent and retrieved is huge. However, again since optimization of time to retrieve and handle queries was not the focus of this course, future work would involve working on query and time optimization by adding techniques and algorithms used in the industry widely.

References

- <https://www.kaggle.com/yingwurenjian/chicago-divvy-bicycle-sharing-data#data.csv>
- <https://www.mongodb.com/cloud/atlas>