Design Document

Project Title: MBTA Transit App for Movin Maps

Section	Author
Preface	Derek
Introduction	Dan
Glossary	Derek, Dan
User Requirements	Dan, Archie, Derek
System Requirements	Dan, Archie, Derek
Architecture	Dan, Archie, Derek
Database Design	Derek, Dan
Restful Services	Derek

1 Preface

Version	Date	Description
1	2/28/2023	Starting template.
2	3/10/2023	Initial version.

2 Introduction

The MBTA offers various transportation methods to the Massachusetts area through services such as the subway system, commuter rail, bus, and ferry routes. These transportation methods can be difficult to navigate. The MBTA Live Map offers users real-time locations of transport through a GUI to make it easier to follow and plan routes. Users will be able to store their favorite routes on their account, filter the GUI to show specific routes or methods of transportation, and be recommended routes relative to their location. The status will display arrivals and alerts will update with delays.

3 Glossary

Term	**Definition**
Card	provides a flexible and extensible content container with multiple variants and options.
GPS	Global Positioning System. U.S.-owned utility that provides users with positioning, navigation, and timing (PNT) services.
Landing Page	standalone page designed to display information to the user
Drop Down Menu	toggleable, contextual overlays for displaying lists of links
Route(s)	a way or course taken in getting from a starting point to a destination.
drop-up	toggleable, contextual overlays for displaying lists of links
Pop-up Menu	Graphical User Interface of menu that opens on top of a background
Nav Bar	Box with input buttons to access other features and navigate to other pages
Transit	Process of moving to another location through a system.
MongoDB	Open Source NoSQL Database
Express	Backend web application for building restful API’s
React	Open source front end library using to create interactive interfaces
Node.js	Open source server application

4 User Requirements

Movin Maps - User Requirements

5 System Requirements

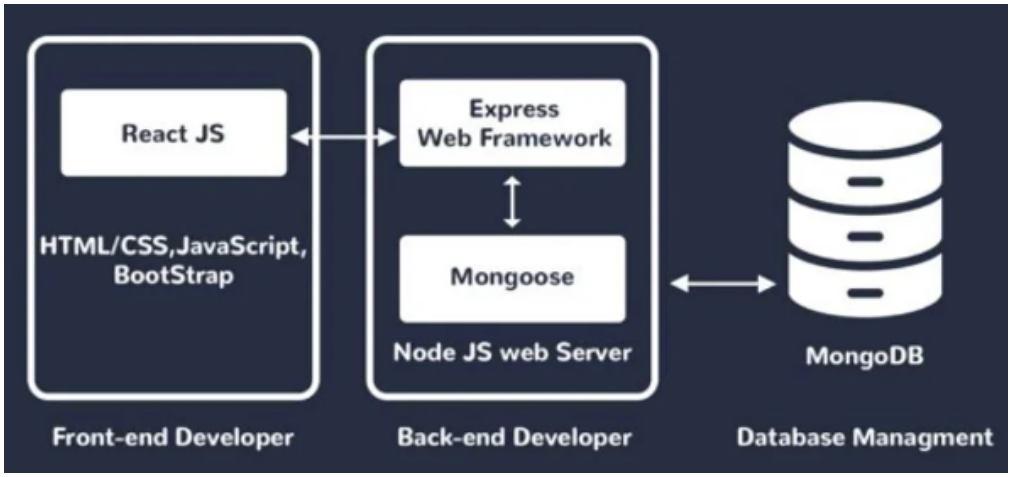
Movin Maps - System Requirements

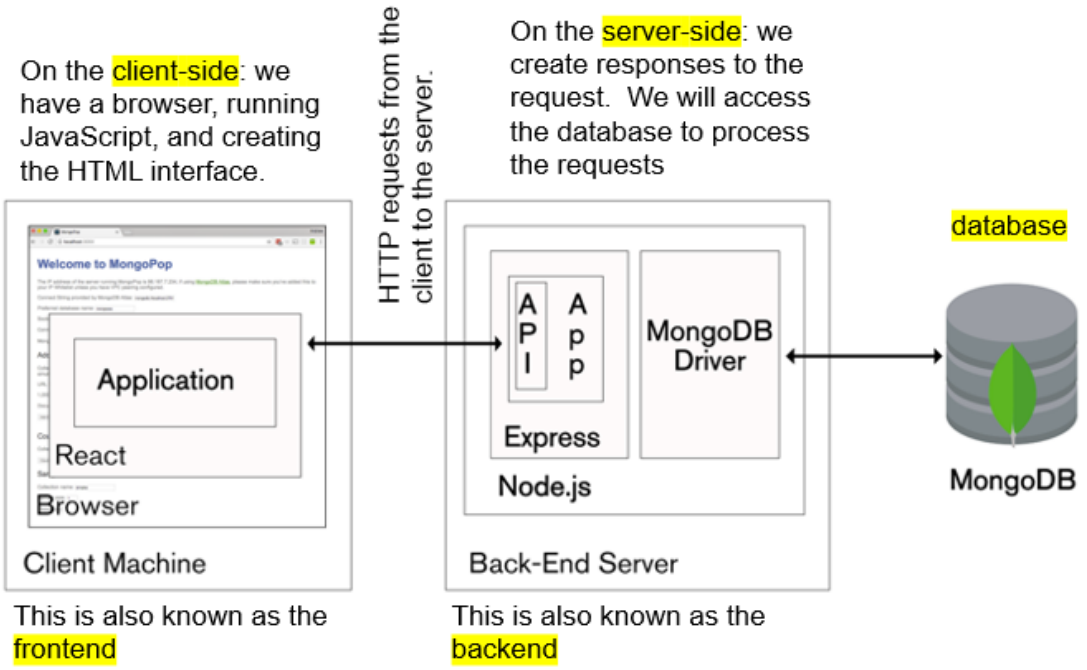
6 Architecture

6.1 Architectural View

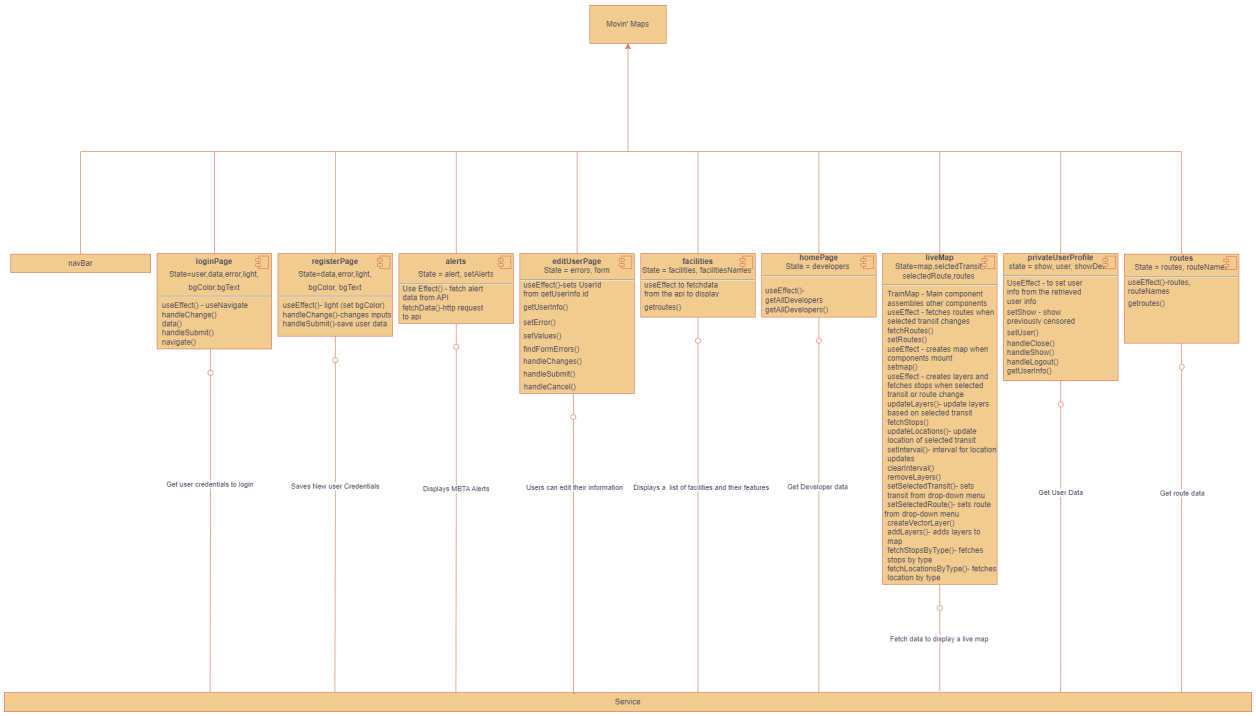
The framework for the application is the MERN stack. We are using a Model-View-Controller-Router (MVCR).

- Model - Represents the data and business logic of the application. This is done using MongoDB.
- View - This is the UI built using React.
- Controller - acts as an intermediary between the model and view, handling user input and updating the model accordingly. This is implemented using the Express web framework for Node.js.
- Router - handles incoming requests from the client-side, determining which controller to use to handle the request. This is also implemented using the Express web framework.





6.2 Component View



6.2.1 loginPage

This component displays a list of contributors to the application. The component also has a button so that any contributor can be deleted.

6.2.2 mbtaLiveMap

Displays live map tracking mbta transit. Suser cans elect transit and route.

7 Database Design

The application uses a non-SQL database, MongoDB. The database consists of “collections”. A collection is a grouping of MongoDB documents. Documents within a collection can have different fields. A collection is the equivalent of a table in a relational database system.

7.1 User Collection

```
const mongoose = require("mongoose");

//user schema/model
const newUserSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
      label: "username",
    },
    email: {
      type: String,
      required: true,
      label: "email",
    },
    password: {
      required: true,
      type: String,
      min : 8
    },
    date: {
```

```
      type: Date,
      default: Date.now,
    },
    favRoute: {
      required: false,
      type: String,
      label: "favorite routes"
    }
  },
  { collection: "users" }
);

module.exports = mongoose.model('users', newUserSchema)
```

7.2 Developer Collections

```
const mongoose = require("mongoose");

// Schema for the developer
const developerSchema = new mongoose.Schema(
  {
    fName: {
      type: String,
      required: true,
      label: "first name",
    },
    lName: {
      type: String,
      required: true,
      label: "Last Name",
    },
    projDescription: {
        type: String,
        required: true,
        label: "Description of sections worked on"
    }

  },
  { collection: "developerSchema" }
);
// Export the model
module.exports = mongoose.model('developerSchema', developerSchema)
```

Includes fields for developer first name, last name, and description of what they worked on for the project.

8 RESTful Services

The backend provides RESTful services to retrieve information from the MongoDB database. This section describes these services.

RESTful services are those that define architectural principles to help design web services , REST standing for Representional State Transfer. It helps when interacting with the backend and the database. With HTTP the methods that are used with the database are POST,

that makes a resource on the server, GET, retrieves a resource from the server, PUT which update the resource or changes the state of it and DELETE to delete a resource.

8.1 Users

The User Service is responsible for authenticating the users of the service. The service provides the ability to: create users, delete users, edit users, and authenticate users. The stored passwords are encrypted. Users have usernames.

8.1.1 User Create an Account

For a new user, three fields must be completed to develop a new user profile.

POST http://localhost:8081/user/createUser

```
{
    "username" : "jdoe",
    "email" : "jdoe@gmail.com",
    "password" : "jdoe123"
}
```

Response:

Returns code 400 if there is a bad request.

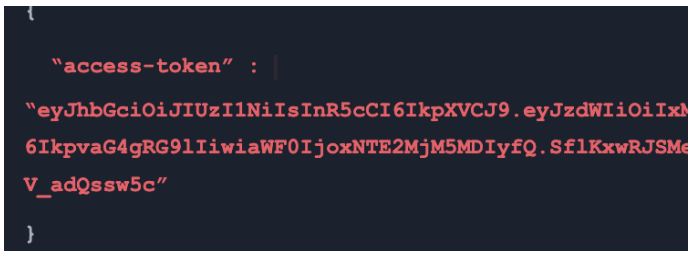
8.1.2 User Log into Account

For a returning user, two fields must be filled in. The service will attempt to authenticate the user and return success or failure.

POST http://localhost:8081/user/login

Response:

Returns code 404 if the userID entered is not found.



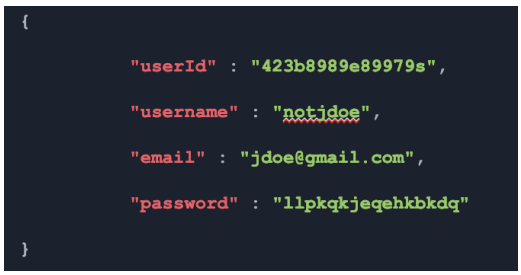
8.1.3 EditUsername based on UserID

For a signed in session user, the user has the option to modify the username

PUT http://localhost:8081/user/editUser/{userId}

Response:

Returns code 404 if the userID is not found.



8.1.4 Retrieve User List

Users are able to access a list of all the current users in the social media app.

GET http://localhost:8081/user/getAll

Response:

Returns code 404 if the userID is not found.

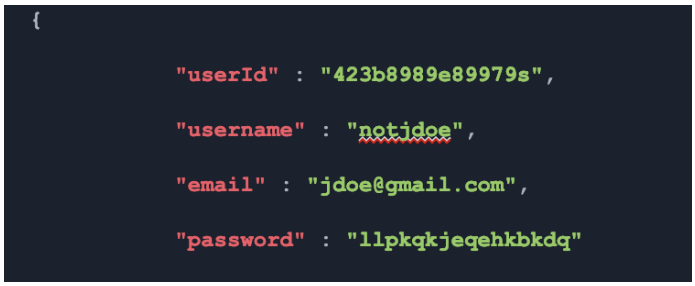
8.1.5 Retrieve User by UserID

GET http://localhost:8081/user/getUser/{userId}

User will search for a fellow user based on the username, then receive back the user's profile

Response:

Returns code 400 if there is a bad request



8.1.6 Changes

Create a function to input favRoute and store it in mongodb.
Adding a checkbox to select favRoute.
Display favRoute on live map.