TO DOCUMENT

SERVER

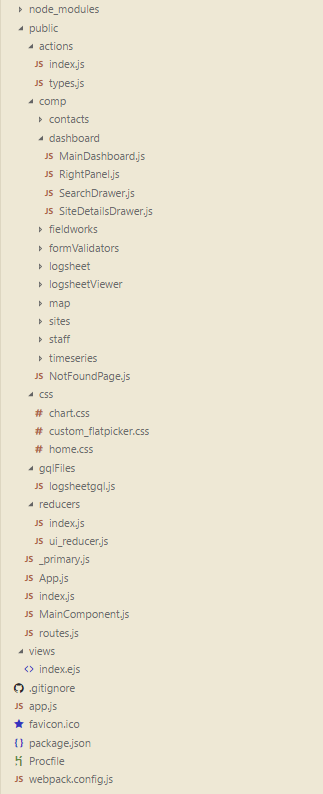
* Project structure
* Packages
* Express Server
* Node js filesystem module
* Graphql Schema
* Graphql Resolvers
  + Queries
  + Mutations
  + Subscriptions
* Connector Database (Mysql – sequelize)
* Python part
* Instructions to build and run
* Github repo

CLIENT

* Project structure
* Packages
* Express server setup
* Views
  + Index ejs
* React Components
  + All folders
* Design
  + Css
  + Material-ui
* Redux
  + Store
  + Action Creator
  + Reducers
    - Ui
    - Routing
* Apollo Client
  + Queries
  + Mutations
  + Subscriptions
* Instructions to build and run
* Github repo

GPSTEAM WEB APP PROJECT CODE DOCUMENTATION

CLIENT

1. PROJECT STRUCTURE

This is the typical node.js project structure. Each folder serves an important role in the project. Below is the summarized explanation of each part from top folder to bottom**.**

*node\_modules* – contains all project dependencies. The contents of this folder is automatically created when ‘npm install’ command is run on the project’s main directory.

*public –* where most of the project’s source code is saved. Everything in this folder is not yet compiled. Webpack will dive into this directory and package them into one bundle.

*actions –* a subfolder that contains the redux actions necessary to manage the state of the application. The files on this folder holds functions or methods for changing the applications ui and data states.

*comp* – short for components, a subfolder that contains subfolders for react components which are organized by classes. Most of the visible or viewable parts of the application is managed here.

*css* – cascading stylesheets are located here. This handles the styling of some components.

*gqlFiles* – gql is short for graphql, this subfolder holds the query objects that are exported and used by react components to request data from the server.

*reducers –* another subfolder used by Redux that handles the changes to the redux store. Functions on this folders are being called when certain actions are called. Reducers specifies the application’s state changes in response to every actions.

*views* – this folder contains the ejs file or embedded javascript template. Ejs file is a simple templating language that will generate HTML markup with plain Javascript. Webpack will also look for this file and execute it to generate an HTML file which will serve as the main view of the application.

2. PACKAGES

{

"name": "gps\_project",

"version": "1.0.0",

"description": "GPSteam dashboard web application",

"main": "index.js",

"scripts": {

"dev": "webpack-dev-server",

"prod": "npm run clean && webpack -p",

"clean": "rimraf ./dist/\*",

"start": "nodemon app.js",

"buildrun": "npm run prod && npm run start"

},

"author": "Oriel Absin",

"license": "ISC",

"dependencies": {

"apollo-client": "^1.9.0-0",

"apollo-upload-client": "^5.1.0",

"autoprefixer": "^7.1.2",

. . .

},

"devDependencies": {

"babel-cli": "^6.24.1",

"babel-core": "^6.25.0",

"babel-eslint": "^7.2.3",

. . .

}

}

Package.json files – manages locally installed npm packages. It documents all packages that the project depends on. It also contains the project description and the scripts that will use to build and run the application. Read more from [npm website](https://docs.npmjs.com/getting-started/using-a-package.json).

These are the functions of the scripts:

|  |  |
| --- | --- |
| npm run dev | Run the application in development mode |
| npm run prod | Run the application in production mode |
| npm run clean | Clean the contents of the distribution folder |
| npm run start | Alternate way of running the application in prod mode (not using webpack) |
| npm run buildrun | Build and run the application |

3. Express Server setup

// app.js

// import express and path modules

// express is a web framework for node.js

// while path provides utilities for working with file and directories.

var path = require("path"),

express = require("express");

// get the build directory where all compiled source code resides

// provide a port number and put express function in a variable app

var DIST\_DIR = path.join(\_\_dirname, "dist"),

PORT = 8000,

app = express();

//Serving the files from the dist folder

app.use(express.static(DIST\_DIR));

//Send index.html when the user access the web

app.get("\*", function (req, res) {

res.sendFile(path.join(DIST\_DIR, "index.html"));

});

console.log('production app running on PORT: ', PORT)

// make the application running on the desired Port number

app.listen(PORT);

4. Views

// index.ejs

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">

<link href='https://api.mapbox.com/mapbox-gl-js/v0.36.0/mapbox-gl.css' rel='stylesheet' />

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css" />

<title><%= htmlWebpackPlugin.options.title %></title>

</head>

<body>

<div id='app' />

</body>

</html>

This is the ejs file. EJS is a templating engine that produces HTML markup. This file is just basically an HTML with additional syntax to help interface with webpack bundle builder. The essential part of the code within this file is the **<div id=’app’>** where the actual application will render. It is the element that the react modules will look up for when initializing its views.

5. React Components

React is a library that helps developers create interactive user interfaces. It has mechanism in handling application states while the data changes. It encourages a declarative way of coding to make the codes more predictable and easier to debug. Learn more from <https://facebook.github.io/react/>.

5.1 The Logsheet Component