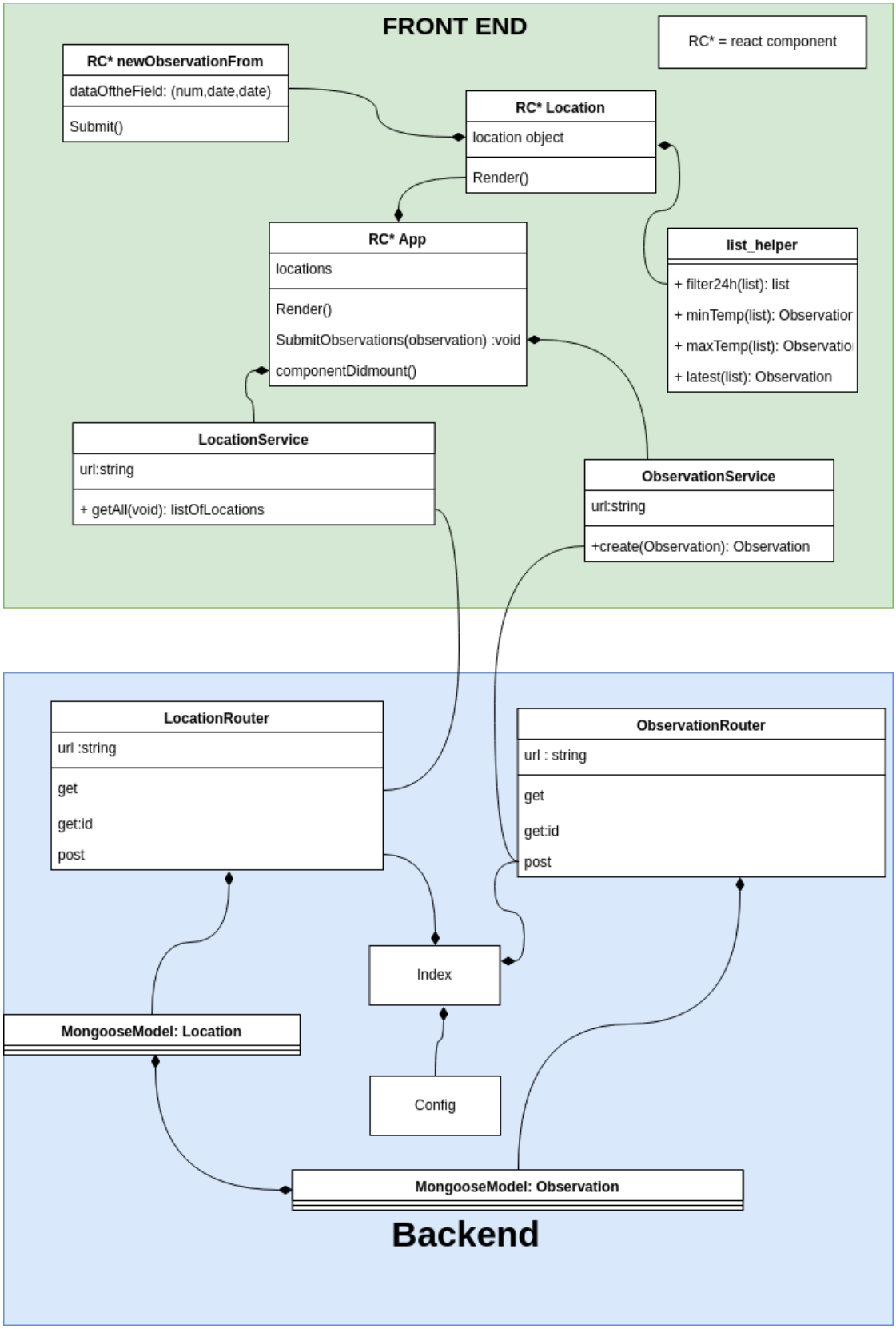
Weatherapp-Documentation



Introduction

This is a documentation for the weather web application. Application is based on Fullstack-design paradigm and consists of a backend and a frontend. The backend utilizes node.js and express, to manage an api that is mostly REST compatible. The backend provides two services, one for weather observations, and the other for specific locations. For storage the backend uses MongoDB that is accessed with mongoose. As of now the mongo is deployed in mlab and that causes increased latency in the application.

The front end is a react-application, that allows all users to view and add observations to the three locations specified in the assigment. It shows the most recent temperature and the highest and lowest temperatures of the last 24 hours for all of the locations.

Backend

index.js

The entry point to the backend is index.js. Index.js defines routers for '/api/locations/' and '/api/observations/' that are respectively managed by '/controllers/locations.js'/ and '/controllers/observations.js'/. It also connects the application to mongodb specified in environment variables which is accessed by '/utils/config.js'. Server binds to the port defined in the environment variables.

/controllers/locations.js

This is defines a simple express router that provides interface for getting all locations and posting new locations. The post feature is not suppported by the frontend and is solely for adding initial location points. The get router fetches all of the locations from mongodb and populates their observations and then sends all of the data to the client.

Usage:

"GET {baseurl}/api/locations"

→ Response = {

city: "cityname",

coordinates: "string of coordinates",

_id: "string id from mongodb",

observations: [obs1,obs2….,obsN]

}

'/controllers/observations.js'

This defines a simple express router that provides interfaces for getting all weather observations and for posting new observations. Only the post feature is used by front end as fetching all of the

locations will also return all of the observations. This controller also assigns new observations to the respective locations. That is why the location and its _id must be within the observation object when it is sent to the backend.

Usage:

"POST {baseurl}/api/locations", Request body = {

temperature: number,
location: {city:"cn",
_id :"mongoId"},

time: dateObject

}

→ Response body = {

_id: "mongoid",
temperature: number,
location: {city:"cn",
_id :"mongoId"},

time: dateObject

}

Front end

As said above the front end is a react-application. It has a list of toggleable location-components. The location components show the information specified in the assingment: max, min and most recent temperature. It also displays a form for adding new observations. The form has a number field for temperatures, a date picker and a time picker. The excecution is bit clunky and it would have to be polished in the future, but because of tight time constraints it will have to do for now.

The front end applies just enough css to make it usable and could use more polishing.

The front end is divided to services, components and utils. Services describes two axios based collections that communicate with the backend. Components are the react-components that are rendered to the front end. In utils there is a list-helper that describes the list operations needed to display the specified data.

The state of this app is managed by the App object and only it uses access to the backend.