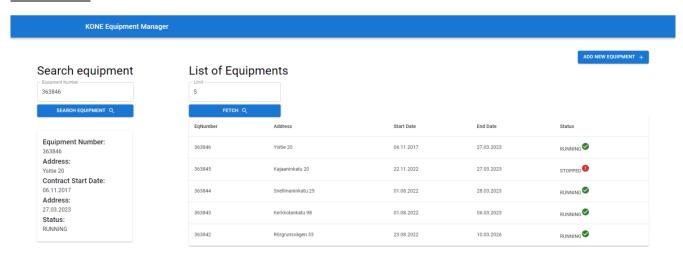
# **FRONTEND**

## Home Screen



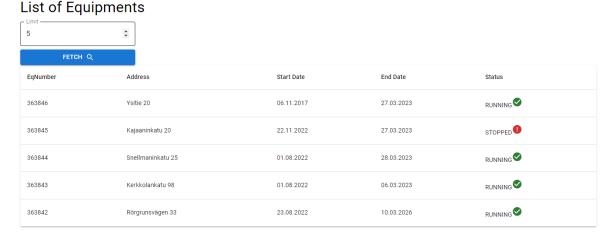
**React MUI(Material UI)** has been used to for each component. **TypeScript** used.

Components inside HomeScreen:

- 1. Appbar(on Top)
- 2. The Search Equipment component on the left, where you can search any equipment by it's number.
- 3. The List view on the right where by default it shows 5 equipments(latest added first). There is option to limit the number of equipments shown in the tabular view.
- 4. The "Add Equipment" button which opens up the Drawer from right side and allows to add new equipment to DB.

## List Equipments(Task 1)

Equipment details shown in MUI Tabular view. By default shows 5 equipment details(last added on top). There is an option to increase the limit(max 100 right now).



# Search Equipment(Task 2)

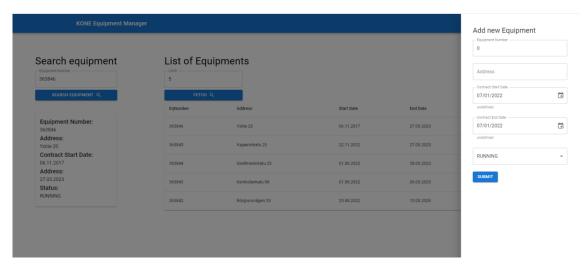
Takes an equipment number as input, shows the returned details in MUI card.

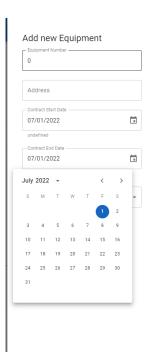
#### Search equipment



# Add Equipment Drawer(Task 3)

The "Add new Equipment" drawer contains a form created using **Formik**. Basic Form validations on frontend are handled using **Yup**. I did not add complex validations on UI just to keep it simple and quick.

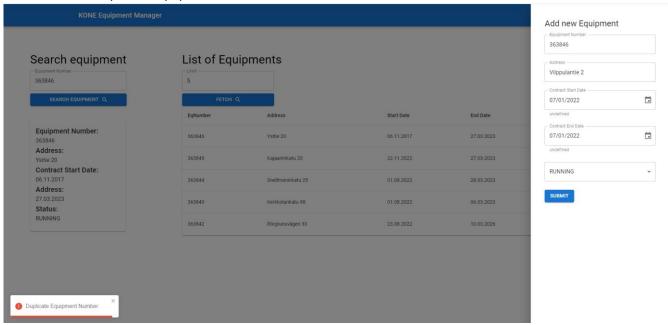




MUI Datepickers are used for dates.

# **Basic Validations and Error Messages**

**Axios interceptors** have been used to intercept request and response and avoid code duplication while sending requests to REST apis. **React toastify** has been used to show any errors that comes from API, no separate handling needs to be done inside any components. This screenshot shows the error on bottom left corner of screen when a Duplicate Equipment number is added.



# **Test Cases**

Test cases have been written for some components using RTL and jest.



# **BACKEND**

**Serverless framework** has been used just to speed up development and deployment. Amplify could also have been used but since I am using MongoDB Atlas as the No-Sql DB, it becomes little tricky to connect to it. Amplify has good preference for DynamoDB.

With serverless, you just have to write a YML(Infra as a code) and behind the scene it uses CloudFormation to do the work and deploy it on AWS, just make the process easy.

## **Handler Functions**

Each handler function has been put in their respective file. TypeScript has not been used. MongoDB connection code has been placed in separate file so that the connection is reused instead of creating new connection for each request. There are 3 handler functions created respective to each task.

## SOME POINTS THAT I WOULD LIKE TO HIGHLIGHT ABOUT BACKEND

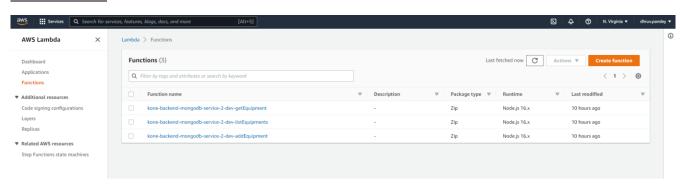
- 1. I have not used any middlewares for validations or parsing requests or preparing responses. Most of the work has been done manually just to keep it simple and quick.
- 2. I had added IAM auth to secure the endpoints(it's quite simple with YML file). It was working nicely with Postman but somehow it was not working on the React frontend because there is a tricky way to send the credentials. I didn't want to consume too much time, so removed it.

```
JS listEquipments.js X
                                                     \begin{array}{ll} \mbox{handlers} \ \searrow \ \mbox{\sc MistEquipments} \ \searrow \ \mbox{\sc MistEquipments} \ \searrow \ \mbox{\sc Meaders} \\ \mbox{\sc 1} & \mbox{\sc const} \ \mbox{\sc dbConnector} \ = \ \mbox{\sc require("../mongo-client");} \end{array}
∨ KONE-BACKEND-... [4 📮 🖔 🗗
                                                                  module.exports.listEquipments = async (event) => {
  const db = await dbConnector.getDBInstance();
  const limit = event.queryStringParameters?.limit || 5;
  ∨ handlers
  JS addEquipment.js
  JS getEquipment.js
                                                                     if (isNaN(limit) || Number(limit) < 1) {</pre>
  JS listEquipments.js
  > node_modules
                                                                            headers: {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Credentials": true,
.gitignore
JS mongo-client.js
                                                                             body: JSON.stringify({
{} package-lock.json
{} package.json
                                                                         .sort({ equipmentNumber: -1 })
                                                                          .limit(Number(limit))
```

## Serverless Dashboard



# **AWS Dashboard**



# Mongo DB Atlas(Cloud) Dashboard

