# Corona Management System

This is a management system for keeping track of members, manufacturers, medical conditions, vaccination records, and vaccinations related to the coronavirus. The system is built using Node.js and PostgreSQL as the database. It provides a RESTful API for performing some of the  CRUD operations on the data.

## Getting started

## Prerequisites

- Node.js
- PostgreSQL
- Express.js
- Nodemon- optional
- Postman - optional

To get started with the project, follow these steps:

1. Clone the repository to your local machine
2. Install the required dependencies  by running 'npm install'
   cd corona-management-system
   npm install
3. Set up the database
   You will need to have PostgreSQL installed and running on your machine. Create a new database and update the database and password values in the index.js file to match your own configuration.

   Create a .env file with the following information:

   DB_USER=postgres
   DB_HOST=localhost
   DB_DATABASE="your database name"
   DB_PASSWORD= "your local password"
   DB_PORT=5432

4. Start the server by running npm app.js.
   If you have installed nodemon run nodemon app.js

Step by step:

Install Node.js and NPM: First, make sure you have Node.js and NPM (Node Package Manager) installed on your system. You can download and install them from the official website https://nodejs.org/en/download/.

Create a new project directory: Create a new directory for your project and navigate to it using the command line.

Initialize a new Node.js project: In the command line, run the following command to initialize a new Node.js project and create a package.json file: npm init -y

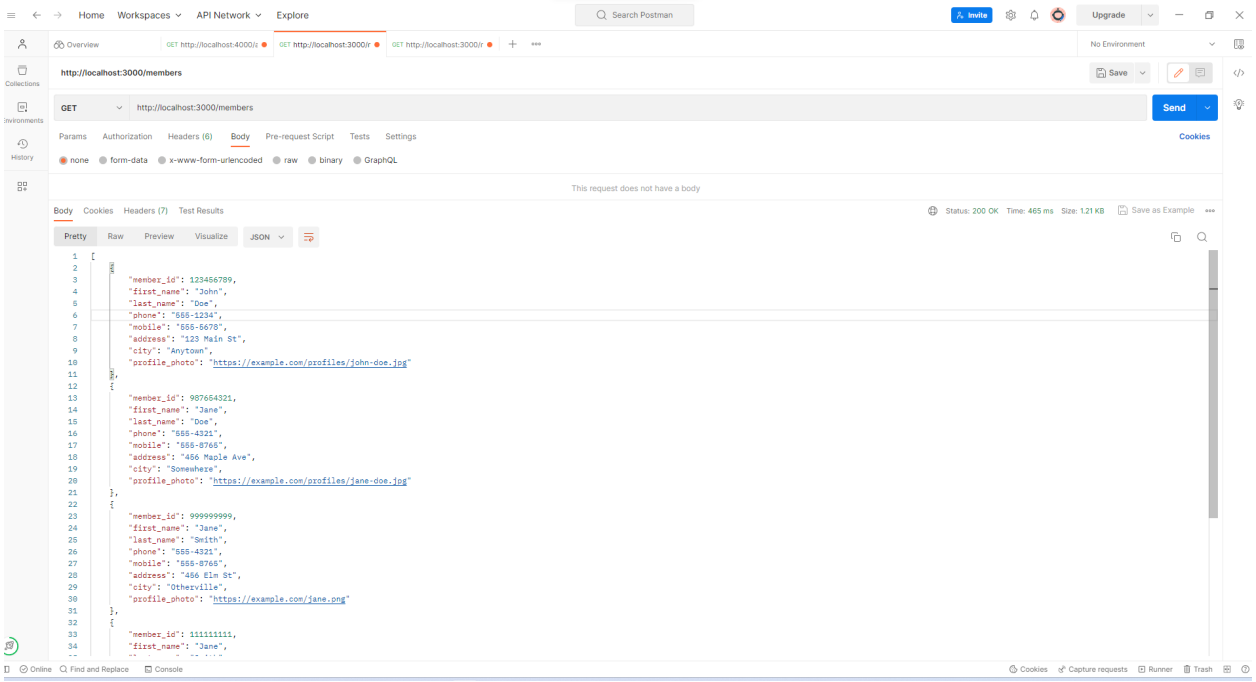
Install Express.js: Run the following command in the command line to install Express.js and save it as a dependency in your package.json file: npm install express --save

Clone the repository to your local machine

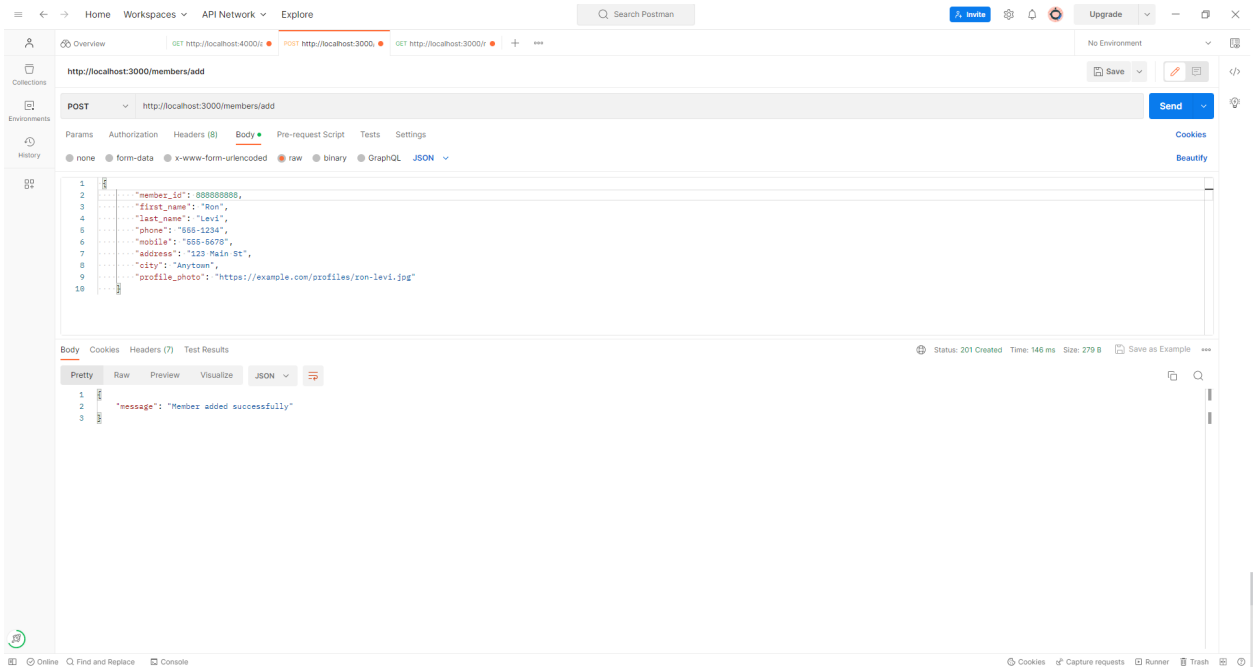
Start the server: In the command line, run the following command to start the server: node app.js

Test the server: Open a web browser and navigate to http://localhost:3000/members
You will get the full list of all members in your DB-in case it exists, otherwise, you can find the create table script in the appendix.

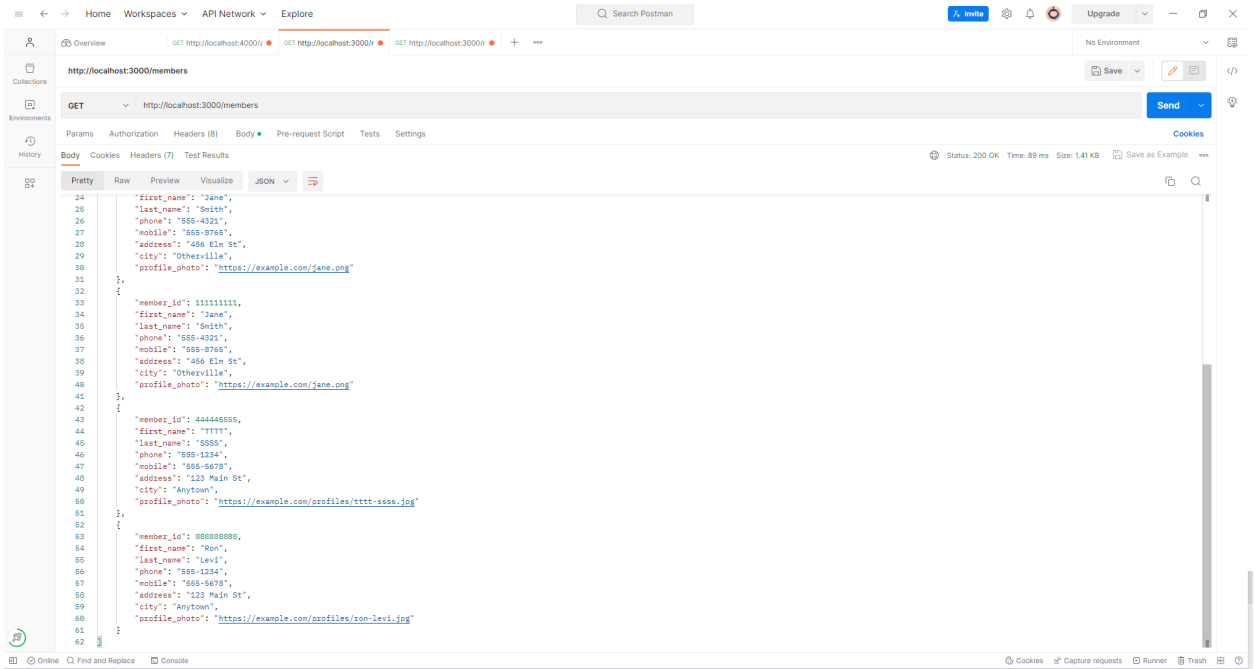
Postman is really recommended.
Example:



For adding a member you have to choose Post and put some body:




And now you can see Ron Levi on the Members List:




# API endpoints

The following endpoints are available:

## Members

GET /members - Get information about all members in the system
GET /members/:member_id - Get information about a specific member
POST /members/add - Add a new member to the system

## Manufacturers

GET /manufacturers - Get information about all manufacturers in the system
POST /manufacturers/add - Add a new manufacturer to the system

## Medical conditions

GET /medicalconditions - Get information about all medical conditions in the system
POST /medicalconditions/add - Add a new medical condition to the system

## Vaccinations

GET /vaccinations - Get information about all vaccinations in the system
POST /vaccinations/add - Add a new vaccination to the system

## VaccinationsRecords

GET /vaccinationrecords - Get information about all VaccinationsRecords in the system
POST /vaccinationrecords/add - Add a new VaccinationsRecord to the system

## Assumptions about the project:

- It was written: “Each **employee** in the system must keep the following details:
  1. Personal details:
  o First name and last name
  o Identity card
  o Address (city of residence, street and number)
  o Date of birth
  o Telephone
  o Mobile phone”
  I assumed that every **employee** meant every **member** of the health fund since we keep the medical information on Corona regarding them.
- I assumed it is possible to get sick with corona once at most as suggested in the exercise instructions.
- I would add various options for access permissions, i.e. editing or viewing, passwords, etc.

## Notes:

I built the project by dividing it into modules. However, the project is mainly in the app.js folder due to a need for more time to separate it into files and run away.

## Extensions:

If I had time left:
1. I would connect whether the members of the health insurance fund were entered into a database of the Population Authority, for example, in order to verify that the addition of a citizen to the health insurance fund system is indeed correct.
2. I would interface with Google Maps API to enter a valid street number and city. As well as the API of mobile companies if possible.
3. I would add, on the one hand, more error messages to the user detailing the nature of the error and how it can be fixed, and on the other hand, information messages that the specific action they wanted to do was captured in the system and what exactly was captured.

## Appendix:

### CREATE TABLE SCRIPT

```
CREATE TABLE Members (
  member_id INTEGER PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  phone VARCHAR(20),
  mobile VARCHAR(20) NOT NULL,
  address VARCHAR(255),
  city VARCHAR(100),
  profile_photo VARCHAR(255),
  CONSTRAINT member_id_length CHECK (member_id >= 100000000 AND member_id
<= 999999999)
);

CREATE TABLE Manufacturers (
  manufacturer_id SERIAL PRIMARY KEY,
  manufacturer_name VARCHAR(255) NOT NULL
);

CREATE TABLE Vaccinations (
  vaccination_id INTEGER PRIMARY KEY,
  manufacturer_id INTEGER REFERENCES Manufacturers(manufacturer_id) NOT NULL
);
```

```
CREATE TABLE vaccinationsRecords (
  vaccination_id INTEGER REFERENCES Vaccinations(vaccination_id),
  member_id INTEGER REFERENCES Members(member_id),
  vaccination_date DATE NOT NULL,
  PRIMARY KEY (vaccination_id, member_id)
);

CREATE TABLE MedicalConditions (
  member_id INTEGER REFERENCES Members(member_id) PRIMARY KEY,
  positive_result_date DATE,
  recovery_date DATE,
  CONSTRAINT recovery_after_positive CHECK (recovery_date > positive_result_date)
);
```