**Architectural specification of the system**

**Corona management for the HMO**

the data layer

The database in mongo DB.

The name: schema Clients.

Schema Clients - describes for each member of the health fund his personal details and the subject of the corona.

customer details:

 fullName : String, required

  idCard : String, required

  address : String, required

  dateOfBirth :Date

  mobilePhone : String, required

  telephone : String

  positive\_result\_date :Date

  recovery\_date :Date

  vaccination\_dates :[ Date ]

  vaccine\_manufacturers :[ String ]

the application layer:

Server side - NODE.JS

which contains:

1.The models folder contains the Schema Clients

2.The controllers folder - contains functions:

addClient getAllClients .

3.The routers folder - contains routings for functions in controllers

The presentation layer:

Client side, API – REACT

which includes

**App.js** – main page. Contains the Routings component.

**Index.js** – the page that runs in the browser. Render the App component (once).

**Routings.js** – contains the routings to the various pages.

**Layout.js** – array, the routing bar for:

Manager - button for manager access

Sign In - customer login

Sign Up - customer registration.

**SignUp.js** – registration. Adding a customer to a database.

**SignIn.js** – login to customer details.

**Manager.js** - login to the health insurance manager by entering the code (2552), including access to the following pages:

**Clientsdetails.js** – details of all clients.

**Activeclients.js** – calculating the number of sick clients.

**Unvacciatedclients.js** – calculation of the number of unvaccinated clients.

**Style.css** – project design.

**The connection between the layers**

The database server:

.env file

DB\_CONNECT="mongodb+srv://hilag:hilamongodb@manage.3gvqyed.mongodb.net/?retryWrites=true&w=majority&appName=Manage"

PORT=8000

In the db.js file.

const connect = () => {

  console.log(process.env.DB\_CONNECT)

  if(process.env.DB\_CONNECT)

  mongoose

    .connect(process.env.DB\_CONNECT)//('mongodb://127.0.0.1:27017/cookbookDb')

    .then(() => {

      console.log("Connected to MongoDB");

    })

    .catch((error) => {

      console.error("Unable to connect to MongoDB.", error);

    });

    else

      console.log("You're not connected to MongoDB");

};

Server side client:

API definition endpoints for various operations, such as receiving data, creating data, updating data and deleting data.

getAllClients – displaying all clients from the database.

async getAllClients(req, res) {

    try {

      const client = await clientModel.find();

      if (client instanceof Error) {

        res.status(404).json({ message: client.message, error: true });

      } else {

        res.status(200).json(client);

      }

    } catch (error) {

      res.status(400).json({ message: error });

    }

  }

import axios from "axios";

const baseUrl = 'http://localhost:5000';

const backendService = {

    clients:{

        getAllClients: () => axios.get(`${baseUrl}/clients`).then(response => response.data),

        addClient: (client) => {console.log(client);

            return axios.post(`${baseUrl}/clients/addClient`, client).then(response =>{console.log(response.data); return response.data})}

    }

addClient - adding a client to a database.

async addClient(req,res){

    console.log(req.body)

    try {

        const newClient=await clientModel.create(req.body);

        res.status(201).json(newClient);

      } catch (error) {

        console.error(`error: ${error}`);

        return error;

      }

  }

updateClient - update status of one of the clients.

async updateClient(req, res) {

    try {

      const Client = await clientModel.findOneAndUpdate(req.params.id);

      if (Client instanceof Error) {

        if (Client.message === 'Client\_NOT\_FOUND') {

          return res.status(404).json({ message: 'Client\_NOT\_FOUND' });

        } else {

          return res.status(400).json({ message: Client, error: true });

        }

      }

      return res.status(200).json(Client);

    } catch (error) {

      return res.status(400).json({ message: error.message, error: true });

    }

  }

deleteClient - deleting a client.

async deleteClient(req, res) {

    try {

      const Client = await clientModel.findOneAndDelete({id:req.params.id});

      if (Client instanceof Error) {

        if (Client.message === 'Client\_NOT\_FOUND') {

          return res.status(404).json({ message: 'Client\_NOT\_FOUND' });

        } else {

          return res.status(400).json({ message: Client, error: true });

        }

      }

      return res.status(200).json(Client);

    } catch (error) {

      return res.status(400).json({ message: error.message, error: true });

    }

  }