[Tech Stack 1](#_Toc1876868720)

[Frontend Stack 1](#_Toc2009175093)

[Backend Stack 1](#_Toc2091435152)

[How to install / run 1](#_Toc931196063)

[Setting up the database 1](#_Toc2059318144)

[Setting Up the code 3](#_Toc2059415033)

[Through Visual Studio Code 3](#_Toc1101178408)

[Test Log 4](#_Toc63968289)

[Website Code 5](#_Toc1600460483)

[Styling 5](#_Toc1811795426)

[Home page 5](#_Toc854146819)

[Dashboard 6](#_Toc669408457)

[Health tracker 22](#_Toc1104587348)

[Diary 26](#_Toc1880895008)

[My Profile 29](#_Toc1394319361)

[Forums 34](#_Toc2131292143)

[Forum 37](#_Toc267654332)

[Environment assessment 38](#_Toc407194905)

[Login 41](#_Toc530504150)

[Register 45](#_Toc1885839239)

[Backend 49](#_Toc1282632360)

[Test connection 49](#_Toc1325275497)

[Login 49](#_Toc1530771231)

[Register 51](#_Toc1091845195)

[Edit user 51](#_Toc1903013102)

[Diary 56](#_Toc1976832067)

[Get diary posts by id 56](#_Toc1519287889)

[Diary post 57](#_Toc251355358)

[Health 58](#_Toc579504900)

[Get health posts 58](#_Toc1204215339)

[Get health info by id 59](#_Toc1232302875)

[Forums 59](#_Toc535570255)

[Get all forums 59](#_Toc58548581)

[Get forums by id 60](#_Toc1613284399)

[Search posts 61](#_Toc112693153)

Tech Stack

Frontend Stack

* React
* Mui
* Bootstrap
* Jquery
* Axios
* React-Pro-Sidebar

Backend Stack

* Express
* Mysql

How to install / run

This app uses React, Express and Mysql. If you know how to set up this tech stack then you can do it your own way. Here is the way I do it.

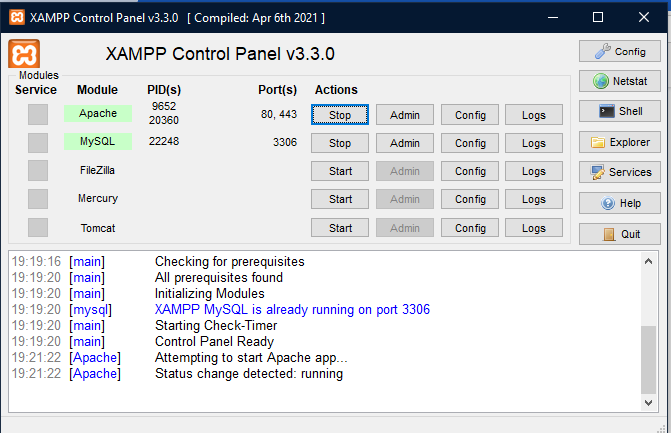
Dependencies

* NPM
* MYSQL
* SQL LOCALHOST SERVER (xampp)
* Node

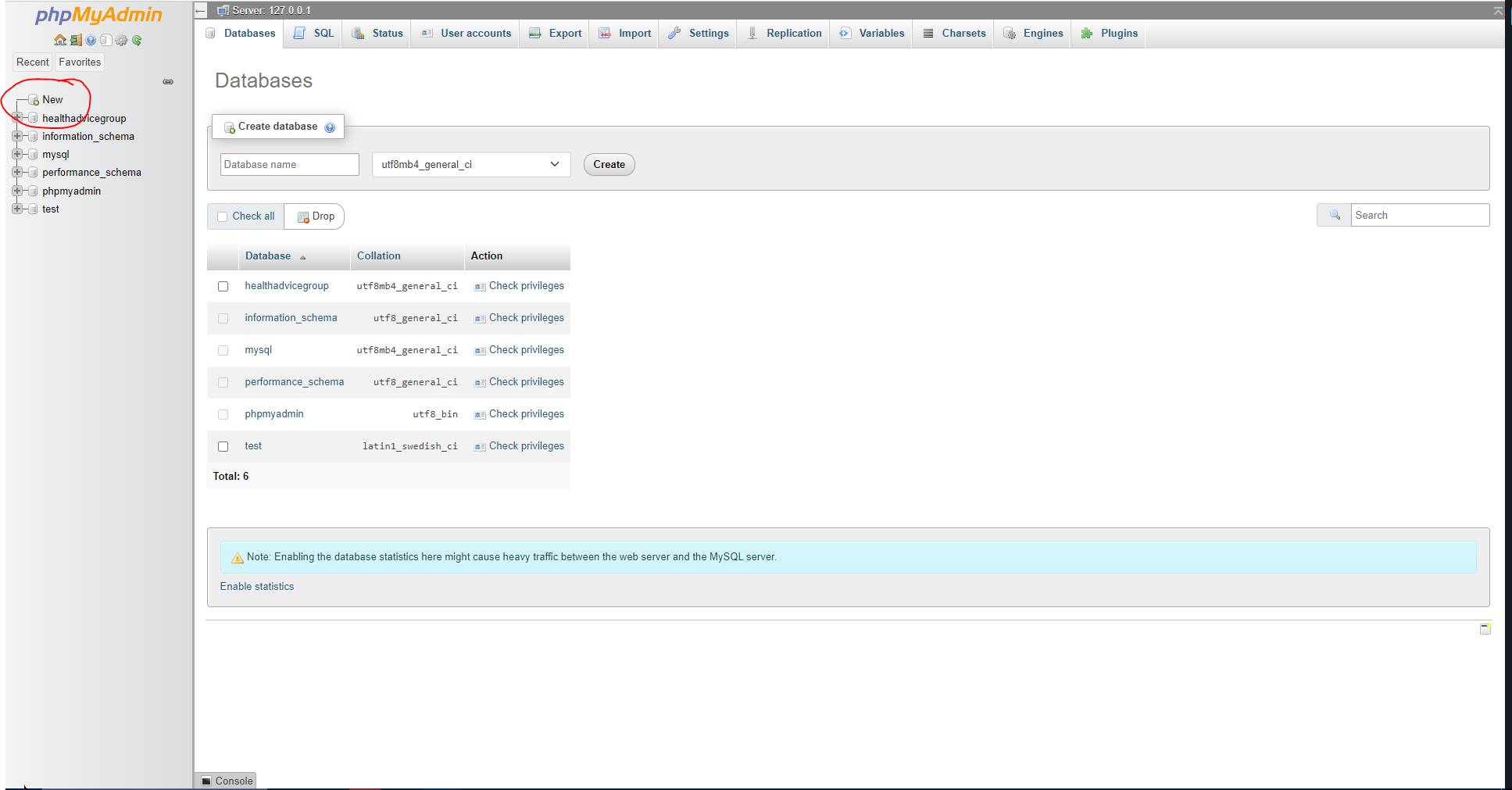
Setting up the database

Through Xampp:

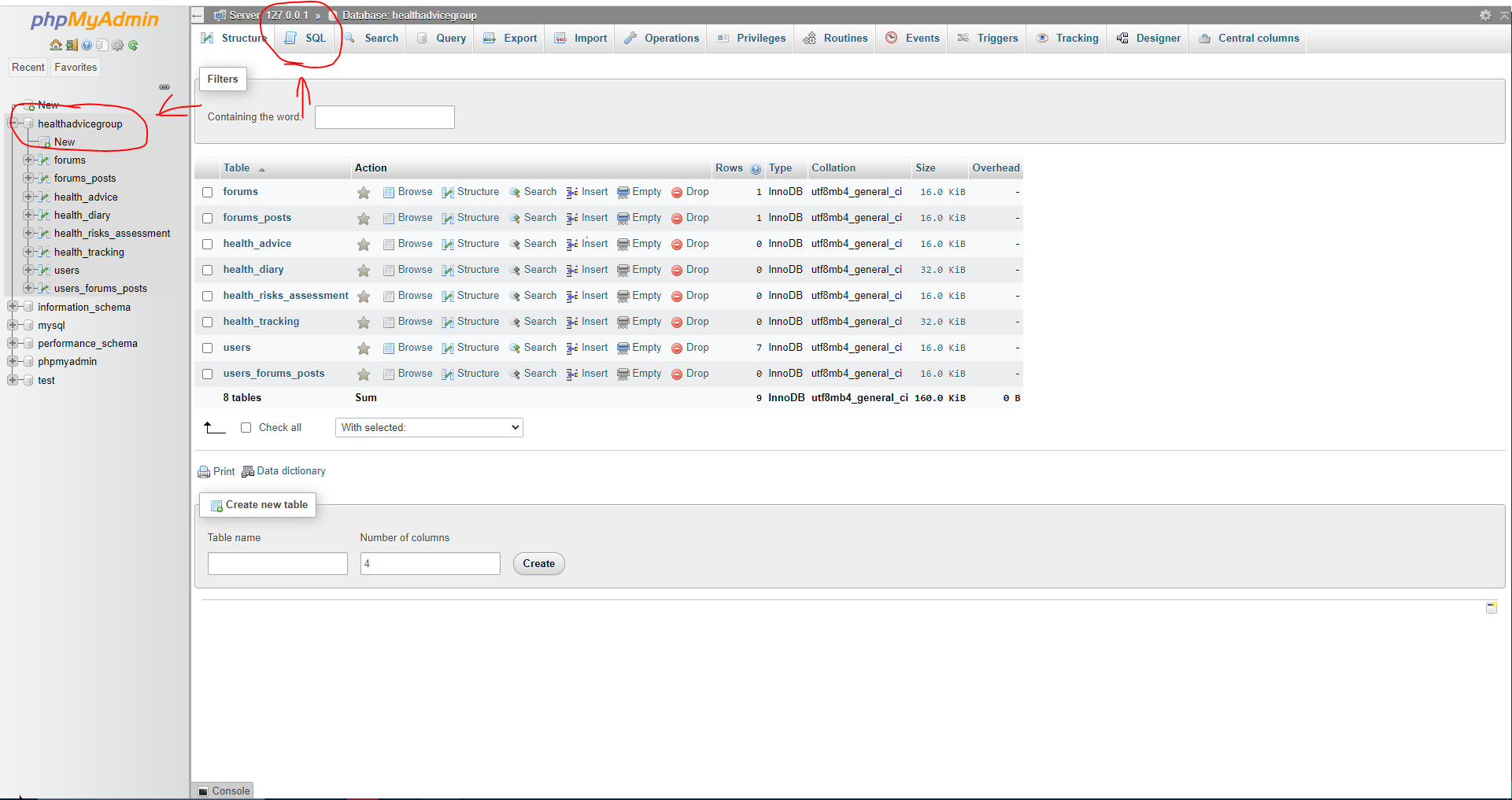
* Download Xampp ( [Here](https://www.apachefriends.org/download.html) )
* Once installed, turn on 2 modules. apache and Mysql.
* They should both be green. If they are not, Please look for a troubleshooting guide



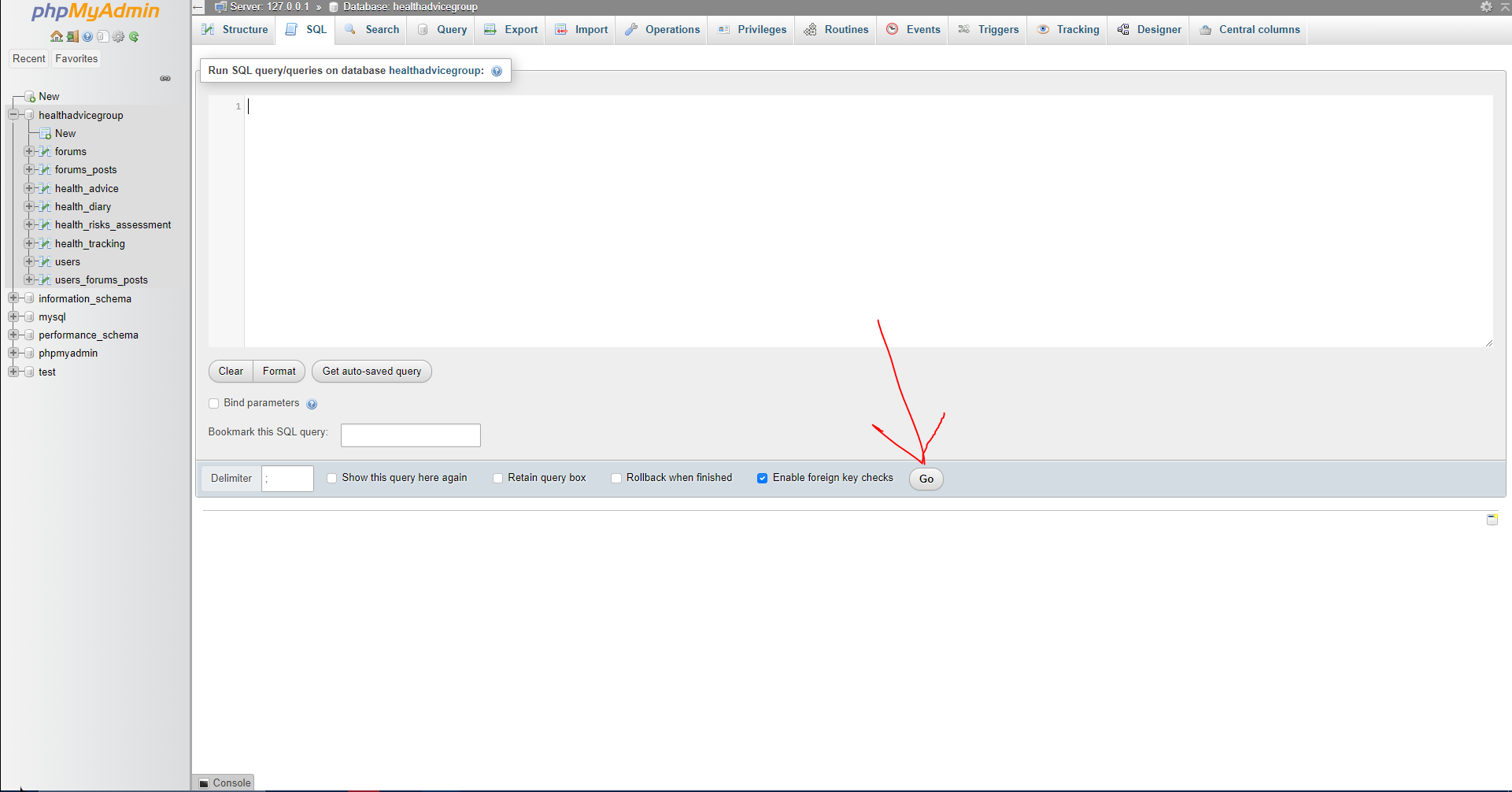
* Navigate to localhost on your browser, either ( [Here](http://localhost:80) ) or ( [Here](http://localhost) )
* Locate php my admin at the top of the page on the nav bar, then click it.
* If you cannot find it or are having trouble try ( [Here](http://localhost/phpmyadmin/) )
* Go over to where it says “New” on the left nav bar.



* Where it says Database name, input exactly “healthadvicegroup”. This is crucial
* After typing it down, press create.
* Click on the database you have just made on the side bar, and then click on “SQL” at the top



* Open up the .sql file in notepad in the folder you was given, and copy everything in there into the box, Then press GO



* You should have no errors from this point onwards, if it is showing up as errors, you may have the wrong version of mysql install or have not installed it in the first place, Please look up a troubleshooting guide if possible.
* You should now have the database running and sorted, Whenever you want to server to work next time, you can turn off apache but you have to keep MySQL running.

## Setting Up the code

### Through Visual Studio Code

* Open Visual studio code to the welcome menu.
* Click on “Open Folder”
* Navigate to the folder you was given, and go to the “Website” folder
* Make sure you have that folder selected and then open it
* Once you have opened it, you should see 2 folders on the left hand side, frontend and backend.
* You are going to want to open 2 terminals. You can do this by going to the top bar and clicking terminal, then “Split Terminal” additionally, you can press CTRL + SHIFT + 5 if you click on the already opened terminal

**Setting up Frontend**

1. Click on 1 terminal

2. Type ` cd frontend `

3. Type ` npm i `

4. Wait to finish installing

5. Npm run start

6. if you dont get redirected, go to [here](http://localhost:3000)

**Setting up Backend**

1. Click on the other terminal

2. Type ` cd frontend `

3. Type ` npm i `

4. Wait to finish installing

5. node app.js

* You should now have it installed and working. If there is something wrong please look up a troubleshooting guide or try the other method

CMD / Batch Script

* In the folder you have gotten, I have made 2 bat scripts. One to install and one to run.
* In install, it will automatically install front end and back end libraries.
* In run, it will automatically start the xampp server ( if it is in c:\xampp)
* It will then run the frontend `npm run start` and backend `node app.js`

Asset Log

Test Log

## Login

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Expected Outcome | Reasons For Test | Actual Outcome |
| Empty Email and password fields | Validation erros appear for both fields | Testing form validation | Validation errors appear as expected. |
| Invalid email and password combo | Error message appears "Wrong email/password combination!" | Testing error handling for incorrect login credentials | Error message appears as expected |
| Valid email and password combination | User data is retrieved and stored in session storage, user is then redirected to the dashboard | Testing successful login process | User is redirected to dashboard as expected |
| Successful token refresh | New token is received and stored in session storage | Testing token refresh process | New token is received and stored in session storage as expected. |
| Server error | Error message appears "Something went wrong" | Testing error handling for server errors | Error message appears as expected and form does not break |
| SQL injection attack  https://github.com/payloadbox/sql-injection-payload-list | Form does not allow SQL injection attack | Testing security measures | Form does not allow SQL injection attack |

# Website Design

## Home Page

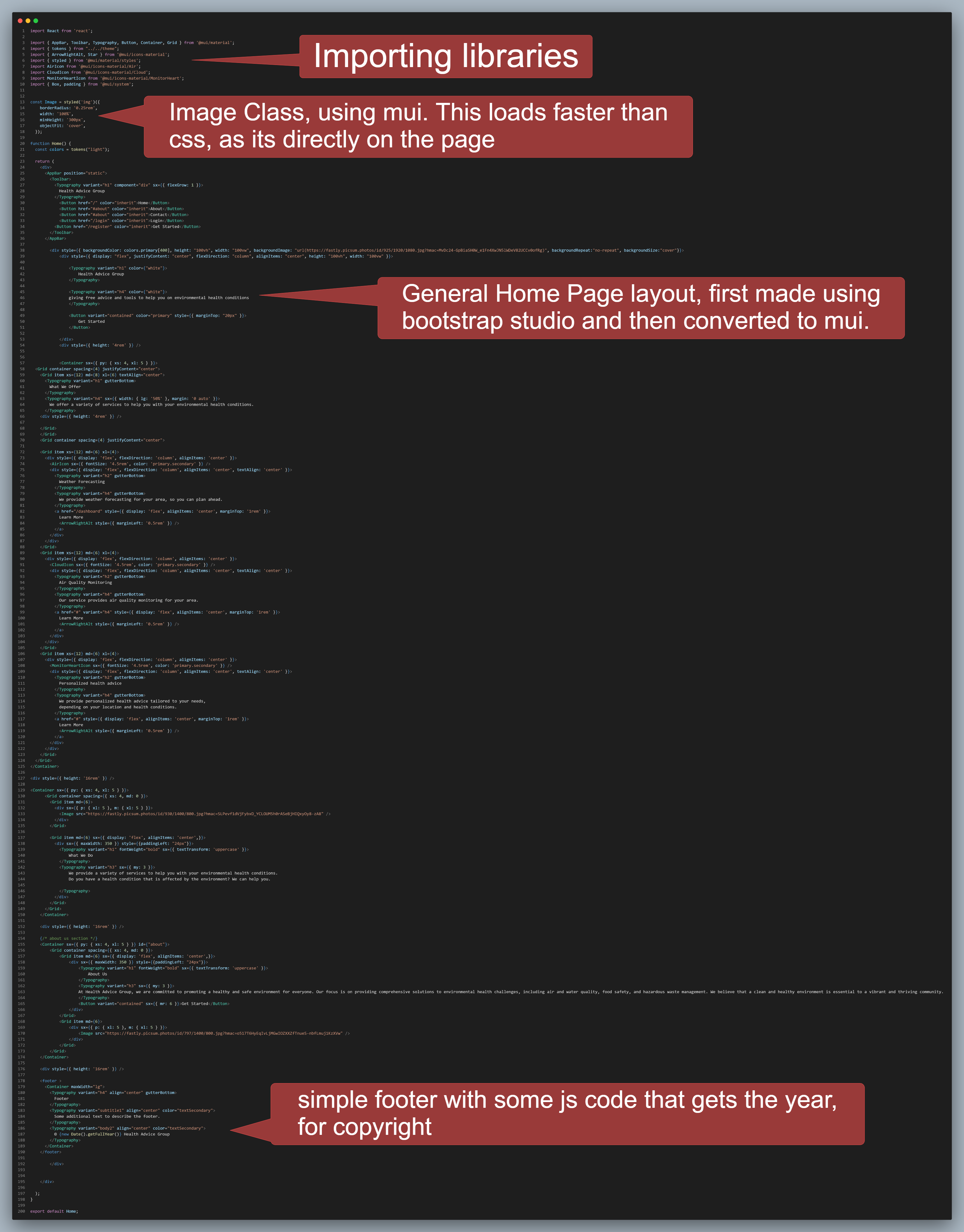
Website Code

The annotations and my comments are encased in [ ], and in black and bold.

## Styling

Front End

### Home page



There is not a lot of code here, just a simple front page landing website.

### Dashboard

|  |
| --- |
| import { Box, Button, IconButton, Typography, useTheme, Autocomplete, TextField, List, ListItemText } from "@mui/material";  import { tokens } from "../../theme";  import DownloadOutlinedIcon from "@mui/icons-material/DownloadOutlined";    import PieChart from "../../components/PieChart";  import Header from "../../components/Header";  import LineChart from "../../components/LineChart";  import StatBox from "../../components/StatBox";  import ForecastCard from "../../components/ForecastCard";    import SearchIcon from "@mui/icons-material/Search";  import CloudIcon from '@mui/icons-material/Cloud';  import AirIcon from '@mui/icons-material/Air';  import WbSunnyIcon from '@mui/icons-material/WbSunny';  import WindPowerIcon from '@mui/icons-material/WindPower';  import InvertColorsIcon from '@mui/icons-material/InvertColors';    import CompressIcon from '@mui/icons-material/Compress';    import ThermostatIcon from '@mui/icons-material/Thermostat';    import { useEffect, useState } from "react";  **[These are the standard imports, that is required on every page.]**  const Dashboard = () => {    const [isGeolocationEnabled, setIsGeolocationEnabled] = useState(false);  **[ Setting the geolocation to false for now, until we call the geolocation effect. ]**    const theme = useTheme();  const colors = tokens(theme.palette.mode);  **[ This sets the theme and colors for the page ]**  const [selectedValue, setSelectedValue] = useState(null);  const [temperature, setTemperature] = useState(0);  const [humidity, setHumidity] = useState(0);  const [dewPoint, setdewPoint] = useState(0);  const [windSpeed, setwindSpeed] = useState(0);  const [windDirection, setwindDirection] = useState(0);  const [uvIndex, setUvIndex] = useState(0);  const [pressure, setPressure] = useState(0);  const [weather, setWeather] = useState(0);  const [data, setData] = useState(null);  const [weatherToday, setWeatherToday] = useState(null);  const [forecastData, setForecastData] = useState([]);  **[ I set all these variables to 0 so that the website will not look blank when the website is loading ]**  const cities = [  "Hanover",  "Aberdeen",  "Echt",  "London",  "Manchester",  "Birmingham",  "Glasgow",  "Liverpool",  "Bristol",  "Edinburgh",  "Leeds",  "Sheffield",  "Dublin",  "Paris",  "Prilep",  ];  **[ This is where we set all of the available cities for the autocomplete mui function. Please note that I intentionally only included a few cities here otherwise the lines would be too long**  **I wanted to include them in this file, instead of defining a “cities” list. But didn’t because it would be better to dynamically load the available cities from openweathermap, however this would eat up far too many api requests ]**  const API\_KEY = "d0654d378c1e5af3cce8c93afb1fa05f"  const GEO\_API\_KEY = "bee6776c35664d63958d0dcaaba3e3b7"  const units = "metric"  **[ Defining what unit to use, and both of the api keys.** **In final development versions, you should be able to switch what units you want to use. ]**  // check if geolocation is enabled  useEffect(() => {  navigator.geolocation.getCurrentPosition(  function(position) {  // The user allowed the request for geolocation  console.log('User allowed geolocation');  setIsGeolocationEnabled(true);  },  function(error) {  if (error.code == error.PERMISSION\_DENIED) {  // The user denied the request for geolocation  console.log('User denied geolocation');  }  }  );  }, []);  **[ On page load, this function will be called. It will try to get the users position in the browsers built in geolocation. If it fails, it knows that the user either has denied the geolocation or the browser does not support it ]**  const getWeatherByCoords = async (lat, lon) => {  **[ This is the main function of the dashboard ]**  const apiUrl = '<https://api.openweathermap.org/data/2.5/weather?lat=>' + lat + '&lon=' + lon + '&appid=' + API\_KEY + '&units=' + units;  const response = await fetch(apiUrl);  const data = await response.json();  const cityName = data.name;  const temperature = data.main.temp;  console.log(`${cityName}: ${temperature}°C`);  setTemperature(temperature);  setHumidity(data.main.humidity);  setwindSpeed(data.wind.speed);  setPressure(data.main.pressure);  setWeather(data.weather[0].main);  setSelectedValue(cityName);    // OneAPI  const exclude = "current,minutely,hourly";    fetch(`https://api.openweathermap.org/data/2.5/onecall?lat=${lat}&lon=${lon}&exclude=${exclude}&appid=${API\_KEY}`)  .then(response => response.json())  .then(data => {    // get uv index  const uvIndex = data.daily[0].uvi;  setUvIndex(uvIndex);    // get wind direction  const windDirection = data.daily[0].wind\_deg;  // convert wind direction to compass direction  function degToCompass(num) {  var val = Math.floor((num / 22.5) + 0.5);  var arr = ["N", "NNE", "NE", "ENE", "E", "ESE", "SE", "SSE", "S", "SSW", "SW", "WSW", "W", "WNW", "NW", "NNW"];  return arr[(val % 16)];  }  const windDirectionCompass = degToCompass(windDirection);  setwindDirection(windDirectionCompass);  **[This function is a really complex one. The wind given by the website is in a 360 degree range, so this function uses a formula to figure out the direction in nesw format ]**    setdewPoint(data.daily[0].dew\_point);  let weatherCounts = {};  data.daily.forEach(day => {  const weather = day.weather[0].main;  if (weatherCounts[weather]) {  weatherCounts[weather] += 1;  } else {  weatherCounts[weather] = 1;  }  });  // convert object to array  weatherCounts = Object.keys(weatherCounts).map(key => {  return {  id: key,  label: key,  value: weatherCounts[key],  color: 'hsl(0, 70%, 50%)'  }  });  **[ this code takes in weather data and creates a count of how many times each type of weather occurred. It then converts this count into an array of objects that can be easily used for visualization ( the pie chart) ]**    setWeatherToday(weatherCounts)  });    // Forecast API    // Set the API endpoint and parameters  const api2Url = '<https://api.openweathermap.org/data/2.5/forecast>';    // Construct the API URL  const apiUrlWithParams = `${api2Url}?lat=${lat}&lon=${lon}&appid=${API\_KEY}&units=${units}`;    // Fetch the data from the API  fetch(apiUrlWithParams)  .then(response => response.json())  .then(data => {  const dailyForecasts = data.list.filter((forecast, index) => index % 8 === 0);  setForecastData(dailyForecasts);  const tempData = {  id: cityName,  color: tokens("dark").greenAccent[500],  data: []  };    // Loop over 24-hour forecast data and format it for the chart  for (let i = 0; i < data.list.length && i < 24; i++) {  const forecast = data.list[i];  const hour = new Date(forecast.dt \* 1000).getHours();  tempData.data.push({  x: `${hour}h`,  y: forecast.main.temp  });  }    var newData = [tempData]  setData(newData);  })  .catch(error => console.error(error));  }  **[This code fetches weather forecast data for a selected location from an API, specifically the OpenWeatherMap API, based on a selectedValue state variable. The fetched data is then filtered to get daily forecasts and is processed to format it for use in a chart. The chart will show the hourly temperature for the next 24 hours. Finally, the formatted data is set as the state variable for the chart. If there is an error fetching the data, an error message is logged to the console.]**  useEffect(() => {  console.log("Starting geolocation...")  if (navigator.geolocation) {  navigator.geolocation.getCurrentPosition((position) => {  console.log("Geolocation is supported by this browser.");  getWeatherByCoords(position.coords.latitude, position.coords.longitude);  });  } else {  setIsGeolocationEnabled(false);  console.log("Geolocation is not supported by this browser.");    }  }, []);  **[This code extracts weather data from the data object and sets various state variables based on that data. It sets the temperature state variable based on the data.main.temp value, the weather state variable based on the item.main value in the data.weather array, the humidity state variable based on the data.main.humidity value, and the pressure state variable based on the data.main.pressure value. It also sets the windSpeed variable based on the data.wind.speed value, and the country variable based on the data.sys.country value.]**    const getWeather = async () => {  if (!selectedValue) {  return;  }    const generalapi = await fetch(`https://api.openweathermap.org/data/2.5/weather?q=${selectedValue}&appid=${API\_KEY}&units=${units}`)  const data = await generalapi.json()  const cityName = data.name;  const temperature = data.main.temp;    // Set the API endpoint and parameters  const apiUrl = '<https://api.openweathermap.org/data/2.5/forecast>';    // Construct the API URL  const apiUrlWithParams = `${apiUrl}?q=${selectedValue}&appid=${API\_KEY}&units=${units}`;    // Fetch the data from the API  fetch(apiUrlWithParams)  .then(response => response.json())  .then(data => {    const dailyForecasts = data.list.filter((forecast, index) => index % 8 === 0);    setForecastData(dailyForecasts);    const tempData = {  id: selectedValue,  color: tokens("dark").greenAccent[500],  data: []  };    // Loop over 24-hour forecast data and format it for the chart  for (let i = 0; i < data.list.length && i < 24; i++) {  const forecast = data.list[i];  const hour = new Date(forecast.dt \* 1000).getHours();  tempData.data.push({  x: `${hour}h`,  y: forecast.main.temp  });  }      var newData = [tempData]  setData(newData);  })  .catch(error => console.error(error));    let weather = "clear";  let weatherLastHour = 0;  data.weather.forEach(item => {  if (item.main === "Rain") {  weather = "rain";  } else if (item.main === "Snow") {  weather = "snow";  if (item.snow && item.snow["1h"]) {  weatherLastHour = item.snow["1h"];  }  }  });  setTemperature(temperature);  setWeather(weather);  setHumidity(data.main.humidity);  setPressure(data.main.pressure);  const windSpeed = data.wind.speed;  const country = data.sys.country;    console.log(`${cityName}, ${country}: ${temperature}°C, ${weather}, ${weatherLastHour} cm last hour, wind speed ${windSpeed} m/s`);    // Pie Chart    // Get Lat and Lon from selectedValue    const openCageAPI = await fetch(`https://api.opencagedata.com/geocode/v1/json?q=${selectedValue}&key=${GEO\_API\_KEY}`)  const openCageData = await openCageAPI.json()  const lat = openCageData.results[0].geometry.lat;  const lon = openCageData.results[0].geometry.lng;  **[ This code is almost the same as the “getweatherbycoords” function, however it gets the selected value ( the city selected ) and gets the lat and lon for that.then it goes through the same function.**   **This in the future can easily be optimized by adding a check ]**    // OneAPI  const exclude = "current,minutely,hourly";    fetch(`https://api.openweathermap.org/data/2.5/onecall?lat=${lat}&lon=${lon}&exclude=${exclude}&appid=${API\_KEY}`)  .then(response => response.json())  .then(data => {    const uvIndex = data.daily[0].uvi;  setUvIndex(uvIndex);    // get wind direction  const windDirection = data.daily[0].wind\_deg;  // convert wind direction to compass direction  function degToCompass(num) {  var val = Math.floor((num / 22.5) + 0.5);  var arr = ["N", "NNE", "NE", "ENE", "E", "ESE", "SE", "SSE", "S", "SSW", "SW", "WSW", "W", "WNW", "NW", "NNW"];  return arr[(val % 16)];  }  const windDirectionCompass = degToCompass(windDirection);  setwindDirection(windDirectionCompass);    setdewPoint(data.daily[0].dew\_point);    let weatherCounts = {};  data.daily.forEach(day => {  const weather = day.weather[0].main;  if (weatherCounts[weather]) {  weatherCounts[weather] += 1;  } else {  weatherCounts[weather] = 1;  }  });  // convert object to array  weatherCounts = Object.keys(weatherCounts).map(key => {  return {  id: key,  label: key,  value: weatherCounts[key],  color: 'hsl(0, 70%, 50%)'  }  });    setWeatherToday(weatherCounts);  console.log(weatherCounts)  })    }  return (  **[ this is the html / jsx ]**  <Box m="20px">  {/\* HEADER \*/}  <Box display="flex" justifyContent="space-between" alignItems="center">  <Header title="DASHBOARD" subtitle="Welcome to your dashboard" />  <Box>  </Box>  </Box>    {/\* WEATHER SEARCH \*/}  <Box display="flex" justifyContent="space-between" p={2}>  {/\* SEARCH BAR \*/}  <Box  display="flex"  bgcolor="primary.400"  borderRadius="3px"  >  <Autocomplete  disablePortal  id="combo-box-demo"  options={cities.filter((city) => city.toLowerCase().startsWith(selectedValue?.toLowerCase() || ''))}  limitTags={5}  sx={{ width: 300 }}  value={selectedValue}  onChange={(event, newValue) => {  setSelectedValue(newValue);  }}  renderInput={(params) => <TextField {...params} label="City" />}  />  <IconButton type="button" sx={{ p: 1 }} onClick={getWeather}>  <SearchIcon />  </IconButton>  </Box>  </Box>  **[ The first section of the code sets up a header for the dashboard, with the title "DASHBOARD" and the subtitle "Welcome to your dashboard". This is followed by a search bar, which allows the user to search for weather data for a specific city. The search bar is implemented using the Autocomplete component from the Material-UI library. ]**  {/\* GRID & CHARTS \*/}  **[ The next section of the code sets up a grid that displays various weather statistics for the city that the user has searched for. The grid consists of three rows and four columns, with each column containing a weather statistic. The first row displays the current temperature, the current weather condition, the current humidity, and the current air pressure. ]**  <Box  display="grid"  gridTemplateColumns="repeat(12, 1fr)"  gridAutoRows="140px"  gap="20px"  >  {/\* ROW 1 \*/}  <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title={temperature}  subtitle="Temperature"  icon={  <ThermostatIcon  sx={{ color: colors.greenAccent[600], fontSize: "32px" }}  />  }  />  </Box>  <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title={weather}  subtitle="Current Weather"  icon={  <CloudIcon  sx={{ color: colors.greenAccent[600], fontSize: "26px" }}  />  }  />  </Box>  <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title={humidity + "%"}  subtitle="Humidity"  icon={  <InvertColorsIcon  sx={{ color: colors.greenAccent[600], fontSize: "26px" }}  />  }  />  </Box>  <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title={pressure + " hPa"}  subtitle="Pressure"  progress="0.80"  increase="+43%"  icon={  <CompressIcon  sx={{ color: colors.greenAccent[600], fontSize: "26px" }}  />  }  />  </Box>    {/\* ROW 2 \*/}  **[ The second row displays the current UV index, wind direction, wind speed, and dew point.]**  <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title= {uvIndex}  subtitle="UV Index"  icon={  <WbSunnyIcon  sx={{ color: colors.greenAccent[600], fontSize: "32px" }}  />  }  />  </Box>    <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title= {windDirection}  subtitle="Wind Direction"  icon={  <AirIcon  sx={{ color: colors.greenAccent[600], fontSize: "32px" }}  />  }  />  </Box>    <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title= {windSpeed + " m/s"}  subtitle="Wind Speed"  icon={  <WindPowerIcon  sx={{ color: colors.greenAccent[600], fontSize: "32px" }}  />  }  />  </Box>    <Box  gridColumn="span 3"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="center"  >  <StatBox  title= {dewPoint}  subtitle="Dew Point"  icon={  <ThermostatIcon  sx={{ color: colors.greenAccent[600], fontSize: "32px" }}  />  }  />  </Box>    {/\* ROW 3 \*/}  **[ Finally, the third row displays a chart of the temperature over time. ]**  <Box  gridColumn="span 8"  gridRow="span 2"  backgroundColor={colors.primary[400]}  >  <Box  mt="25px"  p="0 30px"  display="flex "  justifyContent="space-between"  alignItems="center"  >  <Box>  <Typography  variant="h5"  fontWeight="600"  color={colors.grey[100]}  >  Temperature  </Typography>  <Typography  variant="h3"  fontWeight="bold"  color={colors.greenAccent[500]}  >  Currently: {temperature}  </Typography>  </Box>  <Box>  <IconButton>  <DownloadOutlinedIcon  sx={{ fontSize: "26px", color: colors.greenAccent[500] }}  />  </IconButton>  </Box>  </Box>  <Box height="250px" m="-20px 0 0 0">  <LineChart isDashboard={true} data={data}/>  </Box>  </Box>    {/\* ROW 4 \*/}  <Box  gridColumn="span 4"  gridRow="span 2"  backgroundColor={colors.primary[400]}  p="30px"  >  <Box m="10px">  <Header title="Weather Today"/>  <Box height="200px">  <PieChart data={weatherToday}/>  </Box>  </Box>  </Box>  <Box  gridColumn="span 8"  gridRow="span 2"  backgroundColor={colors.primary[400]}  display="flex"  alignItems="center"  justifyContent="start"  >  <Box m="20px">  <Header title="Weekly Forecast"/>  <Box display="flex" justifyContent="space-between" flexWrap="wrap">    {forecastData.map(forecast => (  <Box m="10px">    <ForecastCard  key={forecast.dt}  date={new Date(forecast.dt \* 1000).toLocaleDateString('en-US', { weekday: 'long' })}  temperature={`${forecast.main.temp} °C`}  icon={<img src={`https://openweathermap.org/img/wn/${forecast.weather[0].icon}.png`} alt={forecast.weather[0].description} />}  />  </Box>  ))}  </Box>  </Box>  </Box>  </Box>  </Box>  **[ Each weather statistic is displayed in a box with a title, a subtitle, and an icon. For example, the temperature box has an icon of a thermostat, and the wind speed box has an icon of a wind power symbol. The temperature box also displays the current temperature in degrees Celsius, while the wind speed box displays the current wind speed in meters per second. ]**    );  };    export default Dashboard; |

### Health tracker

|  |
| --- |
| import React, { useState, useEffect } from 'react';  import { Box, Button, Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow, TextField, Typography } from '@mui/material';  import axios from 'axios';    function HealthTracker() {  const userData = JSON.parse(sessionStorage.getItem('UserData'));  **[ this grabs the user data from the session storage, we are going to use this to grab the health messages in the database ]**  const userId = userData.user\_id;    const [height, setHeight] = useState('');  const [weight, setWeight] = useState('');  const [bloodPressure, setBloodPressure] = useState('');  const [heartRate, setHeartRate] = useState('');  const [healthPosts, setHealthPosts] = useState([]);  **[ this is setting all of the constants needed using react usestates ]**    const handleHeightChange = (event) => {  setHeight(event.target.value);  };    const handleWeightChange = (event) => {  setWeight(event.target.value);  };    const handleBloodPressureChange = (event) => {  setBloodPressure(event.target.value);  };    const handleHeartRateChange = (event) => {  setHeartRate(event.target.value);  };  **[ each function takes an event object as input and calls a corresponding set function (e.g., setHeight) with the value of the changed input element (e.g., event.target.value). The set functions are part of the useState hook, which updates the state of the component and triggers a re-render of the user interface. By using these functions, the user's input for each health parameter can be tracked and updated dynamically in the application. ]**  const handleSubmit = (event) => {  event.preventDefault();  **[ prevents the default form submission behavior so that the page does not reload. ]**  const data = {  userId: userId,  height: height,  weight: weight,  bmi: (weight / ((height / 100) \*\* 2)).toFixed(2),  blood\_pressure: bloodPressure,  heart\_rate: heartRate,  };  **[ creates an object with the data from the form. The object includes the userId, height, weight, bmi, blood\_pressure, and heart\_rate properties. The bmi property is calculated using the formula weight / ((height / 100) \*\* 2) and is rounded to 2 decimal places using the toFixed() method. ]**  axios  .post('http://localhost:3001/healthpost', data)  .then((res) => {  console.log(res.data.message);  setHeight('');  setWeight('');  setBloodPressure('');  setHeartRate('');  // reload health posts  axios  .get(`http://localhost:3001/gethealthinfo/${userId}`)  .then((res) => {  setHealthPosts(res.data.posts);  });  })  .catch((err) => console.log(err));  };  **[ The handleSubmit function prevents a form from reloading the page upon submission. It then creates an object called data with health-related information submitted via the form, sends a POST request to a local server with that data, and updates the state of the form with empty values. It also sends a GET request to retrieve the updated health posts and updates the state of the health posts with that data. If there is an error, it logs the error to the console. ]**  useEffect(() => {  axios  .get(`http://localhost:3001/gethealthinfo/${userId}`)  .then((res) => {  setHealthPosts(res.data.posts);  })  .catch((err) => console.log(err));  }, [userId]);  **[The useEffect hook is used to fetch health-related data from a local server when a component is mounted or when the userId state variable changes. It uses the axios library to send a GET request to the local server at http://localhost:3001/gethealthinfo/ with the userId as a parameter. When the response is received, the function updates the healthPosts state variable with the posts data in the response. If there is an error, it logs the error to the console.]**  return (  **[ this is the html / jsx part.**  **This code basically renders a form that includes four text fields for height, weight, blood pressure, and heart rate, respectively. The value properties of the text fields are initialized with the height, weight, bloodPressure, and heartRate state variables, respectively. Each text field has an onChange event handler that updates its respective state variable when the user types in a value. ]**  <Box sx={{ p: 2 }}>  <Typography variant="h4">Health Tracker</Typography>  <form onSubmit={handleSubmit}>  **[ The onSubmit event of the form is handled by the handleSubmit function, which is defined elsewhere. When the user submits the form by clicking the "Submit" button, the handleSubmit function is executed. It sends a POST request to a local server with the data entered in the form, and updates the state of the form with empty values. ]**  <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2 }}>  <TextField  label="Height (cm)"  variant="outlined"  value={height}  onChange={handleHeightChange}  />  <TextField  label="Weight (kg)"  variant="outlined"  value={weight}  onChange={handleWeightChange}  />  <TextField  label="Blood Pressure"  variant="outlined"  value={bloodPressure}  onChange={handleBloodPressureChange}  />  <TextField  label="Heart Rate"  variant="outlined"  value={heartRate}  onChange={handleHeartRateChange}  />  <Button type="submit" variant="contained" sx={{ mt: 2 }}>  Submit  </Button>  </Box>  </form>  <TableContainer component={Paper} sx={{ mt: 4 }}>  <Table>  <TableHead>  <TableRow>  <TableCell>Date</TableCell>  <TableCell>Height (cm)</TableCell>  <TableCell>Weight (kg)</TableCell>  <TableCell>BMI</TableCell>  <TableCell>Blood Pressure</TableCell>  <TableCell>Heart Rate</TableCell>  </TableRow>  </TableHead>  <TableBody>  {healthPosts.map((post) => (  <TableRow key={post.id}>  <TableCell>{new Date(post.timestamp).toLocaleDateString()}</TableCell>  <TableCell>{post.height}</TableCell>  <TableCell>{post.weight}</TableCell>  <TableCell>{post.bmi}</TableCell>  <TableCell>{post.blood\_pressure}</TableCell>  <TableCell>{post.heart\_rate}</TableCell>  </TableRow>  ))}  </TableBody>  </Table>  **[ This is just a simple table that outputs all of the data from the database onto it.]**  </TableContainer>  </Box>  );  }    export default HealthTracker; |

### Diary

|  |
| --- |
| import React, { useEffect, useState } from 'react';  import { Table, TableBody, TableCell, TableContainer, TableHead, TableRow, Paper,Button, TextField, Typography } from '@mui/material';  import axios from 'axios';  import useMediaQuery from "@mui/material/useMediaQuery";  import { Box } from '@mui/system';  **[ Importing all the necessary modules ]**  function Diary() {  **[ this has the same functionality as the healthtracker, as in it gets the data from the user ids. ]**    const isNonMobile = useMediaQuery("(min-width:600px)");  **[Defining a boolean variable isNonMobile that stores the result of a media query.]**    // get user id from session storage  const userData = JSON.parse(sessionStorage.getItem("UserData"));  const userId = userData.user\_id;  [ Grabbing and saving the user data again,,    const [diaryPosts, setDiaryPosts] = useState([]);  const [type, setType] = useState('');  const [message, setMessage] = useState('');    const handleTypeChange = (event) => {  setType(event.target.value);  };    const handleMessageChange = (event) => {  setMessage(event.target.value);  };    const handleSubmit = (event) => {  **[ This function handles the submission of the diary posting.]**  event.preventDefault();  const data = {  userId: userId,  type: type,  message: message,  };  **[ this object is the one that gets sent to the backend ]**  // check if type and message are not empty  if (type === '' || message === '') {  alert('Please fill in all fields');  return;  } else {  axios.post('http://localhost:3001/diarypost', data)  .then(res => {  console.log(res.data.message);  setType('');  setMessage('');  // reload diary posts  axios.get(`http://localhost:3001/getdiaryposts/${userId}`)  .then(res => {  setDiaryPosts(res.data.posts);  }  )  **[ If the type and message fields are not empty, the data is sent to the server, and the function then updates the state of the component by resetting the type and message fields, and retrieves updated diary posts from the server using a GET request.]**    })  .catch(err => console.log(err));    }      };    useEffect(() => {  [This effect gets the backend api, and gets the diary posts, by user id]  axios.get(`http://localhost:3001/getdiaryposts/${userId}`)  .then(res => {  setDiaryPosts(res.data.posts);  })  .catch(err => console.log(err));  }, [userId]);    return (  <Box sx={{ p: 2 }}>  <Typography variant="h4" sx={{ mb: 2 }}>  Diary  </Typography>    <TableContainer component={Paper}>  <Table aria-label="diary post table">  <TableHead>  <TableRow>  <TableCell>ID</TableCell>  <TableCell>User ID</TableCell>  <TableCell>Type</TableCell>  <TableCell>Message</TableCell>  </TableRow>  </TableHead>  <TableBody>  {diaryPosts.map((post) => (  <TableRow key={post.id}>  <TableCell component="th" scope="row">  {post.id}  </TableCell>  <TableCell>{post.user\_id}</TableCell>  <TableCell>{post.type}</TableCell>  <TableCell>{post.message}</TableCell>  </TableRow>  ))}  </TableBody>  </Table>  </TableContainer>    <Typography variant="h4" sx={{ mt: 2, mb: 2 }}>  Add a new post  </Typography>    <form onSubmit={handleSubmit}>  <Box  display="grid"  gap="30px"  gridTemplateColumns="repeat(4, minmax(0, 1fr))"  sx={{  "& > div": { gridColumn: isNonMobile ? undefined : "span 4" },  }}  >    <TextField  id="type"  label="Type"  variant="filled"  margin="normal"  value={type}  onChange={handleTypeChange}  />  <TextField  id="message"  label="Message"  variant="filled"  margin="normal"  value={message}  onChange={handleMessageChange}  />  <Button type="submit" color="secondary" variant="contained">  Submit  </Button>    </Box>  </form>  </Box>  );  }    export default Diary; |

### My Profile

|  |
| --- |
| import { Box, Button, TextField } from "@mui/material";  import { Formik } from "formik";  import \* as yup from "yup";  import useMediaQuery from "@mui/material/useMediaQuery";  import Header from "../../components/Header";  import Axios from "axios";  **[Importing necessary dependencies and components]**    const Profile = () => {  const isNonMobile = useMediaQuery("(min-width:600px)");  **[ defining the profile component and initializing the “isnonmobile” variable using the “usemediaquery” hook. This is for mobile responsiveness.]**    const handleFormSubmit = (values) => {  // get user id from session storage  const userData = JSON.parse(sessionStorage.getItem("UserData"));  const userId = userData.user\_id;  console.log(userData)  [ This does the same thing as all of the other userdatas ]    Axios.post("http://localhost:3001/edituser", {  userId: userId,  firstName: values.firstName,  lastName: values.lastName,  email: values.email,  password: values.password,  confirmPassword: values.confirmPassword,  }).then((response) => {  const first\_name = response.data.user.first\_name  const last\_name = response.data.user.last\_name  const email = response.data.user.email  const profile\_pic = response.data.user.profile\_pic  const authorization\_level = response.data.user.authorization\_level  const id = response.data.user.id    const userData = {  user\_id: id,  first\_name: first\_name,  last\_name: last\_name,  email: email,  profile\_pic: profile\_pic,  authorization\_level: authorization\_level  }  sessionStorage.setItem("UserData", JSON.stringify(userData))    window.location.href = "/profile"  }, (error) => {  console.log(error);  });  };    return (  <Box m="20px">  <Header title="User Profile" subtitle="Change User Profile" />    <Formik  onSubmit={handleFormSubmit}  initialValues={initialValues}  validationSchema={checkoutSchema}  >  {({  values,  errors,  touched,  handleBlur,  handleChange,  handleSubmit,  }) => (  <form onSubmit={handleSubmit}>  <Box  display="grid"  gap="30px"  gridTemplateColumns="repeat(4, minmax(0, 1fr))"  sx={{  "& > div": { gridColumn: isNonMobile ? undefined : "span 4" },  }}  >  <TextField  fullWidth  variant="filled"  type="text"  label="First Name"  onBlur={handleBlur}  onChange={handleChange}  value={values.firstName}  name="firstName"  error={!!touched.firstName && !!errors.firstName}  helperText={touched.firstName && errors.firstName}  sx={{ gridColumn: "span 2" }}  />  <TextField  fullWidth  variant="filled"  type="text"  label="Last Name"  onBlur={handleBlur}  onChange={handleChange}  value={values.lastName}  name="lastName"  error={!!touched.lastName && !!errors.lastName}  helperText={touched.lastName && errors.lastName}  sx={{ gridColumn: "span 2" }}  />  <TextField  fullWidth  variant="filled"  type="text"  label="Password"  onBlur={handleBlur}  onChange={handleChange}  value={values.password}  name="firstName"  error={!!touched.password && !!errors.password}  helperText={touched.password && errors.password}  sx={{ gridColumn: "span 2" }}  />  <TextField  fullWidth  variant="filled"  type="text"  label="Confirm Password"  onBlur={handleBlur}  onChange={handleChange}  value={values.confirmPassword}  name="lastName"  error={!!touched.confirmPassword && !!errors.confirmPassword}  helperText={touched.confirmPassword && errors.confirmPassword}  sx={{ gridColumn: "span 2" }}  />    <TextField  fullWidth  variant="filled"  type="text"  label="Email"  onBlur={handleBlur}  onChange={handleChange}  value={values.email}  name="email"  error={!!touched.email && !!errors.email}  helperText={touched.email && errors.email}  sx={{ gridColumn: "span 4" }}  />  <TextField  fullWidth  variant="filled"  type="text"  label="Profile Picture"  onBlur={handleBlur}  onChange={handleChange}  value={values.profilePicture}  name="contact"  error={!!touched.profilePicture && !!errors.profilePicture}  helperText={touched.profilePicture && errors.profilePicture}  sx={{ gridColumn: "span 4" }}  />  </Box>  <Box display="flex" justifyContent="end" mt="20px">  <Button type="submit" color="secondary" variant="contained">  Submit Changes  </Button>  </Box>  </form>  )}  </Formik>  </Box>  );  };    const checkoutSchema = yup.object().shape({    });  const initialValues = {  firstName: "",  lastName: "",  password: "",  confirmPassword: "",  email: "",  profilePicture: "",  };    export default Profile; |

### Forums

|  |
| --- |
| import { useState, useEffect } from 'react';  import { Link } from 'react-router-dom';  import { Box, TextField, Typography,IconButton, } from '@mui/material';  import SearchIcon from "@mui/icons-material/Search";  import axios from 'axios';    function Forums() {  const [forums, setForums] = useState([]);  const [search, setSearch] = useState('');    const Search = () => {  axios.get('http://localhost:3001/searchposts/:search')  .then(response => {  setForums(response.data.forums);  })  .catch(error => {  console.log(error);  });  }  **[ This code defines a function component named Search that makes a GET request to a server API endpoint at http://localhost:3001/searchposts/:search using the axios library.**    **Once the server responds with data, the response object is used to extract the forums data and update the state using the setForums function.**    **If there is an error during the request, the catch block logs the error to the console.**    **Note that this code does not get executed on its own and needs to be called by an event or a function trigger to perform the API request. ]**    useEffect(() => {  axios.get('http://localhost:3001/getforums')  .then(response => {  setForums(response.data.forums);  })  .catch(error => {  console.log(error);  });  }, []);  **[This code sets up a side effect in a React functional component using the useEffect hook. The useEffect hook is used to perform side effects, such as fetching data, in a React component.**    **In this case, when the component mounts, the useEffect hook sends an HTTP GET request to http://localhost:3001/getforums using the Axios library to fetch the list of forums. The response data is then used to update the forums state using the setForums function, which causes the component to re-render with the new data.**    **The second argument to useEffect is an empty array [], which means the effect only runs once when the component mounts and not on subsequent re-renders.]**    return (  <Box sx={{ p: 2 }}>  {/\* make input box, with mui \*/}    <Box sx={{alignItems: "center", display: "flex"}}>  <TextField  id="outlined-basic"  label="Search"  variant="outlined"  value={search}  onChange={e => setSearch(e.target.value)}  />  <IconButton onClick={Search}>  <SearchIcon />  </IconButton>  </Box>  **[This part creates an input box for the user to enter their search query. It uses the Material-UI Box and TextField components to create a search box with a label "Search", an outlined border and an icon button to execute the search when clicked.]**    <Box sx={{ display: 'flex', flexWrap: 'wrap' }}>    {forums.map(forum => (  <Box  key={forum.id}  sx={{  display: 'flex',  flexDirection: 'column',  alignItems: 'center',  justifyContent: 'center',  width: '300px',  height: '300px',  border: '1px solid #000',  borderRadius: '10px',  padding: '10px',  margin: '10px',  textDecoration: 'none',  }}  >  <Link to={`/forum/${forum.id}`} style={{ textDecoration: 'none' }}>  <Typography variant="h5" sx={{ mb: 2 }}>  {forum.forum\_title}  </Typography>  <Typography variant="body1" sx={{ mb: 2 }}>  {forum.forum\_text}  </Typography>  <img src={forum.forum\_thumbnail} alt={forum.forum\_title} style={{ width: '100%' }} />  <Typography variant="body1" sx={{ mb: 2 }}>  Tags: {forum.forum\_tags}  </Typography>  </Link>  </Box>  ))}  </Box>  **[The second part is responsible for displaying the list of forums that match the user's search query. It uses the Material-UI Box, Link, and Typography components to create a box for each forum, with its title, description, thumbnail, and tags. The forums are retrieved from the back-end using axios, stored in the forums state variable using setForums, and mapped to individual boxes using .map(). Each forum box is wrapped in a Link component that leads to that forum's page, and each piece of forum information is displayed using the Typography component.]**  </Box>  );  }    export default Forums; |

### Forum

|  |
| --- |
| import { useState, useEffect } from 'react';  import { useParams } from 'react-router-dom';  import { Box, Typography } from '@mui/material';  import Axios from 'axios';    function Forum(forum\_title) {  **[ This forum is an extension of forums. This is the actual post.]**  const { forumId } = useParams();  const { id } = useParams();  **[The component starts by destructuring the forumId and id from the URL parameters. It then defines multiple state variables using the useState hook: forum, forumText, forumTitle, forumIdd, and forumUser.]**      const [forum, setForum] = useState(null);  const [forumText, setForumText] = useState(null);  const [forumTitle, setForumTitle] = useState(null);  const [forumIdd, setForumIdd] = useState(null);  const [forumUser, setForumUser] = useState(null);    useEffect(() => {  // Fetch the forum data from the server  Axios.get(`http://localhost:3001/getforum/${id}`)  .then(response => {  console.log(response);  console.log(response.data.posts[0].post\_text);  setForumTitle(response.data.forum.forum\_title);  setForumText(response.data.posts[0].post\_text);    })  .catch(error => {  console.log(error);  })    ;  }, [forumId]);  **[The useEffect hook is used to fetch data from the server using Axios, a popular HTTP client library. It sends a GET request to the server at http://localhost:3001/getforum/${id} to fetch the data of a specific forum using its ID. The forumTitle and forumText state variables are then updated with the response data from the server using the setForumTitle and setForumText functions.]**    return (  <Box sx={{ p: 2 }}>  <Typography variant="h4" sx={{ mb: 2 }}>  {forumTitle}  </Typography>    <Typography variant="body1" sx={{ mb: 2 }}>  {forumText}  </Typography>    </Box>  **[ This is a very basic bax component using mui to display the forum title and the forum text.]**  );  }    export default Forum; |

### Environment assessment

|  |
| --- |
| import React, { useState } from 'react';  import { Button, FormControlLabel, FormGroup, FormLabel, Radio, RadioGroup, Typography, Box } from '@mui/material';    const HomeEnvironmentalRiskAssessment = () => {  const [questions, setQuestions] = useState([  { id: 1, question: 'Do you have a fire extinguisher in your home?', points: 5, answer: '' },  { id: 2, question: 'Do you have working smoke detectors?', points: 5, answer: '' },  { id: 3, question: 'Are all electrical cords and appliances in good condition?', points: 2, answer: '' },  { id: 4, question: 'Is your home free of clutter and excess debris?', points: 2, answer: '' },  { id: 5, question: 'Are all chemicals and cleaning products stored safely?', points: 2, answer: '' },  ]);  **[The component uses the useState hook to maintain the state of an array of questions, where each question object has an ID, a question, points associated with the question, and an answer (which is empty by default). It also maintains the state of the score, which starts at 0.]**    const [score, setScore] = useState(0);    const handleAnswerChange = (event, questionId) => {  const updatedQuestions = questions.map((question) =>  question.id === questionId ? { ...question, answer: event.target.value } : question  );  setQuestions(updatedQuestions);  };  **[The handleAnswerChange function is called when a user selects an answer for a question. It maps over the questions array, updates the answer for the question that matches the ID passed in, and returns a new array of updated questions.]**    const handleSubmit = () => {  const totalScore = questions.reduce((accumulator, question) => {  const answerValue = question.answer === 'no' ? question.points : 0;  return accumulator + answerValue;    }, 0);    setScore(totalScore);  console.log(score)  };  **[The handleSubmit function is called when the user clicks the Submit button. It calculates the total score by iterating over the questions array and adding the points associated with each question that was answered with "no". It then sets the score state to the total score, and logs the score to the console.]**    const shouldShowProfessionalRecommendation = score >= 10;  **[The shouldShowProfessionalRecommendation variable is set to true when the score is greater than or equal to 10.]**    return (  **[The component renders a form that displays each question with Yes/No radio button options. The onChange event for each radio button updates the state of the corresponding question's answer. When the Submit button is clicked, the handleSubmit function is called, which calculates the total score and updates the score state. If the score is greater than or equal to 10, the component displays a message recommending a professional conduct a home environmental risk assessment.]**  <>  <Box p={5}>  <Typography variant="h4" sx={{ mb: 3 }}>  Home Environmental Risk Assessment  </Typography>  <FormGroup sx={{ mb: 3 }}>  <FormLabel component="legend">Please answer the following questions:</FormLabel>  {questions.map((question) => (  <FormGroup key={question.id}>  <Typography variant="body1">{question.question}</Typography>  <RadioGroup  row  aria-label={`Question ${question.id}`}  name={`question-${question.id}`}  value={question.answer}  onChange={(event) => handleAnswerChange(event, question.id)}  >  <FormControlLabel value="yes" control={<Radio />} label="Yes" />  <FormControlLabel value="no" control={<Radio />} label="No" />  </RadioGroup>  </FormGroup>  ))}  </FormGroup>  <Button variant="contained" onClick={handleSubmit}>  Submit  </Button>  {shouldShowProfessionalRecommendation && (  <Typography variant="body1" sx={{ mt: 3 }}>  Your score is {score}. We recommend that you have a professional conduct a home environmental risk assessment.  </Typography>  )}    </Box>  </>  );  };    export default HomeEnvironmentalRiskAssessment; |

Login

This login code was taken and tweaked from my old bootstrap login I made quite a while ago, while I changed it up to use formik, that is why this is in divs.

|  |
| --- |
| import { useFormik } from "formik";  import UserContext from "../../usercontext";  import { basicSchema, loginSchema } from "../../schemas";  import Axios from "axios";  import Cookies from 'js-cookie';  import { reach } from "yup";  import React, { useContext } from "react";  const bcrypt = require('bcryptjs');    // add Access-Control-Allow-Origin header to the request    const Login = () => {    const onSubmit = async (values, actions) => {  **[When the form is submitted, the onSubmit function is called. This function sends an HTTP POST request to the backend server at "http://localhost:3001/login" with the email and password entered by the user. If the email and password match a user in the database, the server will respond with a token and the user's information. This information is then stored in sessionStorage.]**    console.log(values);  console.log(actions);  try {  // add Access-Control-Allow-Origin header to the request    Axios.post("http://localhost:3001/login", {  email: values.email,  password: values.password,  }).then((response) => {  if (response.data.message == "Wrong email/password combination!") {  document.getElementById("dberror").innerHTML = "Wrong email/password combination!";  } else {    const token = response.data.token\_name  console.log(token)  sessionStorage.setItem("jwt", token)    const first\_name = response.data.firstName  const last\_name = response.data.lastName  const email = response.data.email  const profile\_pic = response.data.userPic  const authorization\_level = response.data.authorizationLevel  const id = response.data.id    const userData = {  user\_id: id,  first\_name: first\_name,  last\_name: last\_name,  email: email,  profile\_pic: profile\_pic,  authorization\_level: authorization\_level  }    sessionStorage.setItem("UserData", JSON.stringify(userData))    window.location.href = "/Dashboard"    }  });  } catch (error) {  console.log(error);  }  await new Promise((resolve) => setTimeout(resolve, 1000));  actions.resetForm();  };    const {  values,  errors,  touched,  isSubmitting,  handleBlur,  handleChange,  handleSubmit,  } = useFormik({  initialValues: {  email: "",  password: "",  },  validationSchema: loginSchema,  **[ the login uses a validation schema called “loginSchema”]**  onSubmit,    });    return (  **[the form is rendered with input fields for email and password. When the user clicks the "Submit" button, the onSubmit function is called. The component also includes links for resetting a forgotten password and creating a new account.]**    <div className="LoginForm">  <div className="GlassForm">    <form onSubmit={handleSubmit}>  [  <h2>Login</h2>  <div className="form-group">  <p className="plabel" htmlFor="email">Email</p>  <input  type="text"  id="email"  values={values.email}  onChange={handleChange}  onBlur={handleBlur}  className= {errors.email && touched.email ? "input-error" : ""}  placeholder="Enter email" />  {errors.email && touched.email && <p className="error">{errors.email}</p>}    </div>  <div className="form-group">  <p className="plabel" htmlFor="password">Password</p>  <input  id="password"  type="password"  placeholder="Enter your password"  value={values.password}  onChange={handleChange}  onBlur={handleBlur}  className={errors.password && touched.password ? "input-error" : ""}  />  {errors.password && touched.password && (  <p className="error">{errors.password}</p>  )}  </div>  <div className="bottomForm">  <p className="error" id="dberror"></p>  <button className=""disabled={isSubmitting} type="submit">  Submit  </button>  <div className="centered">  <small><a href="#">Forgot your password?</a></small>  </div>  <div className="centered">  <small><a href="/register">Don't have an account? Sign up</a></small>  </div>  </div>      </form>    </div>  </div>  );  }    export default Login; |

### Register

Register is kind of the same thing as the login in the frontend. This was taken from my old bootstrap and tweaked.

|  |
| --- |
| import { useFormik } from "formik";  import { registerSchema, loginSchema } from "../../schemas";  import Axios from "axios";  const bcrypt = require('bcryptjs');  // add Access-Control-Allow-Origin header to the request  **[This code defines a React component named Register that handles user registration. It imports useFormik hook from the formik library, as well as registerSchema and loginSchema from the ../../schemas file, which are used for form validation.]**  const onSubmit = async (values, actions) => {  **[The onSubmit function is called when the user submits the form, it logs the form values and actions to the console, generates a salt using bcrypt.genSaltSync() and hashes the user's password using bcrypt.hashSync(). It then sends a POST request using Axios to the server to register the user, passing in the user's form values and hashed password as data.]**  console.log(values);  console.log(actions);  const salt = bcrypt.genSaltSync(10);  const hash = bcrypt.hashSync(values.password, salt);  try {  // add Access-Control-Allow-Origin header to the request    Axios.post("http://localhost:3001/register", {  firstName: values.firstName,  lastName: values.lastName,  email: values.email,  authorizationLevel: "Member",  password: hash,  }).then((response) => {  console.log(response);  if (response.data.message == "Wrong email/password combination!") {  document.getElementById("dberror").innerHTML = "Wrong email/password combination!";  }    });  } catch (error) {  console.log(error);  }  await new Promise((resolve) => setTimeout(resolve, 1000));  actions.resetForm();  };    const Register = () => { **[The Register component uses the useFormik hook to set up the form with its initial values and validation schema, and handles form inputs and submits through the handleChange and handleSubmit functions. It renders a form with fields for the user's first name, last name, email, password, and confirm password, along with error messages for invalid input and a submit button. If the form is submitted successfully, it also calls actions.resetForm() to clear the form.]**    const {  values,  errors,  touched,  isSubmitting,  handleBlur,  handleChange,  handleSubmit,  } = useFormik({  initialValues: {  firstName: "",  lastName: "",  email: "",  password: "",  confirmPassword: "",  },  validationSchema: registerSchema,  onSubmit,    });    return (    <div className="LoginForm">  <div className="GlassForm">    <form onSubmit={handleSubmit}>  <h2>Register</h2>  <div className="name-group form-group">    <div className="name-module" style={{marginRight:"4px"}}>  <p className="plabel" htmlFor="firstName">First Name</p>  <input  type="text"  id="firstName"  values={values.firstName}  onChange={handleChange}  onBlur={handleBlur}  className= {errors.firstName && touched.firstName ? "input-error" : ""}  placeholder="Enter first name" />  {errors.firstName && touched.firstName && <p className="error">{errors.firstName}</p>}  </div>    <div className="name-module">  <p className="plabel" htmlFor="lastName">Last Name</p>  <input  type="text"  id="lastName"  values={values.lastName}  onChange={handleChange}  onBlur={handleBlur}  className= {errors.lastName && touched.lastName ? "input-error" : ""}  placeholder="Enter first name" />  {errors.lastName && touched.lastName && <p className="error">{errors.lastName}</p>}  </div>  </div>    <div className="form-group">  <p className="plabel" htmlFor="email">Email</p>  <input  type="text"  id="email"  values={values.email}  onChange={handleChange}  onBlur={handleBlur}  className= {errors.email && touched.email ? "input-error" : ""}  placeholder="Enter email" />  {errors.email && touched.email && <p className="error">{errors.email}</p>}  </div>    <div className="form-group">  <p className="plabel" htmlFor="password">Password</p>  <input  id="password"  type="password"  placeholder="Enter your password"  value={values.password}  onChange={handleChange}  onBlur={handleBlur}  className={errors.password && touched.password ? "input-error" : ""}  />  {errors.password && touched.password && (  <p className="error">{errors.password}</p>  )}  </div>      <div className="form-group">  <p className="plabel" htmlFor="confirmPassword">Confirm Password</p>  <input  id="confirmPassword"  type="password"  placeholder="Enter your password"  value={values.confirmPassword}  onChange={handleChange}  onBlur={handleBlur}  className={errors.confirmPassword && touched.confirmPassword ? "input-error" : ""}  />  {errors.confirmPassword && touched.confirmPassword && (  <p className="error">{errors.confirmPassword}</p>  )}  </div>  <div className="bottomForm">  <p className="error" id="dberror"></p>  <button className=""disabled={isSubmitting} type="submit">  Submit  </button>  <div className="centered">  <small><a href="/login">Already have an account? Log in</a></small>  </div>  </div>      </form>    </div>  </div>  );  }    export default Register; |

## Backend

### Test connection

|  |
| --- |
| const db = mysql.createConnection({  user: "root",  host: "localhost",  password: "",  database: "healthadvicegroup"  });    // make test connection  db.connect((err) => {  if (err) {  console.log(err);  } else {  console.log("MySQL connected");  }  });  **[this is mainly included for developers, to see if the database is connected and working properly]** |

## Login

|  |
| --- |
| app.post('/login', (req, res) => {  **[When the '/login' endpoint receives a POST request, it extracts the email and password from the request body. Then it queries the MySQL database to find a user with a matching email address. If the email is found, the code uses bcrypt to compare the entered password with the hashed password stored in the database. If the passwords match, the code generates a JSON Web Token (JWT) and sends the user details along with the token as a response. Otherwise, if the passwords don't match, it sends a 401 Unauthorized response. If the email is not found in the database, the code sends a 404 Not Found response.**  **The code also logs any errors or responses for debugging purposes.**  **]**  const email = req.body.email;  const password = req.body.password;    db.query(  'SELECT \* FROM users WHERE email = ?',  [email],  (err, result) => {  if (err) {  res.status(500).json({ message: 'Internal Server Error' });  console.log(err);  return;  }  if (result.length > 0) {  // Compare the entered password with the hashed password from the database  const isPasswordMatch = bcrypt.compareSync(password, result[0].password);  if (isPasswordMatch) {  // Generate JWT  const token = jwt.sign({ userId: result[0].id }, 'your\_secret\_key\_here', { expiresIn: '1h' });    // Send user details as response  res.json({  id: result[0].id,  firstName: result[0].first\_name,  lastName: result[0].last\_name,  email: result[0].email,  userPic: result[0].profile\_picture,  authorizationLevel: result[0].authorization\_level,  token\_name: token  });    console.log(result);  } else {  res.status(401).json({ message: 'Wrong email/password combination!' });  console.log('Wrong email/password combination!');  }  } else {  res.status(404).json({ message: 'User not found!' });  console.log('User not found!');  }  }  );  }); |

## Register

|  |
| --- |
| app.post("/register", (req, res) => {  const firstName = req.body.firstName;  const lastName = req.body.lastName;  const email = req.body.email;  const password = req.body.password;  const authorizationLevel = req.body.authorizationLevel    db.query(  "INSERT INTO users (first\_name, last\_name, email, password,authorization\_level) VALUES (?,?,?,?,?)",  [firstName, lastName, email, password,authorizationLevel],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "User registered!" });  console.log("User registered!")  }  }  )  });  **[When a POST request is sent to the endpoint "/register", the code extracts the user's first name, last name, email, password, and authorization level from the request body. It then inserts these values into the "users" table of the database by using the query() method of the db object. If the insertion is successful, the server responds with a JSON object containing the message "User registered!". If an error occurs, the server responds with a JSON object containing the error message. Finally, the console logs either the error message or the message "User registered!" depending on whether or not an error occurred during the registration process.]** |

### Edit user

|  |
| --- |
| app.post("/edituser", (req, res) => {  const firstName = req.body.firstName;  const lastName = req.body.lastName;  const email = req.body.email;  const password = req.body.password;  const authorizationLevel = req.body.authorizationLevel  const id = req.body.userId    // get current user info from db  // compare current user info with new user info  // if current user info is different from new user info, update the db    db.query(  "SELECT \* FROM users WHERE id = ?",  [id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  if (result[0].first\_name !== firstName && firstName !== "") {  db.query(  "UPDATE users SET first\_name = ? WHERE id = ?",  [firstName, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].last\_name !== lastName && lastName !== "") {  db.query(  "UPDATE users SET last\_name = ? WHERE id = ?",  [lastName, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].email !== email && email !== "") {  db.query(  "UPDATE users SET email = ? WHERE id = ?",  [email, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].password !== password && password !== "") {  db.query(  "UPDATE users SET password = ? WHERE id = ?",  [password, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }    // send back the new user info  db.query(  "SELECT \* FROM users WHERE id = ?",  [id],  (err, result) => {    if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "User edited!", user: result[0] });  console.log(result[0])  console.log("User edited!")  }  }  )    }    }  )        });    app.post("/edituser", (req, res) => {  const firstName = req.body.firstName;  const lastName = req.body.lastName;  const email = req.body.email;  const password = req.body.password;  const authorizationLevel = req.body.authorizationLevel  const id = req.body.userId    // get current user info from db  // compare current user info with new user info  // if current user info is different from new user info, update the db    db.query(  "SELECT \* FROM users WHERE id = ?",  [id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  if (result[0].first\_name !== firstName && firstName !== "") {  db.query(  "UPDATE users SET first\_name = ? WHERE id = ?",  [firstName, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].last\_name !== lastName && lastName !== "") {  db.query(  "UPDATE users SET last\_name = ? WHERE id = ?",  [lastName, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].email !== email && email !== "") {  db.query(  "UPDATE users SET email = ? WHERE id = ?",  [email, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }  if (result[0].password !== password && password !== "") {  db.query(  "UPDATE users SET password = ? WHERE id = ?",  [password, id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  console.log("User edited!")  }  }  )  }    // send back the new user info  db.query(  "SELECT \* FROM users WHERE id = ?",  [id],  (err, result) => {    if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "User edited!", user: result[0] });  console.log(result[0])  console.log("User edited!")  }  }  )  }  }  )  });    **[This code block defines an API endpoint that handles user edits. When a POST request is made to this endpoint, the server retrieves the user's current information from the database and compares it to the new information provided in the request body. If any of the fields are different, the server updates the corresponding fields in the database. After the updates are made, the server sends back the new user information in the response.**    **The endpoint expects the following parameters in the request body: firstName, lastName, email, password, authorizationLevel, and userId. It then uses these parameters to update the user's information in the users table of the database.**    **Note that this code only updates the user's information if there is a difference between the current information and the new information provided in the request body. If a field in the request body is blank, it is not updated in the database.]** |

## Diary

#### Get diary posts by id

|  |
| --- |
| app.get("/getdiaryposts/:id", (req, res) => {  const id = req.params.id  db.query(  "SELECT \* FROM health\_diary WHERE user\_id = ?",  [id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "Diary posts retrieved!", posts: result });  console.log("Diary posts retrieved!")  }  }  )  });  **[This is an endpoint in an Express.js app that listens for a GET request at the URL path "/getdiaryposts/:id". When a GET request is received, it extracts the value of the "id" parameter from the request URL using req.params.id, which represents the user ID of the person whose diary posts are being requested.**    **The endpoint then performs a SELECT query on a database table called "health\_diary", filtering by the user ID to retrieve all diary posts for that user. The results of the query are passed to a callback function that handles the response to the client.**    **If the query encounters an error, the endpoint sends an error response to the client with the error message and logs the error to the console. If the query is successful, the endpoint sends a success response to the client with the message "Diary posts retrieved!" and the retrieved diary posts data, and logs a success message to the console.]** |

#### Diary post

|  |
| --- |
| app.post("/diarypost", (req, res) => {  **[This code creates an endpoint /diarypost that listens for a POST request. When a request is received, it extracts the userId, type, and message data from the request body. It then executes a SQL query that inserts these data into the health\_diary table. If there is an error, it sends a response containing the error message. If the query is successful, it sends a response with the message "Diary post created!" and logs "Diary post created!" to the console.]**  const userId = req.body.userId;  const type = req.body.type;  const message = req.body.message;    db.query(  "INSERT INTO health\_diary (user\_id, type, message) VALUES (?,?,?)",  [userId, type, message],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "Diary post created!" });  console.log("Diary post created!")  }  }  )  }); |

### Health

#### Get health posts

|  |
| --- |
| app.post("/healthpost", (req,res) => {  const userId = req.body.userId  const height = req.body.height  const weight = req.body.weight  const bmi = req.body.weight  const blood\_pressure = req.body.blood\_pressure  const heart\_rate = req.body.heart\_rate    db.query(  "INSERT INTO health\_tracking (user\_id, height, weight, bmi, blood\_pressure, heart\_rate) VALUES (?,?,?,?,?,?)",  [userId,height,weight,bmi,blood\_pressure,heart\_rate],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "health post created!" });  console.log("health post created!")  }  }  )  });  **[This code creates a route for a POST request to the "/healthpost" endpoint. When a request is sent to this endpoint, the server extracts the values of the fields "userId", "height", "weight", "bmi", "blood\_pressure", and "heart\_rate" from the request body. It then uses these values to insert a new record into the "health\_tracking" table of a database using a SQL INSERT statement. If the insertion is successful, the server sends a response with the message "health post created!" and logs this message to the console. If there is an error during the insertion process, the server sends a response with an error object and logs the error to the console.]** |

#### Get health info by id

|  |
| --- |
| app.get("/gethealthinfo/:id", (req, res) => {  const id = req.params.id  db.query(  "SELECT \* FROM health\_tracking WHERE user\_id = ?",  [id],  (err,result) => {  if (err) {  res.send({ err: err });  console.log(err)  } else {  res.send({ message: "health info retrieved!", posts: result})  console.log("Health info retrieved!")  }  }  )  })  **[This code extracts the user ID from the URL parameter using req.params.id. It then queries the database to retrieve all rows from the health\_tracking table where the user\_id matches the extracted ID.**  **If there are any errors during the query, it sends a response with an error message and logs the error to the console. Otherwise, it sends a response with a success message and the retrieved health information as an array of objects in the result property, and logs a success message to the console.]** |

### Forums

#### Get all forums

|  |
| --- |
| app.get("/getforums", (req, res) => {  db.query(  "SELECT \* FROM forums",  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "Forums retrieved!", forums: result });  console.log("Forums retrieved!")  }  }  )  });  **[ This code sets up a GET endpoint at "/getforums". When this endpoint is accessed, it sends a SELECT query to the database to retrieve all the rows from the "forums" table. If the query is successful, it sends a response to the client with a message "Forums retrieved!" and the retrieved data as an object with a key "forums". If there is an error in the query, it sends an error message to the client and logs the error in the console.]** |

#### Get forums by id

|  |
| --- |
| app.get("/getforum/:id", (req, res) => {  const id = req.params.id  console.log(id)  db.query(  "SELECT \* FROM forums\_posts WHERE forum\_id = ?",  [id],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  // get forum from the id, then send back the forum and the posts  db.query(  "SELECT \* FROM forums WHERE id = ?",  [id],  (err, result2) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "Forum retrieved!", forum: result2[0], posts: result });  console.log("Forum retrieved!")  }  }  )  }  }  )  });  **[This code defines a route for getting a specific forum by its ID. It expects the ID to be passed in as a parameter in the URL. The code first extracts the ID from the request parameters and then queries the database to retrieve all posts associated with that forum using the ID as a filter.**  **If the query to retrieve the posts is successful, the code then queries the database again to retrieve the forum details using the same ID. If that query is also successful, the code sends a response containing a message indicating that the forum was retrieved successfully, the forum object, and an array of post objects associated with that forum.**  **If there is an error with either query, the code sends an error response containing the error object and logs the error to the console.**  **]** |

#### Search posts

|  |
| --- |
| app.get("/searchposts/:search", (req, res) => {  const search = req.params.search  // search title or tags  db.query(  "SELECT \* FROM forums WHERE title LIKE ? OR tags LIKE ?",  ['%' + search + '%', '%' + search + '%'],  (err, result) => {  if (err) {  res.send({ err: err });  console.log(err);  } else {  res.send({ message: "Posts retrieved!", posts: result });  console.log("Posts retrieved!")  }  }  )  });  **[This code sets up an API endpoint that listens for a GET request at the URL path "/searchposts/:search". The endpoint expects a parameter "search" in the URL path, which will be used to search for forum posts that match either the post title or the post tags.**  **When the endpoint is accessed, the code executes a SQL query against a database to select all forum posts whose title or tags contain the search string. The query uses the LIKE operator to perform a partial match on the search string with wildcard characters (%) added before and after it.**  **If the query is successful, the code sends a response with a message "Posts retrieved!" and an array of post objects that match the search criteria. If there is an error, the code sends a response with an error message and logs the error to the console.**  **]** |