Thomas Hoerger

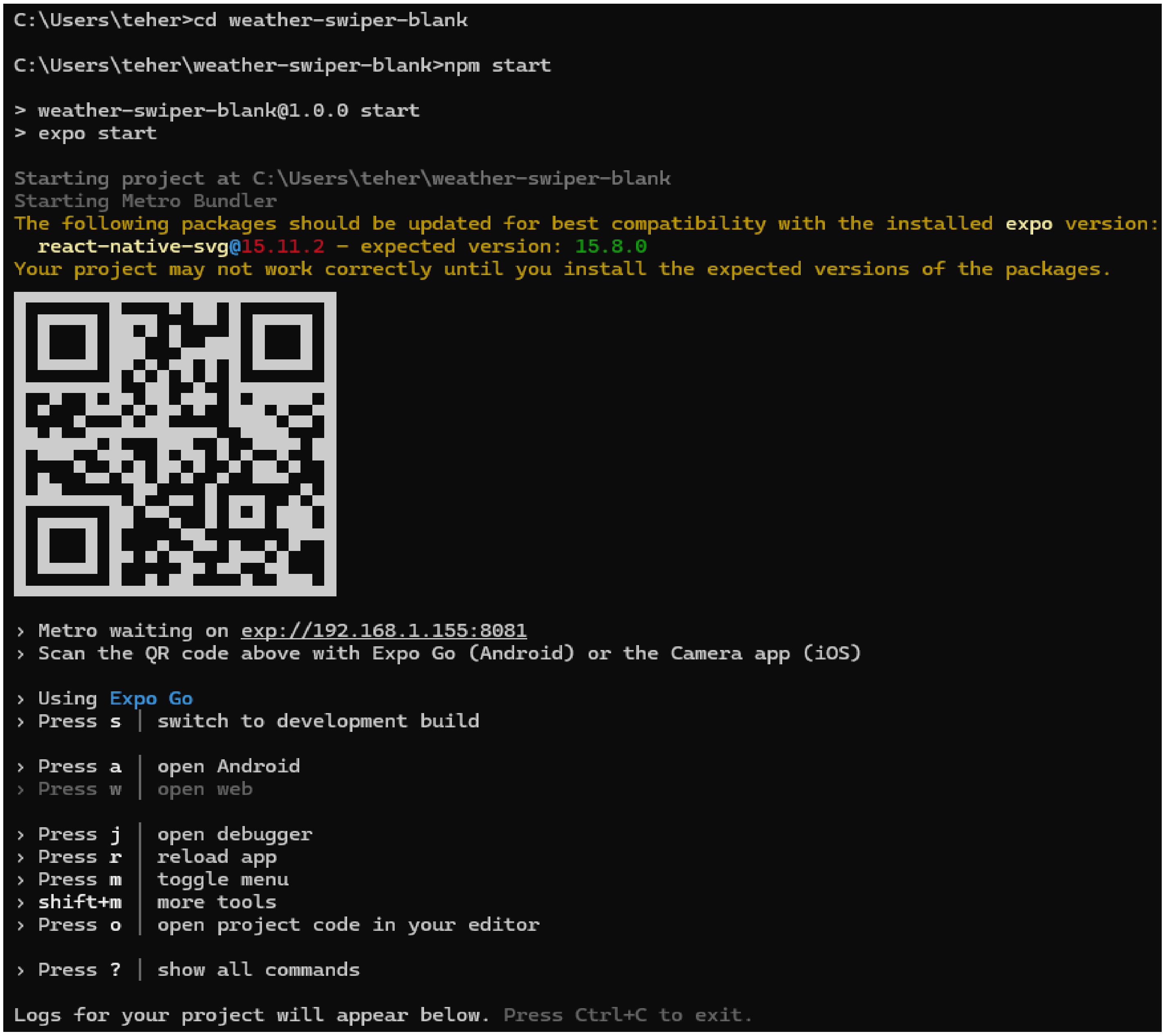
Final Project

4/18/2025

Step-by-Step Implementation

1. VPS Setup and MySQL Database
   - Deployed Ubuntu server on Google Cloud VPS.
   - Installed Apache, PHP, and MySQL.

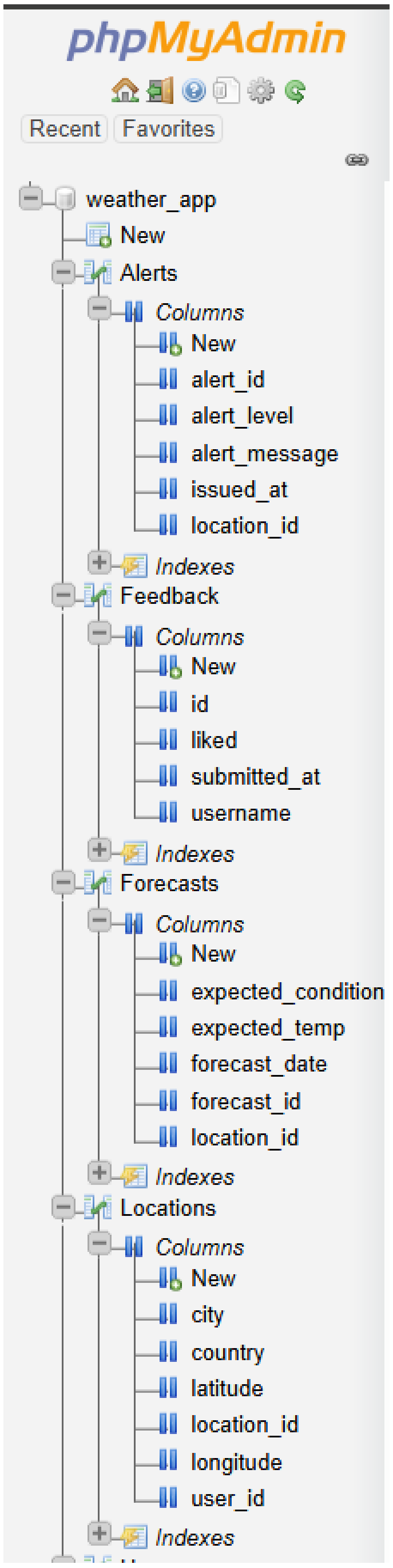
2. Created Expo project with npx create-expo-app weather-swiper-blank after installing it on the local system:



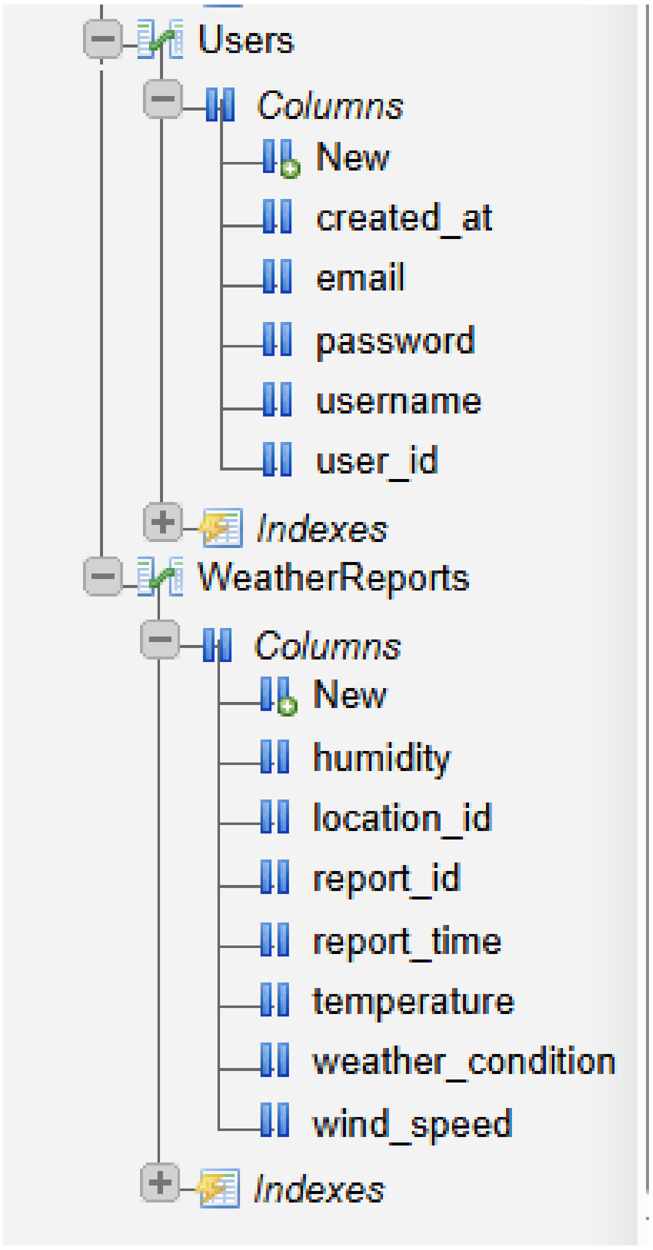
```
C:\Users\teher>cd weather-swiper-blank

C:\Users\teher\weather-swiper-blank>npm start

> weather-swiper-blank@1.0.0 start
> expo start

Starting project at C:\Users\teher\weather-swiper-blank
Starting Metro Bundler
The following packages should be updated for best compatibility with the installed expo version:
  react-native-svg@15.11.2 - expected version: 15.8.0
Your project may not work correctly until you install the expected versions of the packages.
```

```
› Metro waiting on exp://192.168.1.155:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Using Expo Go
› Press s │ switch to development build

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu
› shift+m │ more tools
› Press o │ open project code in your editor

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
```

3. Created a MySQL database weather_app with 6 tables: Users, Locations, WeatherReports, Forecasts, Alerts, and Feedback.



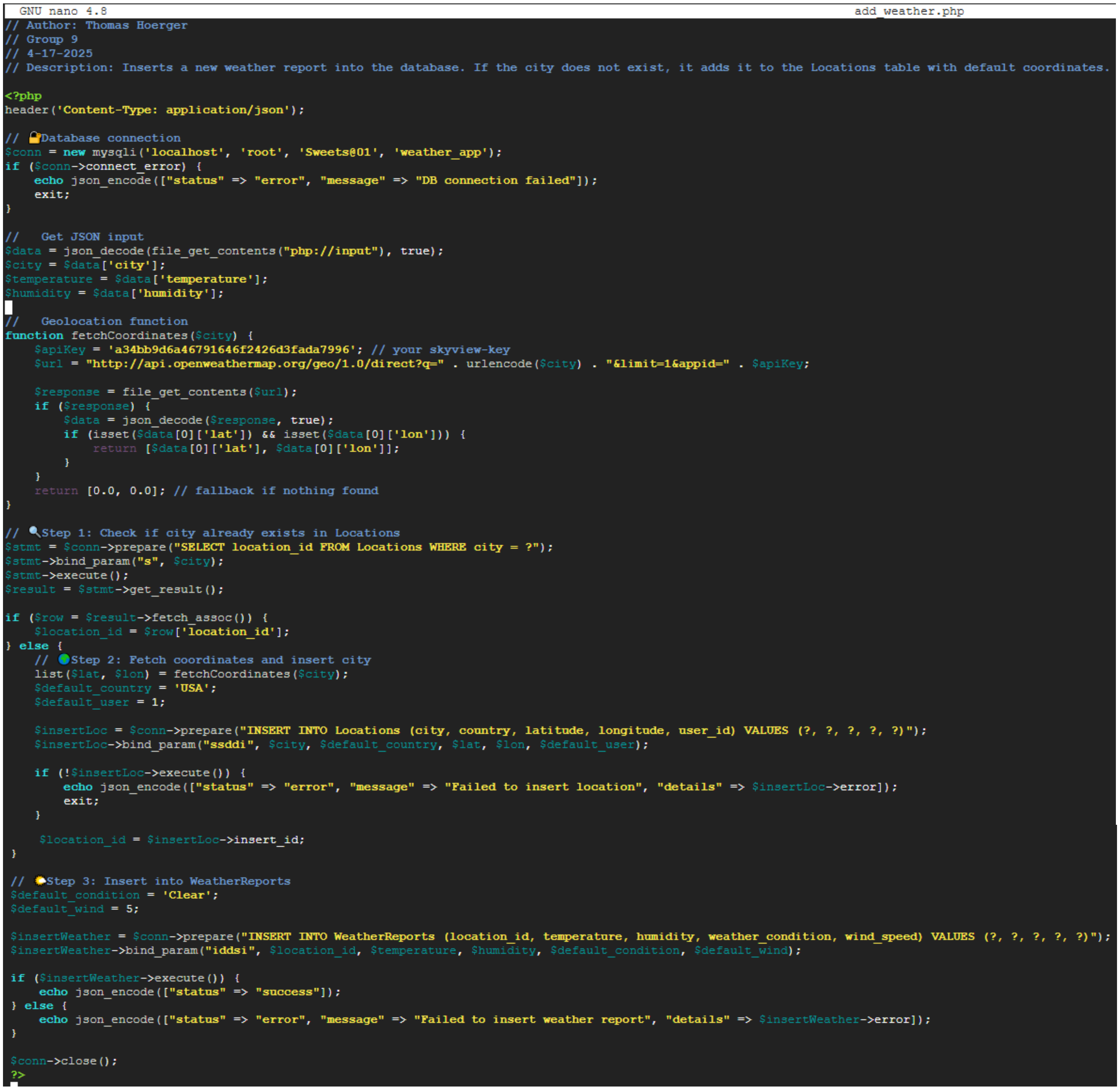
```
weather_app
  New
  Alerts
    Columns
      New
      alert_id
      alert_level
      alert_message
      issued_at
      location_id
    Indexes
  Feedback
    Columns
      New
      id
      liked
      submitted_at
      username
    Indexes
  Forecasts
    Columns
      New
      expected_condition
      expected_temp
      forecast_date
      forecast_id
      location_id
    Indexes
  Locations
    Columns
      New
      city
      country
      latitude
      location_id
      longitude
      user_id
    Indexes
```



```
Users
  Columns
    New
    created_at
    email
    password
    username
    user_id
  Indexes
WeatherReports
  Columns
    New
    humidity
    location_id
    report_id
    report_time
    temperature
    weather_condition
    wind_speed
  Indexes
```

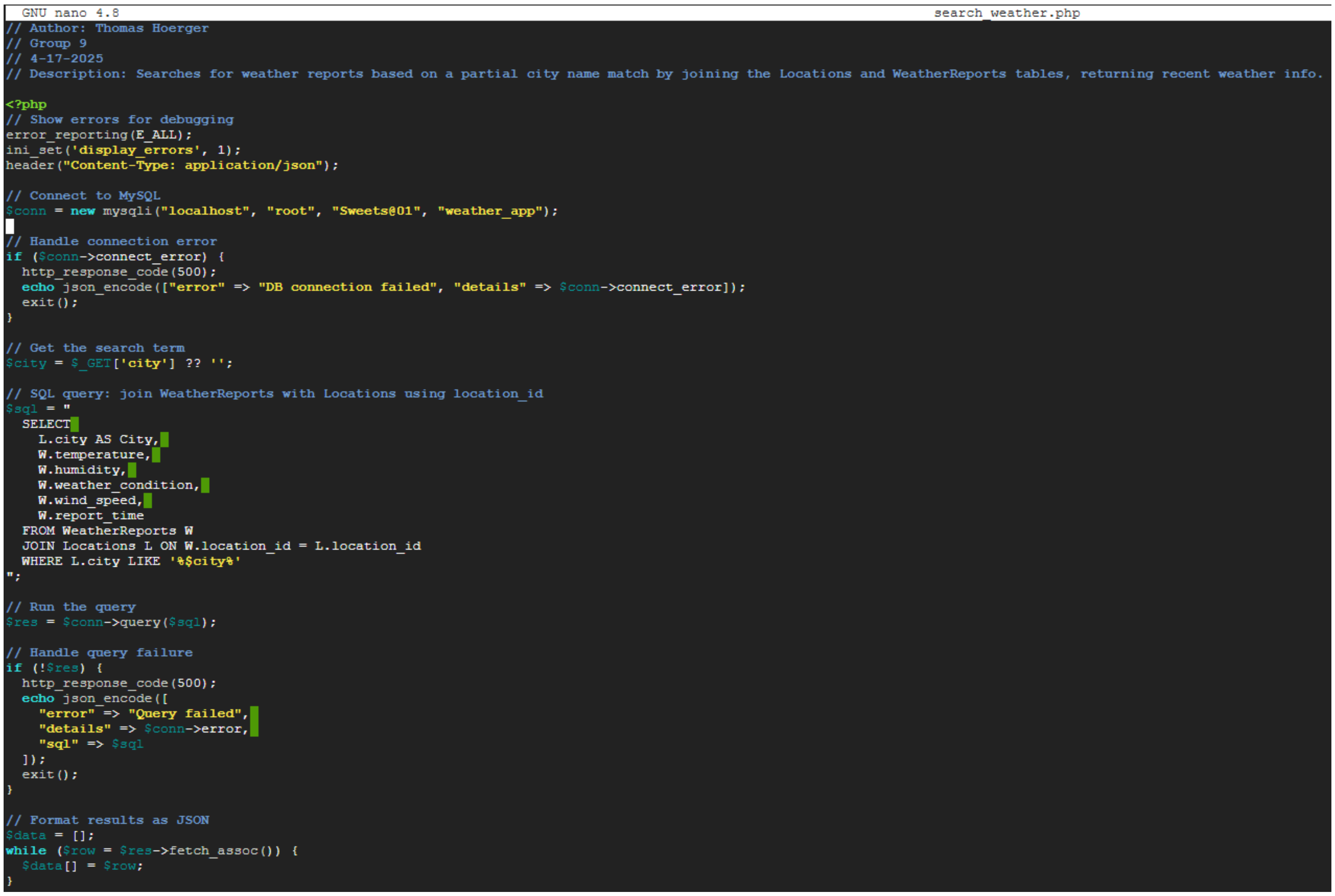
4. Created and enabled remote access to the following PHP scripts on VPS:

add_weather.php: Adds data to the database.



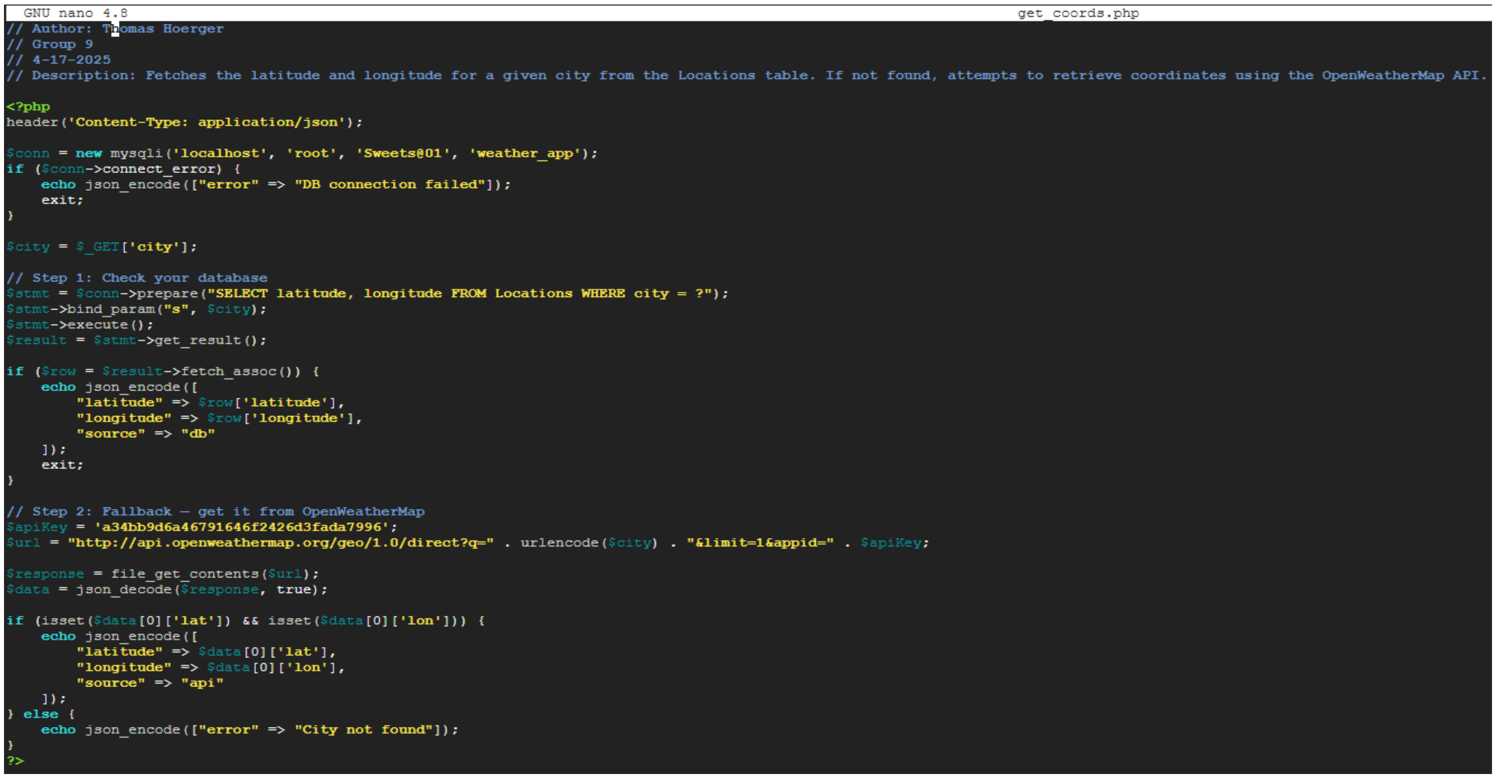
```
GNU nano 4.8                                                                                    add_weather.php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Inserts a new weather report into the database. If the city does not exist, it adds it to the Locations table with default coordinates.

<?php
header('Content-Type: application/json');

// 🔐Database connection
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');
if ($conn->connect_error) {
    echo json_encode(["status" => "error", "message" => "DB connection failed"]);
    exit;
}

//   Get JSON input
$data = json_decode(file_get_contents("php://input"), true);
$city = $data['city'];
$temperature = $data['temperature'];
$humidity = $data['humidity'];

//   Geolocation function
function fetchCoordinates($city) {
    $apiKey = 'a34bb9d6a46791646f2426d3fada7996'; // your skyview-key
    $url = "http://api.openweathermap.org/geo/1.0/direct?q=" . urlencode($city) . "&limit=1&appid=" . $apiKey;

    $response = file_get_contents($url);
    if ($response) {
        $data = json_decode($response, true);
        if (isset($data[0]['lat']) && isset($data[0]['lon'])) {
            return [$data[0]['lat'], $data[0]['lon']];
        }
    }
    return [0.0, 0.0]; // fallback if nothing found
}

// 🔍Step 1: Check if city already exists in Locations
$stmt = $conn->prepare("SELECT location_id FROM Locations WHERE city = ?");
$stmt->bind_param("s", $city);
$stmt->execute();
$result = $stmt->get_result();

if ($row = $result->fetch_assoc()) {
    $location_id = $row['location_id'];
} else {
    // 🌍Step 2: Fetch coordinates and insert city
    list($lat, $lon) = fetchCoordinates($city);
    $default_country = 'USA';
    $default_user = 1;

    $insertLoc = $conn->prepare("INSERT INTO Locations (city, country, latitude, longitude, user_id) VALUES (?, ?, ?, ?, ?)");
    $insertLoc->bind_param("ssddi", $city, $default_country, $lat, $lon, $default_user);

    if (!$insertLoc->execute()) {
        echo json_encode(["status" => "error", "message" => "Failed to insert location", "details" => $insertLoc->error]);
        exit;
    }

    $location_id = $insertLoc->insert_id;
}

// ☀Step 3: Insert into WeatherReports
$default_condition = 'Clear';
$default_wind = 5;

$insertWeather = $conn->prepare("INSERT INTO WeatherReports (location_id, temperature, humidity, weather_condition, wind_speed) VALUES (?, ?, ?, ?, ?)");
$insertWeather->bind_param("iddsi", $location_id, $temperature, $humidity, $default_condition, $default_wind);

if ($insertWeather->execute()) {
    echo json_encode(["status" => "success"]);
} else {
    echo json_encode(["status" => "error", "message" => "Failed to insert weather report", "details" => $insertWeather->error]);
}

$conn->close();
?>
```

search_weather.php: Retrieves weather by city.



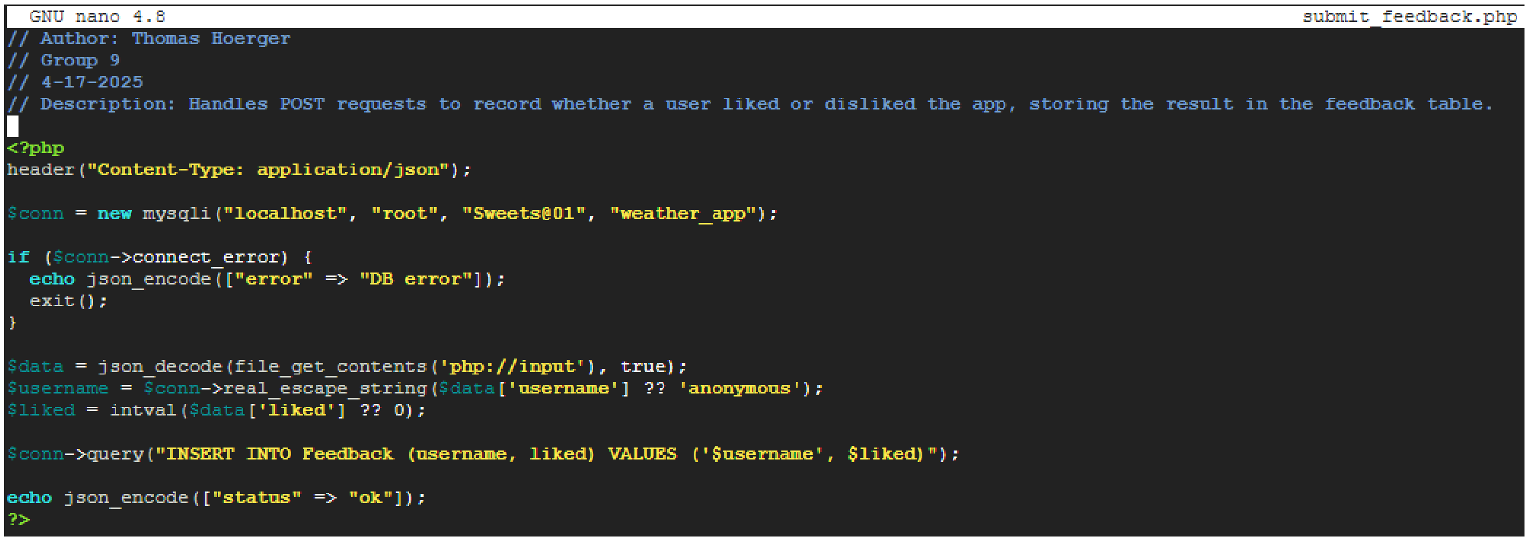
```
  GNU nano 4.8                                                                                    search_weather.php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Searches for weather reports based on a partial city name match by joining the Locations and WeatherReports tables, returning recent weather info.

<?php
// Show errors for debugging
error_reporting(E_ALL);
ini_set('display_errors', 1);
header("Content-Type: application/json");

// Connect to MySQL
$conn = new mysqli("localhost", "root", "Sweets@01", "weather_app");

// Handle connection error
if ($conn->connect_error) {
  http_response_code(500);
  echo json_encode(["error" => "DB connection failed", "details" => $conn->connect_error]);
  exit();
}

// Get the search term
$city = $_GET['city'] ?? '';

// SQL query: join WeatherReports with Locations using location_id
$sql = "
  SELECT
    L.city AS City,
    W.temperature,
    W.humidity,
    W.weather_condition,
    W.wind_speed,
    W.report_time
  FROM WeatherReports W
  JOIN Locations L ON W.location_id = L.location_id
  WHERE L.city LIKE '%$city%'
";

// Run the query
$res = $conn->query($sql);

// Handle query failure
if (!$res) {
  http_response_code(500);
  echo json_encode([
    "error" => "Query failed",
    "details" => $conn->error,
    "sql" => $sql
  ]);
  exit();
}

// Format results as JSON
$data = [];
while ($row = $res->fetch_assoc()) {
  $data[] = $row;
}
```

get_coords.php: Zooms map to city location (with fallback to OpenWeatherMap).



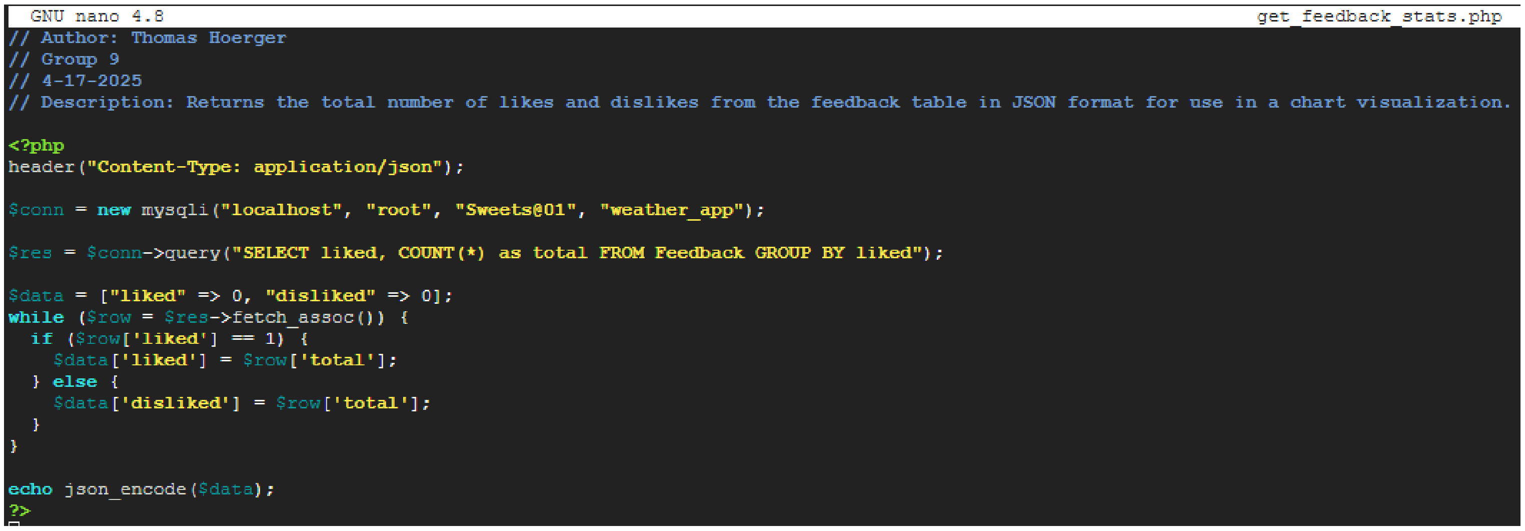
```
  GNU nano 4.8                                                                                    get_coords.php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Fetches the latitude and longitude for a given city from the Locations table. If not found, attempts to retrieve coordinates using the OpenWeatherMap API.

<?php
header('Content-Type: application/json');

$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]);
    exit;
}

$city = $_GET['city'];

// Step 1: Check your database
$stmt = $conn->prepare("SELECT latitude, longitude FROM Locations WHERE city = ?");
$stmt->bind_param("s", $city);
$stmt->execute();
$result = $stmt->get_result();

if ($row = $result->fetch_assoc()) {
    echo json_encode([
        "latitude" => $row['latitude'],
        "longitude" => $row['longitude'],
        "source" => "db"
    ]);
    exit;
}

// Step 2: Fallback — get it from OpenWeatherMap
$apiKey = 'a34bb9d6a46791646f2426d3fada7996';
$url = "http://api.openweathermap.org/geo/1.0/direct?q=" . urlencode($city) . "&limit=1&appid=" . $apiKey;

$response = file_get_contents($url);
$data = json_decode($response, true);

if (isset($data[0]['lat']) && isset($data[0]['lon'])) {
    echo json_encode([
        "latitude" => $data[0]['lat'],
        "longitude" => $data[0]['lon'],
        "source" => "api"
    ]);
} else {
    echo json_encode(["error" => "City not found"]);
}
?>
```

submit_feedback.php: Stores like/dislike values.

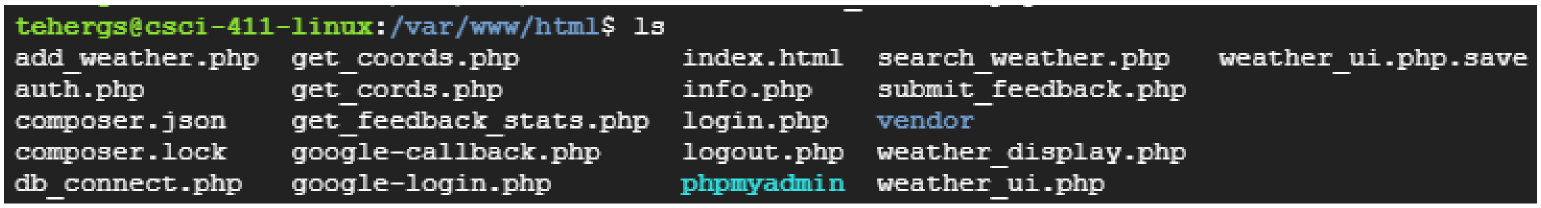


```
  GNU nano 4.8                                                                                   submit_feedback.php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Handles POST requests to record whether a user liked or disliked the app, storing the result in the feedback table.

<?php
header("Content-Type: application/json");

$conn = new mysqli("localhost", "root", "Sweets@01", "weather_app");

if ($conn->connect_error) {
  echo json_encode(["error" => "DB error"]);
  exit();
}

$data = json_decode(file_get_contents('php://input'), true);
$username = $conn->real_escape_string($data['username'] ?? 'anonymous');
$liked = intval($data['liked'] ?? 0);

$conn->query("INSERT INTO Feedback (username, liked) VALUES ('$username', $liked)");

echo json_encode(["status" => "ok"]);
?>
```

get_feedback_stats.php: Returns feedback data.



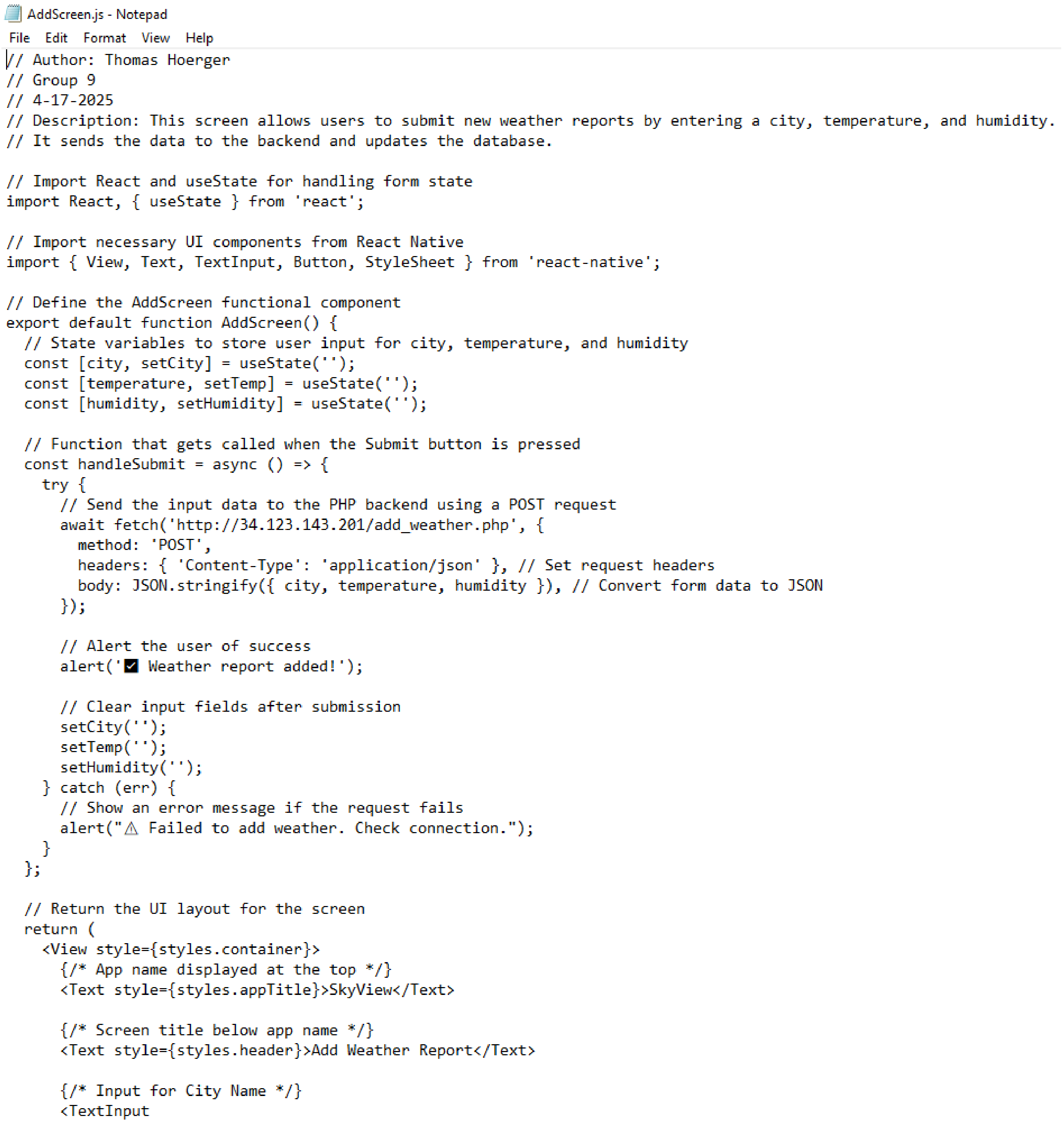
```
  GNU nano 4.8                                                                                get_feedback_stats.php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Returns the total number of likes and dislikes from the feedback table in JSON format for use in a chart visualization.

<?php
header("Content-Type: application/json");

$conn = new mysqli("localhost", "root", "Sweets@01", "weather_app");

$res = $conn->query("SELECT liked, COUNT(*) as total FROM Feedback GROUP BY liked");

$data = ["liked" => 0, "disliked" => 0];
while ($row = $res->fetch_assoc()) {
  if ($row['liked'] == 1) {
    $data['liked'] = $row['total'];
  } else {
    $data['disliked'] = $row['total'];
  }
}

echo json_encode($data);
?>
```



```
tehergs@csci-411-linux:/var/www/html$ ls
add_weather.php  get_coords.php          index.html   search_weather.php   weather_ui.php.save
auth.php         get_cords.php           info.php     submit_feedback.php
composer.json    get_feedback_stats.php  login.php    vendor
composer.lock    google-callback.php     logout.php   weather_display.php
db_connect.php   google-login.php        phpmyadmin   weather_ui.php
```

5. Implemented 5 screens in the Expo project on my local machine using JavaScript:

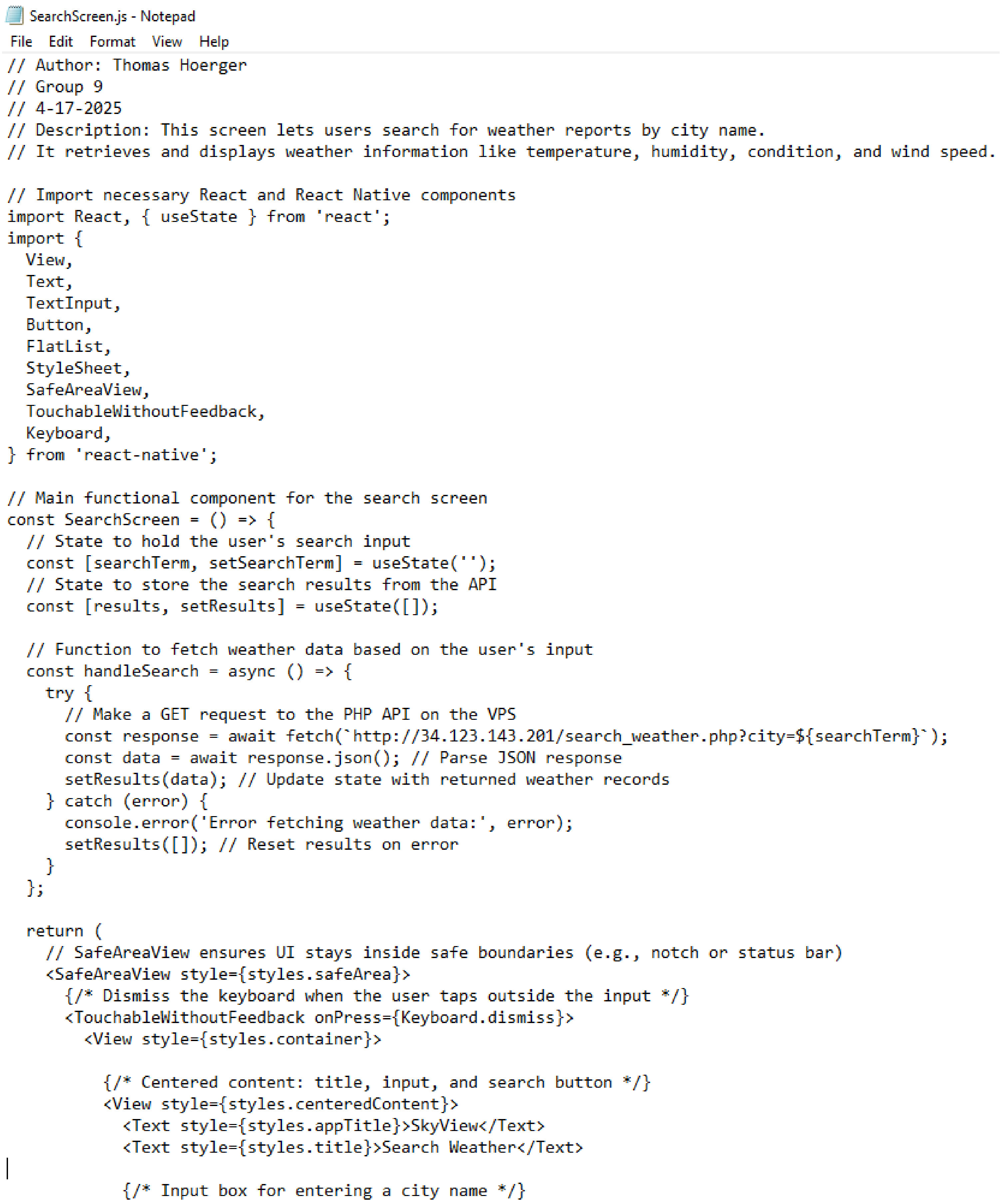
AddScreen.js: Form to submit weather.



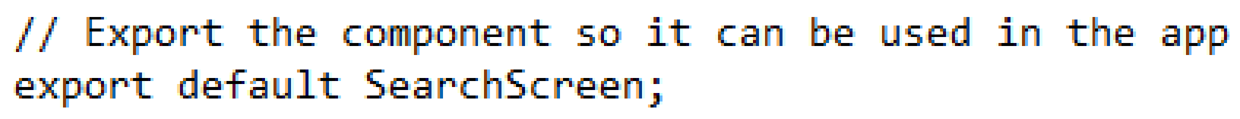
```
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: This screen allows users to submit new weather reports by entering a city, temperature, and humidity.
// It sends the data to the backend and updates the database.

// Import React and useState for handling form state
import React, { useState } from 'react';

// Import necessary UI components from React Native
import { View, Text, TextInput, Button, StyleSheet } from 'react-native';

// Define the AddScreen functional component
export default function AddScreen() {
  // State variables to store user input for city, temperature, and humidity
  const [city, setCity] = useState('');
  const [temperature, setTemp] = useState('');
  const [humidity, setHumidity] = useState('');

  // Function that gets called when the Submit button is pressed
  const handleSubmit = async () => {
    try {
      // Send the input data to the PHP backend using a POST request
      await fetch('http://34.123.143.201/add_weather.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' }, // Set request headers
        body: JSON.stringify({ city, temperature, humidity }), // Convert form data to JSON
      });

      // Alert the user of success
      alert('✅ Weather report added!');

      // Clear input fields after submission
      setCity('');
      setTemp('');
      setHumidity('');
    } catch (err) {
      // Show an error message if the request fails
      alert("⚠ Failed to add weather. Check connection.");
    }
  };

  // Return the UI layout for the screen
  return (
    <View style={styles.container}>
      {/* App name displayed at the top */}
      <Text style={styles.appTitle}>SkyView</Text>

      {/* Screen title below app name */}
      <Text style={styles.header}>Add Weather Report</Text>

      {/* Input for City Name */}
      <TextInput
```

```
          placeholder="City"
          value={city}
          onChangeText={setCity}
          style={styles.input}
        />

        {/* Input for Temperature */}
        <TextInput
          placeholder="Temperature (°F)"
          value={temperature}
          onChangeText={setTemp}
          keyboardType="numeric" // Brings up numeric keypad
          style={styles.input}
        />

        {/* Input for Humidity */}
        <TextInput
          placeholder="Humidity (%)"
          value={humidity}
          onChangeText={setHumidity}
          keyboardType="numeric"
          style={styles.input}
        />

        {/* Submit button that triggers handleSubmit */}
        <Button title="Submit" onPress={handleSubmit} />
      </View>
    );
}

// Styles for layout and UI design
const styles = StyleSheet.create({
  container: {
    flex: 1,                    // Fills the screen vertically
    width: '100%',              // Full width
    alignItems: 'center',       // Center horizontally
    justifyContent: 'center',   // Center vertically
    padding: 16,                // Add padding around the form
  },

  // Style for the app title (SkyView)
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',           // Blue color for branding
    textAlign: 'center',
    marginBottom: 10,
  },

  // Style for the form section title
  header: {
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },

  // Style for each input field
  input: {
    borderWidth: 1,
    borderColor: '#ccc',
    marginBottom: 12,
    padding: 10,
    borderRadius: 5,
    width: '100%',
    maxWidth: 400, // Keeps form a nice width on wider screens
  },
});
```

SearchScreen.js: Uses FlatList to show search results.

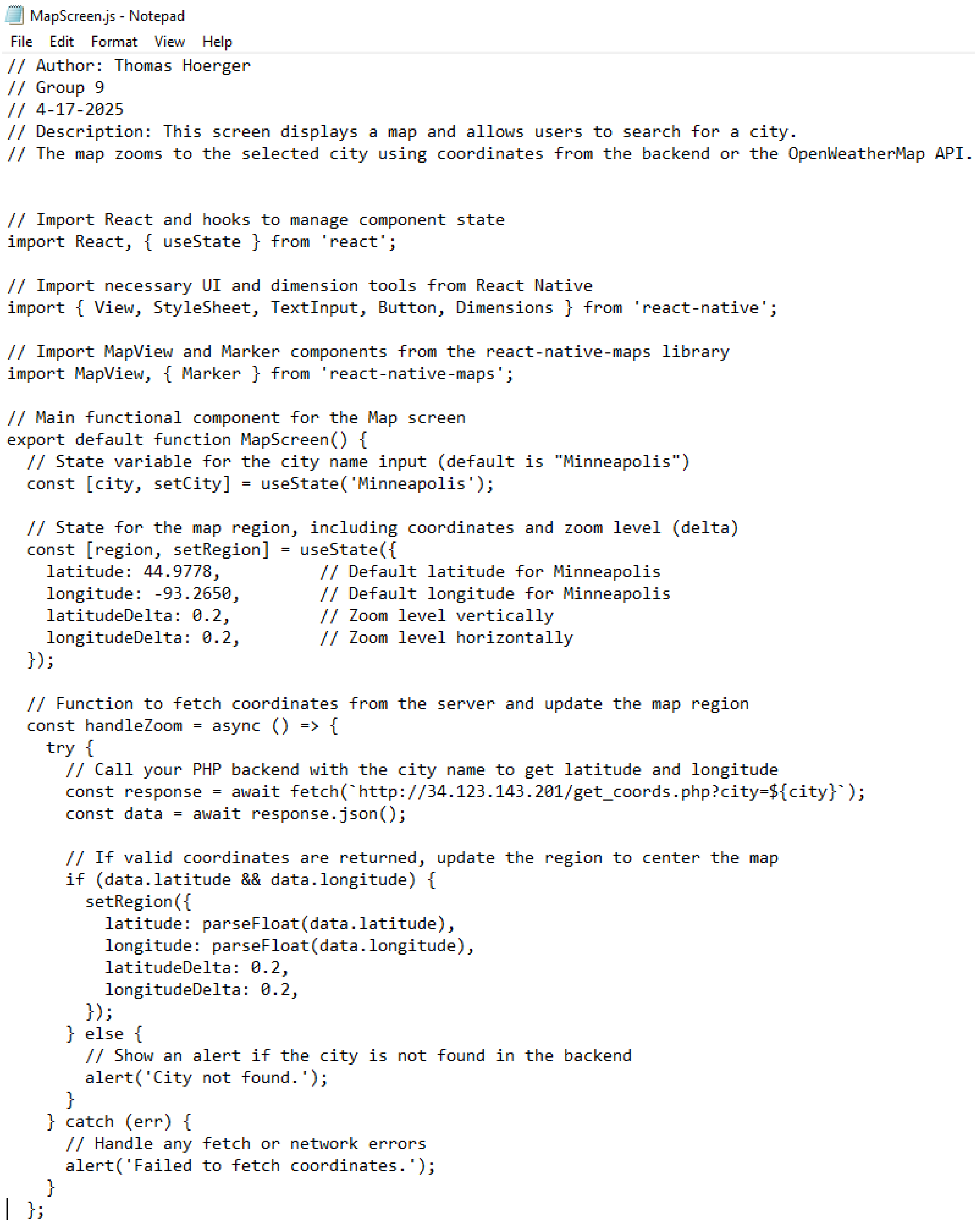


```
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: This screen lets users search for weather reports by city name.
// It retrieves and displays weather information like temperature, humidity, condition, and wind speed.

// Import necessary React and React Native components
import React, { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  Button,
  FlatList,
  StyleSheet,
  SafeAreaView,
  TouchableWithoutFeedback,
  Keyboard,
} from 'react-native';

// Main functional component for the search screen
const SearchScreen = () => {
  // State to hold the user's search input
  const [searchTerm, setSearchTerm] = useState('');
  // State to store the search results from the API
  const [results, setResults] = useState([]);

  // Function to fetch weather data based on the user's input
  const handleSearch = async () => {
    try {
      // Make a GET request to the PHP API on the VPS
      const response = await fetch(`http://34.123.143.201/search_weather.php?city=${searchTerm}`);
      const data = await response.json(); // Parse JSON response
      setResults(data); // Update state with returned weather records
    } catch (error) {
      console.error('Error fetching weather data:', error);
      setResults([]); // Reset results on error
    }
  };

  return (
    // SafeAreaView ensures UI stays inside safe boundaries (e.g., notch or status bar)
    <SafeAreaView style={styles.safeArea}>
      {/* Dismiss the keyboard when the user taps outside the input */}
      <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
        <View style={styles.container}>

          {/* Centered content: title, input, and search button */}
          <View style={styles.centeredContent}>
            <Text style={styles.appTitle}>SkyView</Text>
            <Text style={styles.title}>Search Weather</Text>

            {/* Input box for entering a city name */}
            <TextInput
              style={styles.input}
              placeholder="Enter city name"
              value={searchTerm}
              onChangeText={setSearchTerm}
            />

            {/* Button to trigger the search */}
            <Button title="Search" onPress={handleSearch} />
          </View>

          {/* Section to display search results below the input */}
          <View style={styles.resultsSection}>
            {results.length > 0 ? (
              // If results are returned, show them using a FlatList
              <FlatList
                data={results}
                keyExtractor={(item, index) => index.toString()}
                renderItem={({ item }) => (
                  <View style={styles.resultBox}>
                    <Text style={styles.city}>{item.City}</Text>
                    <Text>{item.temperature}°F | {item.humidity}% Humidity</Text>
                    <Text>{item.weather_condition} | Wind: {item.wind_speed} mph</Text>
                    <Text>{item.report_time}</Text>
                  </View>
                )}
              />
            ) : (
              // If no results and a search term exists, show a message
              searchTerm !== '' && (
                <Text style={styles.noMatch}>No match found for “{searchTerm}”</Text>
              )
            )}
          </View>
        </View>
      </TouchableWithoutFeedback>
    </SafeAreaView>
  );
};

// StyleSheet for the layout and visual styling
const styles = StyleSheet.create({
  safeArea: {
    flex: 1,
    backgroundColor: '#fff', // white background
  },
  container: {
    flex: 1,
    padding: 16,
    justifyContent: 'center', // centers vertically
    alignItems: 'center',     // centers horizontally
  centeredContent: {
    alignItems: 'center',
    justifyContent: 'center',
    marginBottom: 20,
    width: '100%',
    maxWidth: 400, // prevent stretching too far on large screens
  },
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    marginBottom: 10,
    textAlign: 'center',
  },
  title: {
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 12,
    textAlign: 'center',
  },
  input: {
    borderWidth: 1,
    borderColor: '#aaa',
    padding: 10,
    marginBottom: 10,
    borderRadius: 6,
    width: '100%',
  },
  resultsSection: {
    width: '100%',
    maxWidth: 400,
    flexShrink: 1, // allows shrinking if there's not enough space
  },
  resultBox: {
    backgroundColor: '#f0f0f0',
    padding: 10,
    borderRadius: 6,
    marginBottom: 12,
  },
  city: {
    fontSize: 18,
    fontWeight: 'bold',
  },
  noMatch: {
    marginTop: 20,
    fontStyle: 'italic',
    color: 'gray',
    textAlign: 'center',
  },
});
```

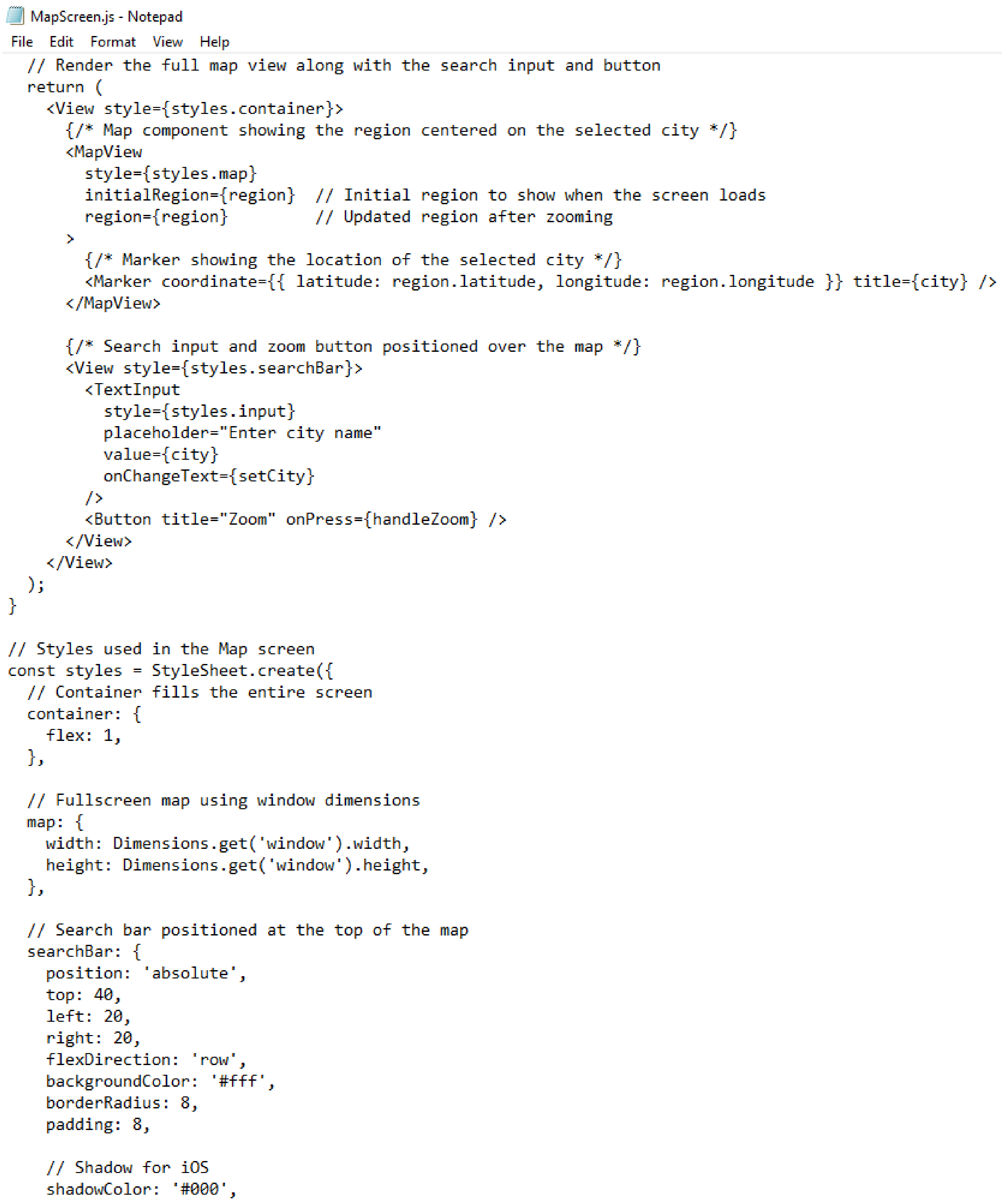


```
// Export the component so it can be used in the app
export default SearchScreen;
```

MapScreen.js: Zoom to coordinates using react-native-maps.

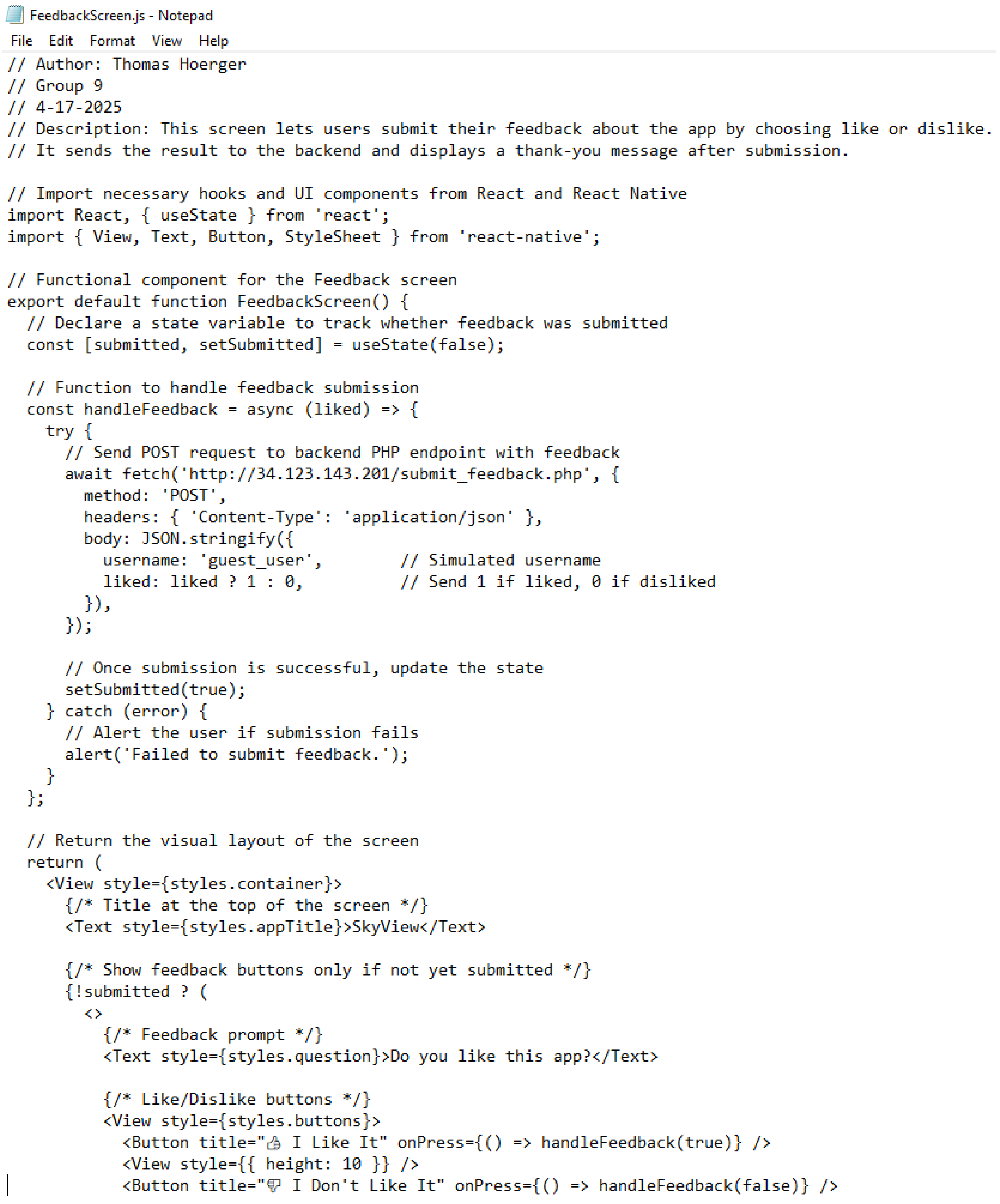


```
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: This screen displays a map and allows users to search for a city.
// The map zooms to the selected city using coordinates from the backend or the OpenWeatherMap API.


// Import React and hooks to manage component state
import React, { useState } from 'react';

// Import necessary UI and dimension tools from React Native
import { View, StyleSheet, TextInput, Button, Dimensions } from 'react-native';

// Import MapView and Marker components from the react-native-maps library
import MapView, { Marker } from 'react-native-maps';

// Main functional component for the Map screen
export default function MapScreen() {
  // State variable for the city name input (default is "Minneapolis")
  const [city, setCity] = useState('Minneapolis');

  // State for the map region, including coordinates and zoom level (delta)
  const [region, setRegion] = useState({
    latitude: 44.9778,          // Default latitude for Minneapolis
    longitude: -93.2650,        // Default longitude for Minneapolis
    latitudeDelta: 0.2,         // Zoom level vertically
    longitudeDelta: 0.2,        // Zoom level horizontally
  });

  // Function to fetch coordinates from the server and update the map region
  const handleZoom = async () => {
    try {
      // Call your PHP backend with the city name to get latitude and longitude
      const response = await fetch(`http://34.123.143.201/get_coords.php?city=${city}`);
      const data = await response.json();

      // If valid coordinates are returned, update the region to center the map
      if (data.latitude && data.longitude) {
        setRegion({
          latitude: parseFloat(data.latitude),
          longitude: parseFloat(data.longitude),
          latitudeDelta: 0.2,
          longitudeDelta: 0.2,
        });
      } else {
        // Show an alert if the city is not found in the backend
        alert('City not found.');
      }
    } catch (err) {
      // Handle any fetch or network errors
      alert('Failed to fetch coordinates.');
    }
  };
```



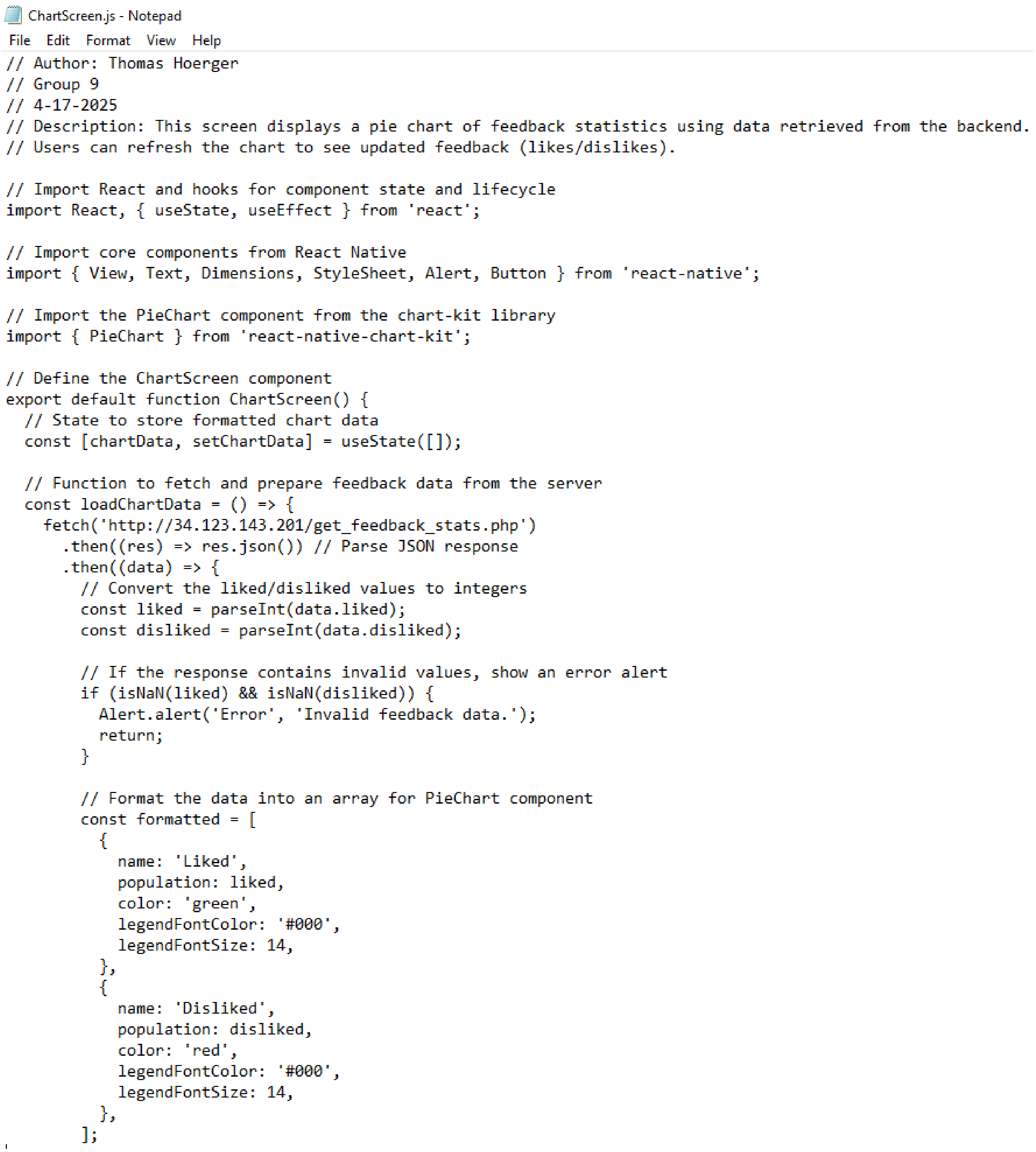
```
  // Render the full map view along with the search input and button
  return (
    <View style={styles.container}>
      {/* Map component showing the region centered on the selected city */}
      <MapView
        style={styles.map}
        initialRegion={region}  // Initial region to show when the screen loads
        region={region}         // Updated region after zooming
      >
        {/* Marker showing the location of the selected city */}
        <Marker coordinate={{ latitude: region.latitude, longitude: region.longitude }} title={city} />
      </MapView>

      {/* Search input and zoom button positioned over the map */}
      <View style={styles.searchBar}>
        <TextInput
          style={styles.input}
          placeholder="Enter city name"
          value={city}
          onChangeText={setCity}
        />
        <Button title="Zoom" onPress={handleZoom} />
      </View>
    </View>
  );
}

// Styles used in the Map screen
const styles = StyleSheet.create({
  // Container fills the entire screen
  container: {
    flex: 1,
  },

  // Fullscreen map using window dimensions
  map: {
    width: Dimensions.get('window').width,
    height: Dimensions.get('window').height,
  },

  // Search bar positioned at the top of the map
  searchBar: {
    position: 'absolute',
    top: 40,
    left: 20,
    right: 20,
    flexDirection: 'row',
    backgroundColor: '#fff',
    borderRadius: 8,
    padding: 8,

    // Shadow for iOS
    shadowColor: '#000',
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.3,
    shadowRadius: 3,

    // Elevation for Android
    elevation: 4,
  },

  // Style for the text input field
  input: {
    flex: 1,                    // Take up available space
    paddingHorizontal: 10,      // Horizontal padding inside the input
  },
});
```

FeedbackScreen.js: Like/dislike UI and submits.



```
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: This screen lets users submit their feedback about the app by choosing like or dislike.
// It sends the result to the backend and displays a thank-you message after submission.

// Import necessary hooks and UI components from React and React Native
import React, { useState } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';

// Functional component for the Feedback screen
export default function FeedbackScreen() {
  // Declare a state variable to track whether feedback was submitted
  const [submitted, setSubmitted] = useState(false);

  // Function to handle feedback submission
  const handleFeedback = async (liked) => {
    try {
      // Send POST request to backend PHP endpoint with feedback
      await fetch('http://34.123.143.201/submit_feedback.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          username: 'guest_user',        // Simulated username
          liked: liked ? 1 : 0,          // Send 1 if liked, 0 if disliked
        }),
      });

      // Once submission is successful, update the state
      setSubmitted(true);
    } catch (error) {
      // Alert the user if submission fails
      alert('Failed to submit feedback.');
    }
  };

  // Return the visual layout of the screen
  return (
    <View style={styles.container}>
      {/* Title at the top of the screen */}
      <Text style={styles.appTitle}>SkyView</Text>

      {/* Show feedback buttons only if not yet submitted */}
      {!submitted ? (
        <>
          {/* Feedback prompt */}
          <Text style={styles.question}>Do you like this app?</Text>

          {/* Like/Dislike buttons */}
          <View style={styles.buttons}>
            <Button title="👍 I Like It" onPress={() => handleFeedback(true)} />
            <View style={{ height: 10 }} />
            <Button title="👎 I Don't Like It" onPress={() => handleFeedback(false)} />
          </View>
        </>
      ) : (
        // After feedback is submitted, show thank you message
        <Text style={styles.thanks}>Thanks for your feedback! ✅</Text>
      )}
    </View>
  );
}

// Styles for the Feedback screen
const styles = StyleSheet.create({
  // Overall container for layout
  container: {
    flex: 1,                          // Fill entire screen
    paddingTop: 40,                   // Top padding for spacing
    alignItems: 'center',             // Center all items horizontally
    justifyContent: 'center',         // Center items vertically
    paddingHorizontal: 16,            // Add side padding
  },

  // SkyView app title style
  appTitle: {
    fontSize: 28,                     // Large title text
    fontWeight: 'bold',              // Bold font
    color: '#007AFF',                // iOS blue color
    marginBottom: 20,
    textAlign: 'center',
  },

  // "Do you like this app?" question style
  question: {
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },

  // Container for feedback buttons
  buttons: {
    width: '80%',                     // Constrain button width
  },

  // Style for the thank you message
  thanks: {
    fontSize: 18,
    color: 'green',
    textAlign: 'center',
  },
});
```

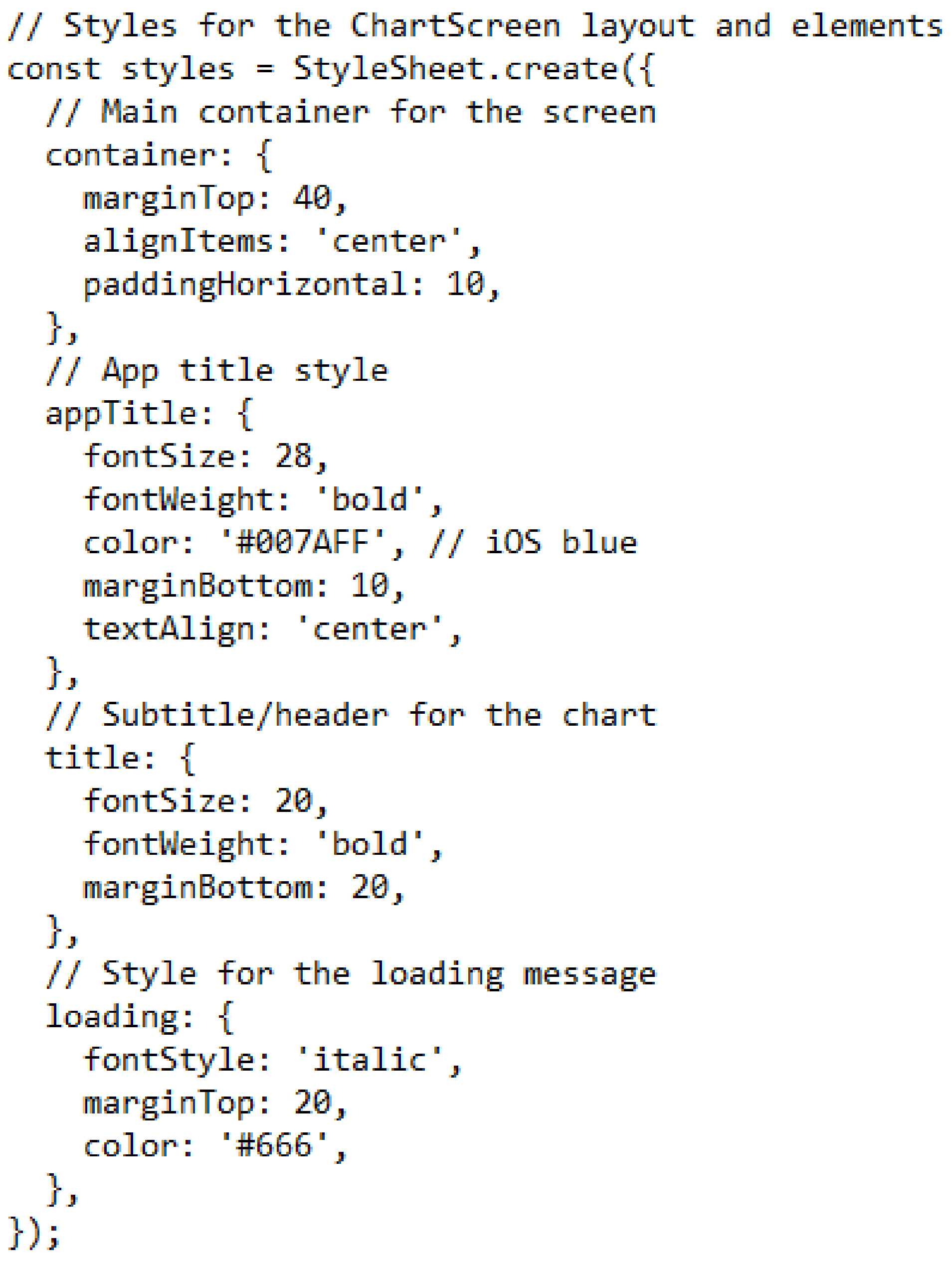
ChartScreen.js: Pie chart using react-native-chart-kit.



ChartScreen.js - Notepad

File Edit Format View Help

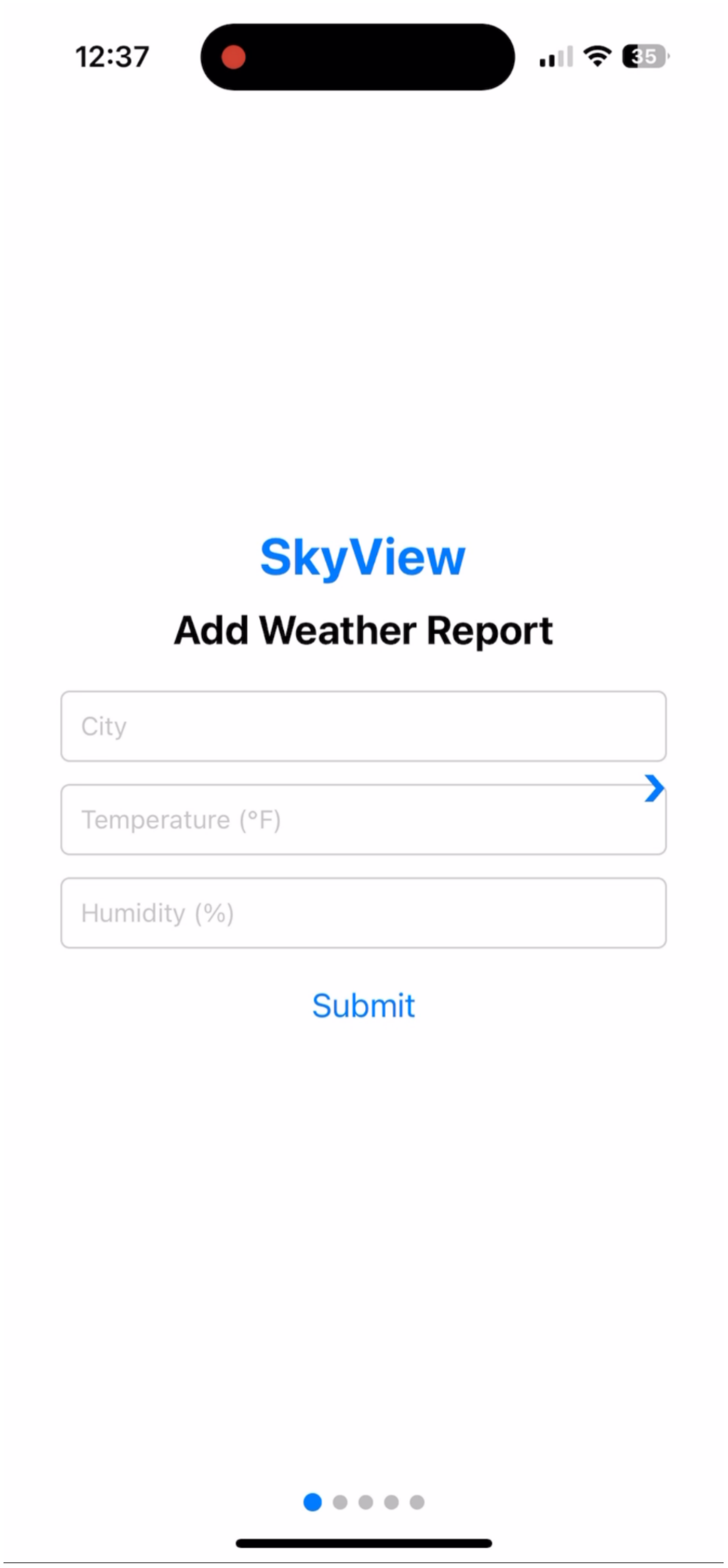
```
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: This screen displays a pie chart of feedback statistics using data retrieved from the backend.
// Users can refresh the chart to see updated feedback (likes/dislikes).

// Import React and hooks for component state and lifecycle
import React, { useState, useEffect } from 'react';

// Import core components from React Native
import { View, Text, Dimensions, StyleSheet, Alert, Button } from 'react-native';

// Import the PieChart component from the chart-kit library
import { PieChart } from 'react-native-chart-kit';

// Define the ChartScreen component
export default function ChartScreen() {
  // State to store formatted chart data
  const [chartData, setChartData] = useState([]);

  // Function to fetch and prepare feedback data from the server
  const loadChartData = () => {
    fetch('http://34.123.143.201/get_feedback_stats.php')
      .then((res) => res.json()) // Parse JSON response
      .then((data) => {
        // Convert the liked/disliked values to integers
        const liked = parseInt(data.liked);
        const disliked = parseInt(data.disliked);

        // If the response contains invalid values, show an error alert
        if (isNaN(liked) && isNaN(disliked)) {
          Alert.alert('Error', 'Invalid feedback data.');
          return;
        }

        // Format the data into an array for PieChart component
        const formatted = [
          {
            name: 'Liked',
            population: liked,
            color: 'green',
            legendFontColor: '#000',
            legendFontSize: 14,
          },
          {
            name: 'Disliked',
            population: disliked,
            color: 'red',
            legendFontColor: '#000',
            legendFontSize: 14,
          },
        ];
        // Update the state with the formatted chart data
        setChartData(formatted);
      })
      .catch((err) => {
        // Handle any network or parsing errors
        console.error('Chart fetch error:', err);
        Alert.alert('Error', 'Could not load chart data.');
      });
  };

  // useEffect runs once on mount to fetch the initial chart data
  useEffect(() => {
    loadChartData();
  }, []);

  // Render the layout
  return (
    <View style={styles.container}>
      {/* App title displayed at the top */}
      <Text style={styles.appTitle}>SkyView</Text>

      {/* Subtitle for the chart section */}
      <Text style={styles.title}>User Feedback</Text>

      {/* Button to manually refresh the chart */}
      <Button title="Refresh Chart" onPress={loadChartData} />

      {/* If chartData is available, show the PieChart. Otherwise, show loading message. */}
      {chartData.length > 0 ? (
        <PieChart
          data={chartData} // Data to display in the chart
          width={Dimensions.get('window').width - 20} // Chart width based on screen size
          height={220} // Chart height
          chartConfig={{
            backgroundColor: '#fff',
            backgroundGradientFrom: '#fff',
            backgroundGradientTo: '#fff',
            color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`, // Chart segment color
            labelColor: () => '#000', // Label color
          }}
          accessor="population" // Tells PieChart which key holds the numeric values
          backgroundColor="transparent"
          paddingLeft="15"
          absolute // Shows numeric values inside the chart
        />
      ) : (
        <Text style={styles.loading}>Loading chart data...</Text>
      )}
    </View>
  );
}
```



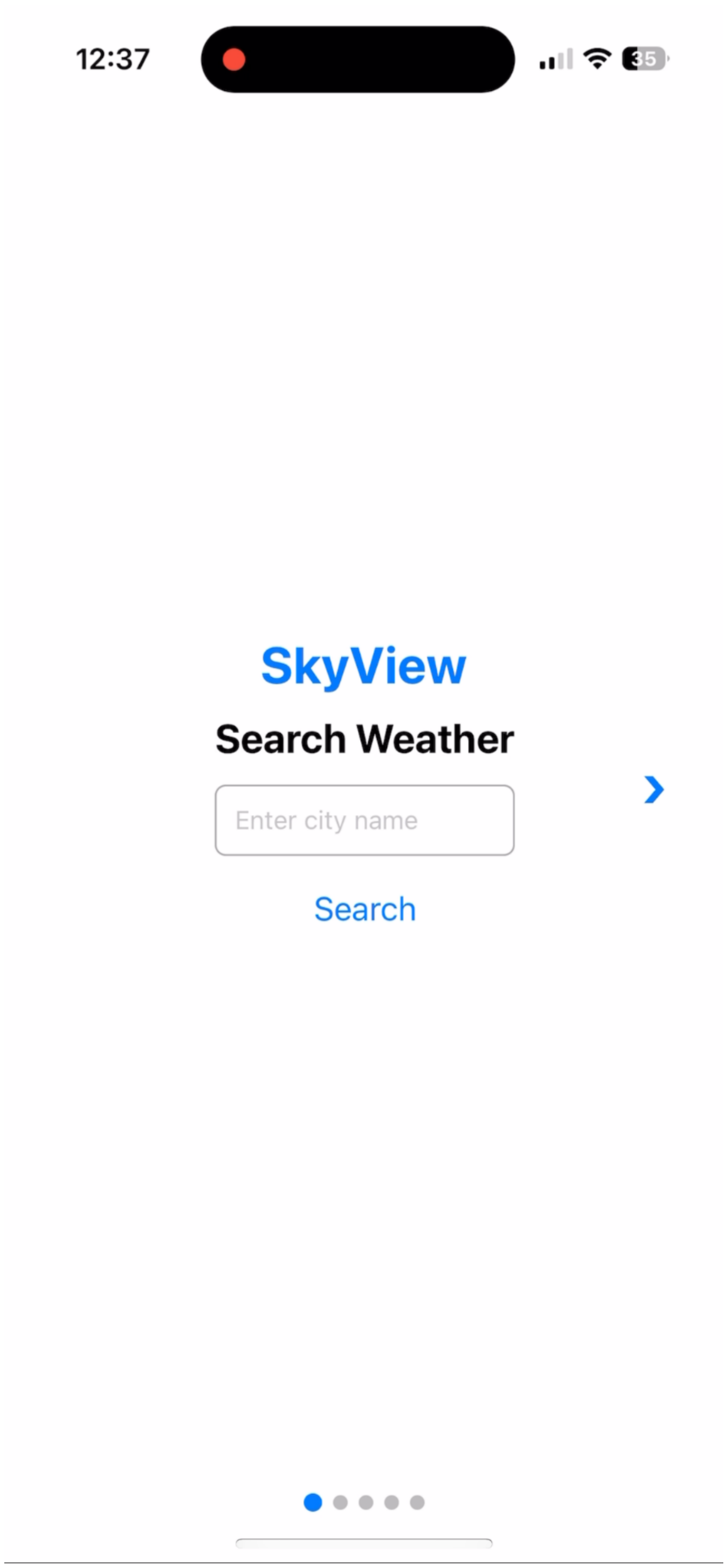
```
// Styles for the ChartScreen layout and elements
const styles = StyleSheet.create({
  // Main container for the screen
  container: {
    marginTop: 40,
    alignItems: 'center',
    paddingHorizontal: 10,
  },
  // App title style
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF', // iOS blue
    marginBottom: 10,
    textAlign: 'center',
  },
  // Subtitle/header for the chart
  title: {
    fontSize: 20,
    fontWeight: 'bold',
    marginBottom: 20,
  },
  // Style for the loading message
  loading: {
    fontStyle: 'italic',
    marginTop: 20,
    color: '#666',
  },
});
```

6. Included Features:
   - Installed react-native-swiper for navigation.
   - Added custom dot and arrow indicators.

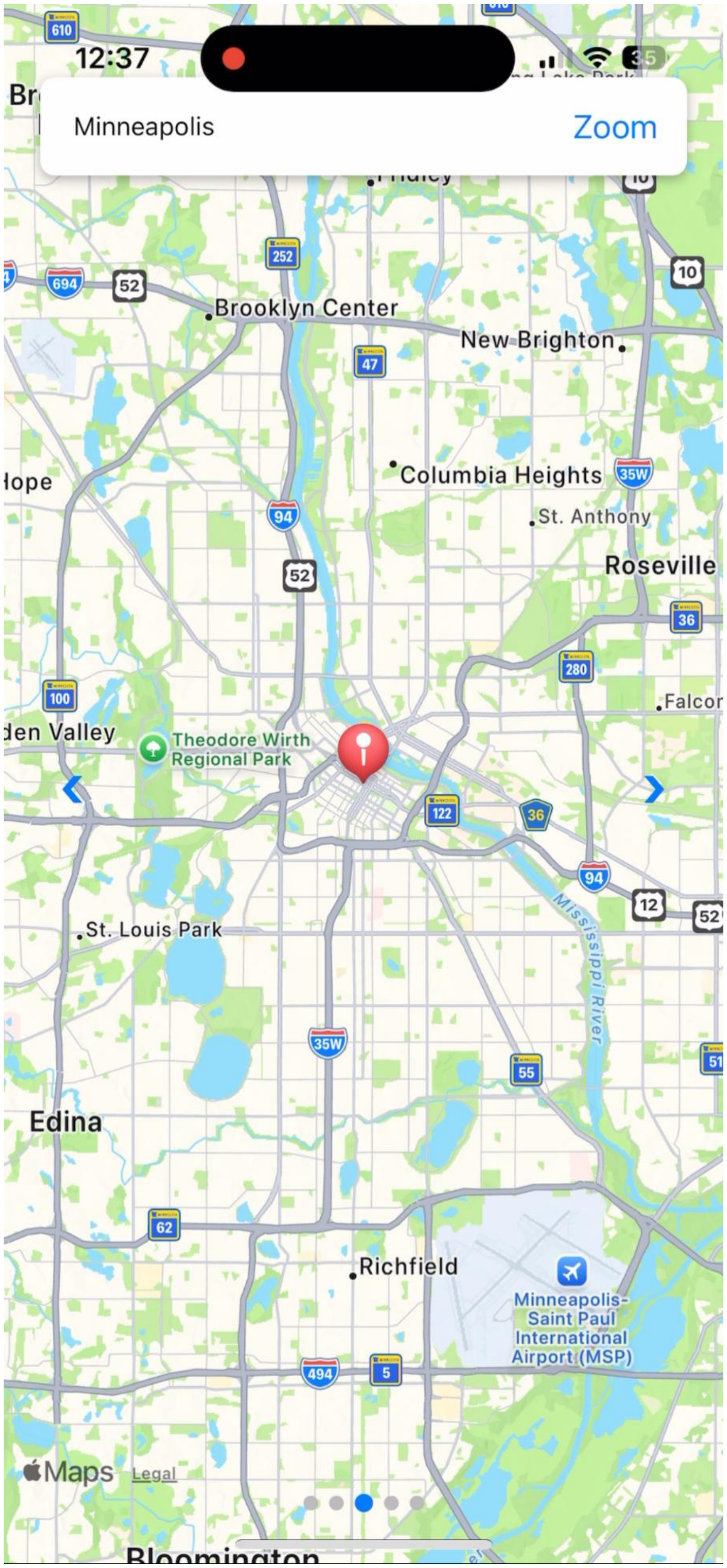
- Implemented add feature so that the user can add to the database via the app.



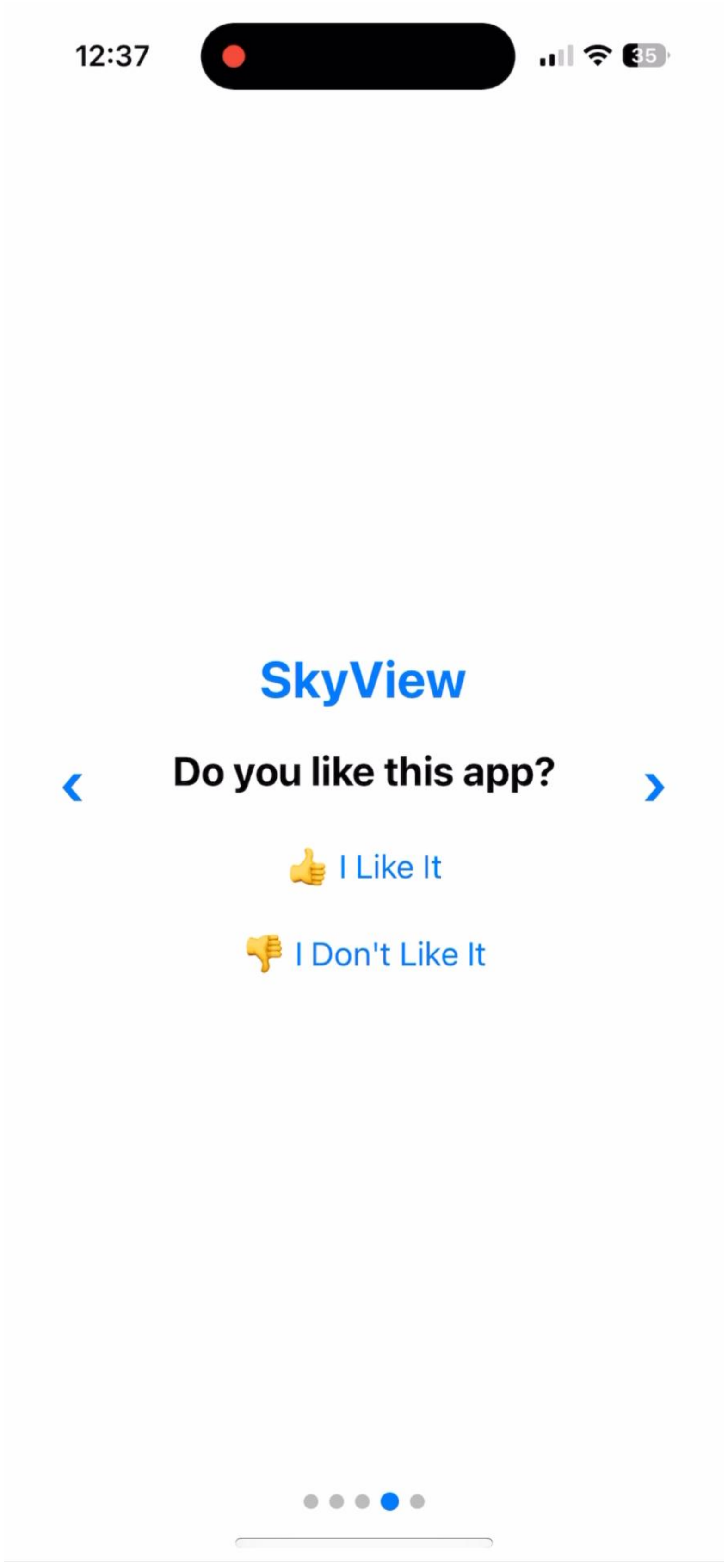
- Added search so that the user can display weather reports from the searched city.

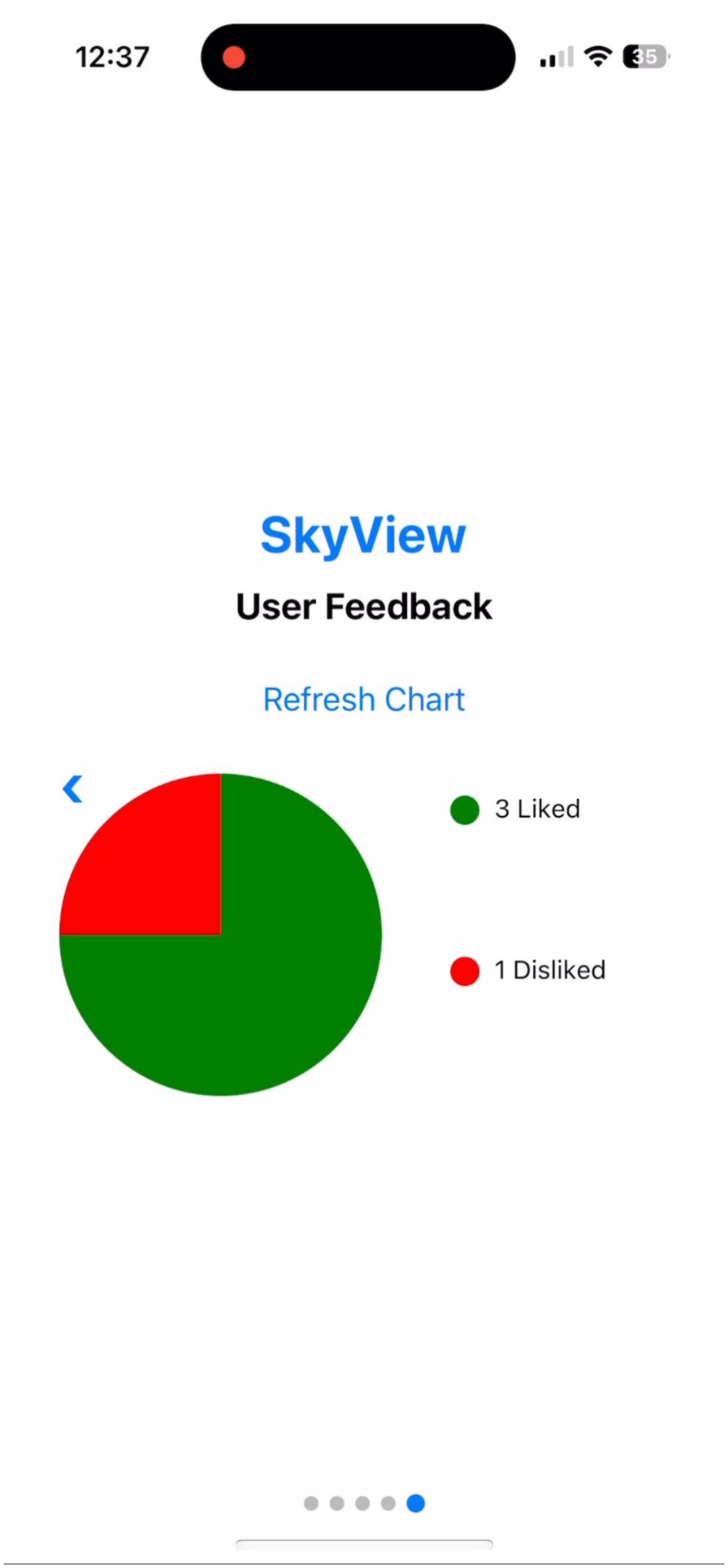


- Added map feature so that the user can search a city and display it on a map.
    - Integrated OpenWeatherMap API (Geo API) into get_coords.php. So if the city isn't in the database, script fetches coordinates live from API



- Feedback + Chart System
    - Users select like/dislike and submit.
    - Chart displays live data from database using a refresh button.





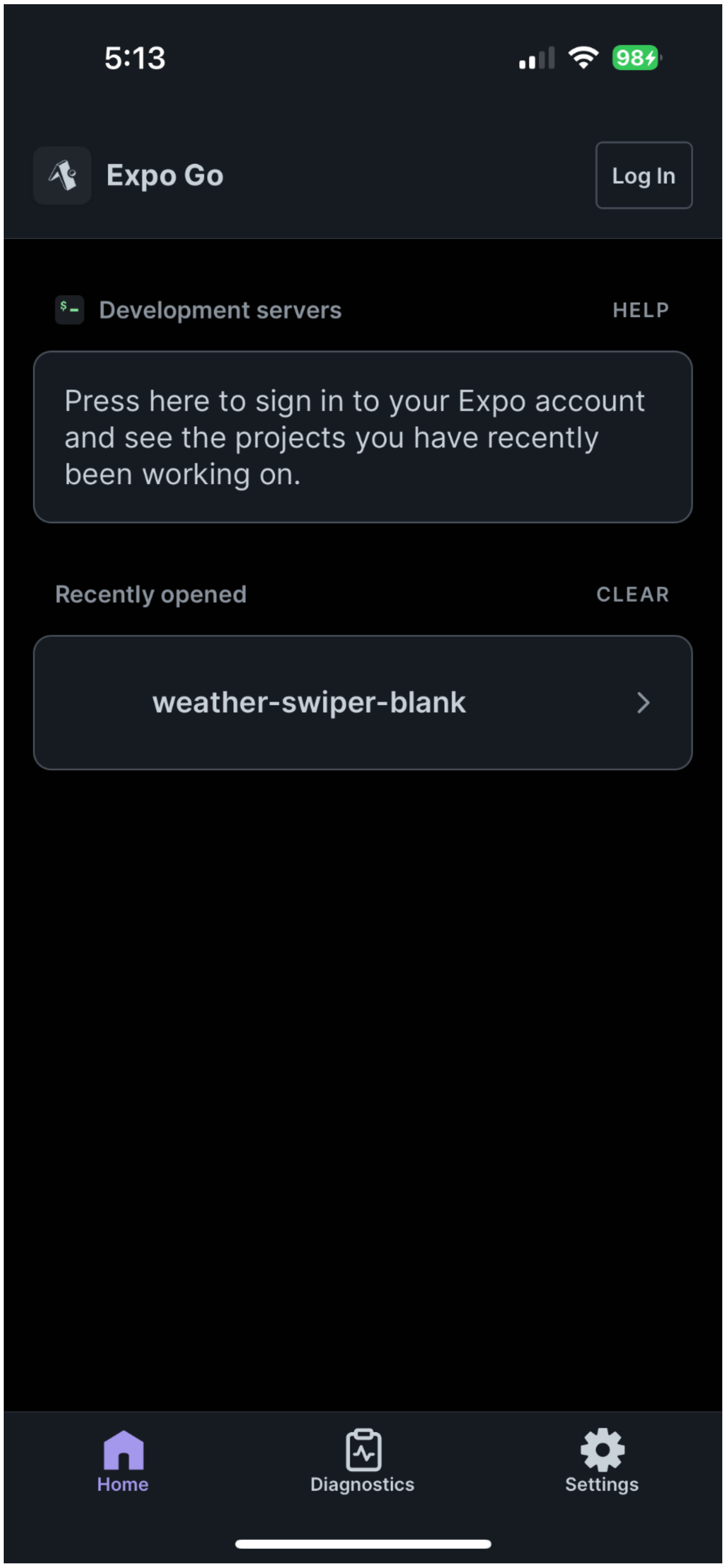
SkyView

User Feedback

Refresh Chart

- Ensure Mobile Responsiveness:
    - Used responsive styles to support both mobile and laptop views.
    - Components such as maps, charts, and form fields scale based on screen size.

7. Install Expo Go app and scan QR code to use app.



Press here to sign in to your Expo account and see the projects you have recently been working on.

Summary:

The SkyView swiper app project successfully integrates a full-stack weather reporting system with modern mobile app development. Users can add and search for city-specific weather reports, view dynamic map locations, and submit real-time feedback. The system is hosted on a cloud-based VPS using Apache, PHP, and MySQL, and interacts with a React Native frontend built using Expo.

Key features include adding/searching the database, dynamic geolocation using OpenWeatherMap, map zoom functionality, and a feedback chart using real-time database stats. The application supports full mobile and desktop responsiveness, ensuring a good experience across platforms.

Overall, this project demonstrates that I can design and deploy a complete, data-driven application that bridges backend, frontend, APIs, and live hosting infrastructure.