Thomas Hoerger

Final Task

4/24/2025

Step-by-Step Implementation

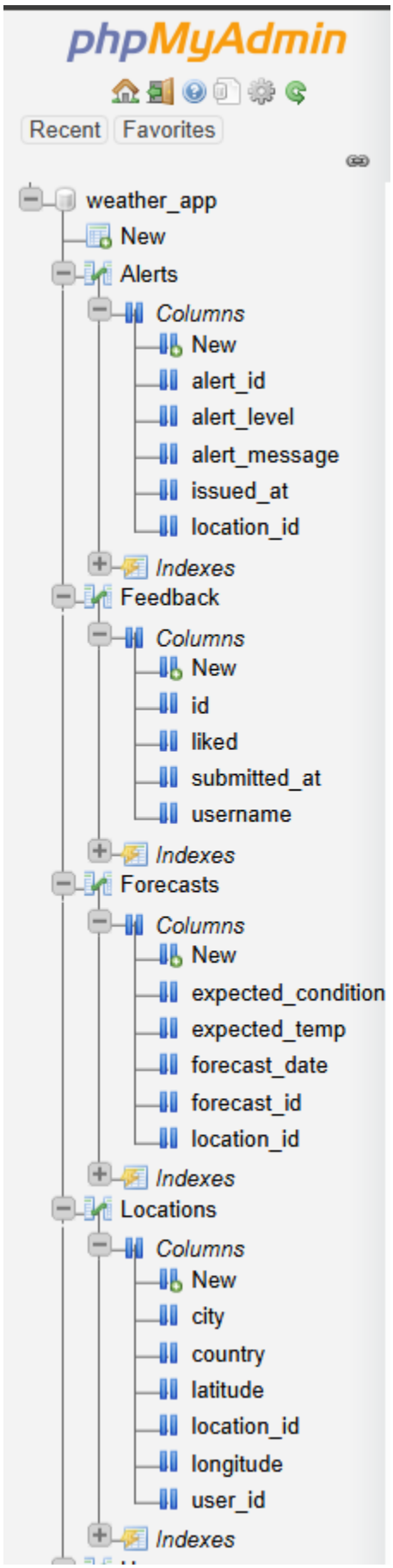
1. VPS Setup and MySQL Database
   - Deployed Ubuntu server on Google Cloud VPS.
   - Installed Apache, PHP, and MySQL.

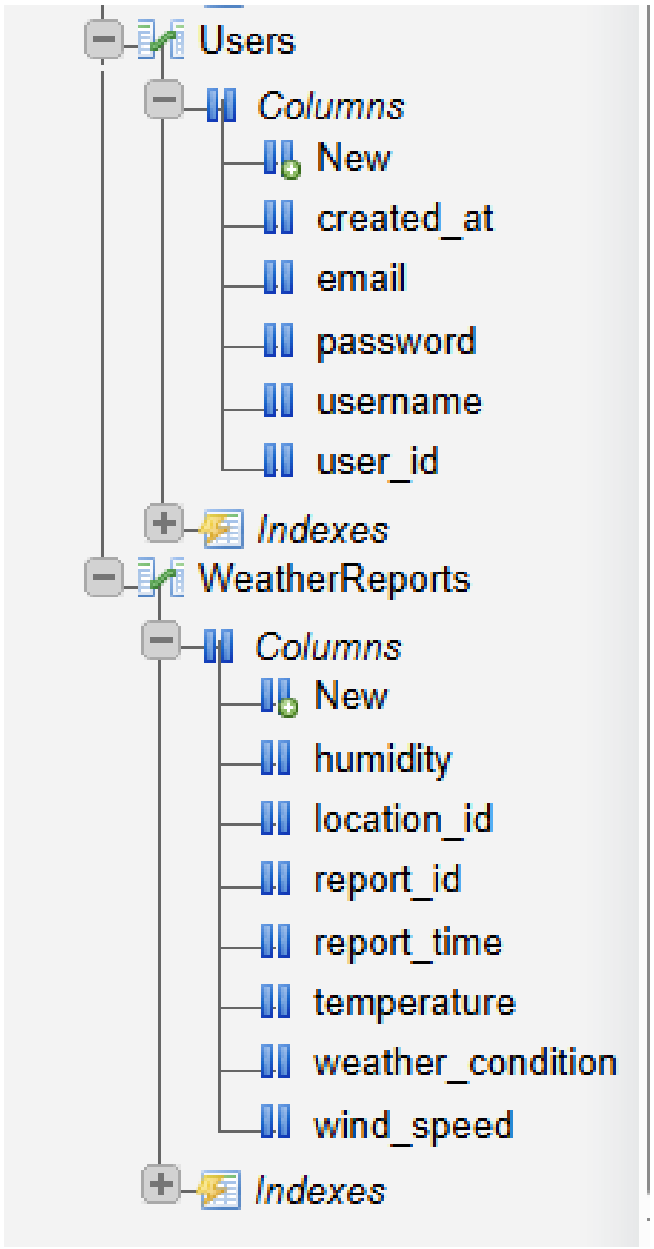
2. Created Expo project with npx create-expo-app weather-swiper-blank after installing it on the local system:

```
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.





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

   add_weather.php: Adds data to the database.

```php
<?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.

header('Content-Type: application/json'); // Set response content type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check if connection to the database failed
if ($conn->connect_error) {
    echo json_encode(["status" => "error", "message" => "DB connection failed"]);
    exit;
}

// Retrieve the JSON data sent in the POST request
$data = json_decode(file_get_contents("php://input"), true);

// Extract individual values from the decoded data
$city = $data['city'];
$temperature = $data['temperature'];
$humidity = $data['humidity'];

// Function to fetch geographical coordinates for a city using OpenWeatherMap API
function fetchCoordinates($city) {
    $apiKey = 'a34bb9d6a46791646f2426d3fada7996'; // OpenWeatherMap API key
    $url = "http://api.openweathermap.org/geo/1.0/direct?q=" . urlencode($city) . "&limit=1&appid=" . $apiKey;

    $response = file_get_contents($url); // Make API call to get location data
    if ($response) {
        $data = json_decode($response, true);
        // If valid coordinates are returned, use them
        if (isset($data[0]['lat']) && isset($data[0]['lon'])) {
            return [$data[0]['lat'], $data[0]['lon']];
        }
    }
    return [0.0, 0.0]; // Return default coordinates if not found
}

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

// If city exists, retrieve its location_id
if ($row = $result->fetch_assoc()) {
    $location_id = $row['location_id'];
} else {
    // Step 2: If city does not exist, fetch coordinates and insert into Locations
    list($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 location insertion fails, return an error
    if (!$insertLoc->execute()) {
        echo json_encode(["status" => "error", "message" => "Failed to insert location", "details" => $insertLoc->error]);
        exit;
    }

    $location_id = $insertLoc->insert_id; // Get ID of newly inserted location
}

// Step 3: Insert the weather report into the WeatherReports table
$default_condition = 'Clear'; // Default weather condition
$default_wind = 5; // Default wind speed

$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 insertion is successful, return success status
if ($insertWeather->execute()) {
    echo json_encode(["status" => "success"]);
} else {
    // If weather report insertion fails, return an error
    echo json_encode(["status" => "error", "message" => "Failed to insert weather report", "details" => $insertWeather->error]);
}

$conn->close(); // Close the database connection
?>
```

search_weather.php: Retrieves weather by city.

```php
<?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.

// Enable error reporting for debugging during development
error_reporting(E_ALL);
ini_set('display_errors', 1);

// Set the content type of the response to JSON
header("Content-Type: application/json");

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

// Check for a connection error and return a 500 response if it fails
if ($conn->connect_error) {
  http_response_code(500);
  echo json_encode(["error" => "DB connection failed", "details" => $conn->connect_error]);
  exit();
}

// Retrieve the city search term from the query parameters (default to empty string)
$city = $_GET['city'] ?? '';

// Construct SQL query to fetch weather reports for cities that match the search term
// Joins the WeatherReports and Locations tables on 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%'
";

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

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

```php
// Initialize an array to store the results
$data = [];

// Loop through each row of the result set and add it to the array
while ($row = $res->fetch_assoc()) {
  $data[] = $row;
}

// Return the results as a JSON response
echo json_encode($data);
?>
```

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

```php
<?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.

header('Content-Type: application/json'); // Set response content type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check for a failed database connection
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]); // Return error as JSON
    exit; // Terminate the script
}

// Retrieve the city parameter from the URL query string
$city = $_GET['city'];

// Step 1: Attempt to find coordinates in the local database
$stmt = $conn->prepare("SELECT latitude, longitude FROM Locations WHERE city = ?");
$stmt->bind_param("s", $city); // Bind the city parameter to the query
$stmt->execute();
$result = $stmt->get_result();

// If the city is found in the database, return the coordinates
if ($row = $result->fetch_assoc()) {
    echo json_encode([
        "latitude" => $row['latitude'],
        "longitude" => $row['longitude'],
        "source" => "db" // Indicate the source is the database
    ]);
    exit;
}

// Step 2: If not found in the database, attempt to get coordinates from the OpenWeatherMap API
$apiKey = 'a34bb9d6a46791646f2426d3fada7996'; // OpenWeatherMap API key
$url = "http://api.openweathermap.org/geo/1.0/direct?q=" . urlencode($city) . "&limit=1&appid=" . $apiKey;

// Make the API request
$response = file_get_contents($url);
$data = json_decode($response, true);

// If valid coordinates are returned from the API, output them
if (isset($data[0]['lat']) && isset($data[0]['lon'])) {
    echo json_encode([
        "latitude" => $data[0]['lat'],
        "longitude" => $data[0]['lon'],
        "source" => "api" // Indicate the source is the API
    ]);
} else {
    // If the city is not found in the API response, return an error message
    echo json_encode(["error" => "City not found"]);
}
```

```php
?>
```

submit_feedback.php: Stores like/dislike values.

```php
<?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.

header("Content-Type: application/json"); // Set the response type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli("localhost", "root", "Sweets@01", "weather_app");

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

// Decode JSON data from the POST request body
$data = json_decode(file_get_contents('php://input'), true);

// Sanitize and extract the username; default to 'anonymous' if not provided
$username = $conn->real_escape_string($data['username'] ?? 'anonymous');

// Get the liked value and ensure it is an integer (1 for like, 0 for dislike)
$liked = intval($data['liked'] ?? 0);

// Insert the feedback entry into the database
$conn->query("INSERT INTO Feedback (username, liked) VALUES ('$username', $liked)");

// Return a success response
echo json_encode(["status" => "ok"]);
?>
```

get_feedback_stats.php: Returns feedback data.

```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Returns total count of likes and dislikes from the Feedback table as JSON.

header('Content-Type: application/json'); // Set the response type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check for a connection error and return an error message if it occurs
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]);
    exit;
}

// SQL query to count likes and dislikes using conditional aggregation
$query = "SELECT
            SUM(CASE WHEN liked = 1 THEN 1 ELSE 0 END) AS liked,
            SUM(CASE WHEN liked = 0 THEN 1 ELSE 0 END) AS disliked
          FROM Feedback";

// Execute the query
$result = $conn->query($query);

// If the query is successful and returns results, encode them as JSON
if ($result && $row = $result->fetch_assoc()) {
    echo json_encode([
        "liked" => $row['liked'] ?? "0",       // Use 0 as fallback if null
        "disliked" => $row['disliked'] ?? "0"  // Use 0 as fallback if null
    ]);
} else {
    // If the query fails, return an error message with details
    echo json_encode(["error" => "Query failed", "details" => $conn->error]);
}

// Close the database connection
$conn->close();
?>
```

db_connect.php: Establishes connection to the database.

```php
// Author: Thomas Hoerger
// Group 9
// 4-17-2025
// Description: Establishes a connection to the MySQL database for the weather_app system.
// If the connection fails, the script stops and displays an error message.

<?php
$servername = "localhost";  // Change to VPS IP if accessing remotely
$username = "root";         // Use your MySQL username
$password = "Sweets@01"; // Use your MySQL root password
$dbname = "weather_app";    // Your database name

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

get_all_weather.php: Returns records from WeatherReports table.

```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Returns all records from WeatherReports table as JSON.

header('Content-Type: application/json'); // Set the response content type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check if the database connection failed
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]); // Return error as JSON
    exit; // Terminate the script if connection fails
}

// Execute a SQL query to retrieve all records from the WeatherReports table
$result = $conn->query("SELECT * FROM WeatherReports");

// Initialize an empty array to store the results
$data = [];

// Loop through each row in the result set and add it to the array
while ($row = $result->fetch_assoc()) {
    $data[] = $row;
}

// Encode the array of records into JSON and return it
echo json_encode($data);

// Close the database connection
$conn->close();
?>
```

get_table_data.php: Returns table data from database.

```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Returns all rows from any valid table as JSON.

header('Content-Type: application/json'); // Set response type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check for connection errors and return an error response if one occurs
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]);
    exit;
}

// Sanitize the 'table' query parameter to prevent SQL injection
// Only allow alphanumeric characters and underscores
$table = preg_replace('/[^a-zA-Z0-9_]/', '', $_GET['table'] ?? '');

// If the table name is invalid or empty, return an error
if (!$table) {
    echo json_encode(["error" => "Invalid table"]);
    exit;
}

// Construct and run a SQL query to retrieve all rows from the specified table
$result = $conn->query("SELECT * FROM `$table`");

// If the query fails, return an error message with details
if (!$result) {
    echo json_encode(["error" => "Query failed", "details" => $conn->error]);
    exit;
}

// Initialize an array to store each row of the result
$rows = [];
while ($row = $result->fetch_assoc()) {
    $rows[] = $row;
}

// Return the table name and all rows as a JSON response
echo json_encode(["table" => $table, "rows" => $rows]);

// Close the database connection
$conn->close();
?>
```

get_table_list.php: Returns table names from database.

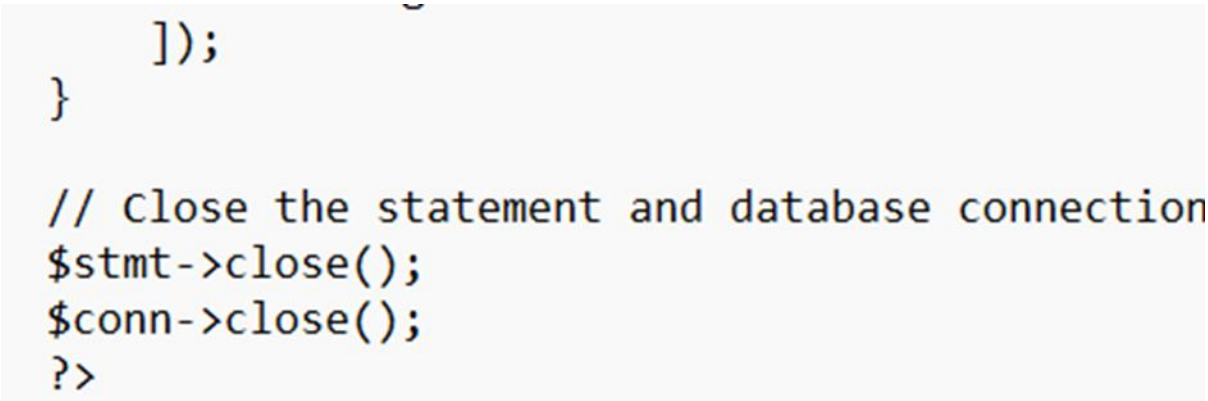
```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Returns a list of all table names in the weather_app database.

header('Content-Type: application/json'); // Set the response type to JSON

// Establish a connection to the MySQL database
$conn = new mysqli('localhost', 'root', 'Sweets@01', 'weather_app');

// Check for connection errors and return an error if the connection fails
if ($conn->connect_error) {
    echo json_encode(["error" => "DB connection failed"]);
    exit;
}

// Initialize an array to store the table names
$tables = [];

// Execute SQL command to list all tables in the database
$result = $conn->query("SHOW TABLES");

// Loop through the result set and add each table name to the array
while ($row = $result->fetch_array()) {
    $tables[] = $row[0];
}

// Encode the list of tables as JSON and return it
echo json_encode($tables);

// Close the database connection
$conn->close();
?>
```

login_user.php: Logs user in by checking database.

```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Logs a user in by checking the username and plain-text password
// from the Users table. This version skips hashing for demo/testing purposes.

header('Content-Type: application/json'); // Set response content type to JSON

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

// Decode the JSON payload from the request body
$data = json_decode(file_get_contents("php://input"), true);
$username = $data['username'];
$password = $data['password'];

// Prepare a SQL statement to select the password for the given username
$stmt = $conn->prepare("SELECT password FROM Users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
$res = $stmt->get_result();

// Check if a matching user was found
if ($row = $res->fetch_assoc()) {
    // Compare the provided plain-text password with the stored password
    if ($password === $row['password']) {
        // Login successful
        echo json_encode(["status" => "success", "username" => $username]);
    } else {
        // Password did not match
        echo json_encode(["status" => "error", "message" => "Incorrect password"]);
    }
} else {
    // No user found with the provided username
    echo json_encode(["status" => "error", "message" => "User not found"]);
}

// Close the database connection
$conn->close();
?>
```

register_user.php: Handles user registration.

```php
<?php
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Handles user registration. Checks for duplicate username
// and inserts plain-text password (for demo use only).

header('Content-Type: application/json'); // Set response type to JSON

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

// Decode JSON data from the request body
$data = json_decode(file_get_contents("php://input"), true);

// Extract user-submitted fields
$username = $data['username'];
$password = $data['password'];
$email = $data['email'];
$created_at = date('Y-m-d H:i:s'); // Get current timestamp

// Check if the username already exists in the database
$check = $conn->prepare("SELECT user_id FROM Users WHERE username = ?");
$check->bind_param("s", $username);
$check->execute();
$check->store_result();

// If the username is already taken, return an error response
if ($check->num_rows > 0) {
    echo json_encode([
        "status" => "error",
        "message" => "Username already exists"
    ]);
    $check->close();
    $conn->close();
    exit; // Terminate script
}

$check->close(); // Close the prepared statement for the username check

// Insert new user into the Users table
// Note: Password is stored in plain-text here for demonstration purposes only
$stmt = $conn->prepare("INSERT INTO Users (username, password, email, created_at) VALUES (?, ?, ?, ?)");
$stmt->bind_param("ssss", $username, $password, $email, $created_at);

// If insertion is successful, return success response
if ($stmt->execute()) {
    echo json_encode([
        "status" => "success",
        "message" => "User registered successfully"
    ]);
} else {
    // Return an error if the insertion fails
    echo json_encode([
        "status" => "error",
        "message" => "Database insert failed"
```



```php
    ]);
}

// Close the statement and database connection
$stmt->close();
$conn->close();
?>
```

VPS project directory:

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

Step 5 Implemented 8 screens and 4 other JavaScript files in the Expo project on my local machine using JavaScript:

App.js: App entry point.

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: App entry point using navigation. Shows login/register until user is authenticated.

import React, { useState } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

import LoginScreen from './screens/auth/LoginScreen';
import RegisterScreen from './screens/auth/RegisterScreen';
import MainSwiper from './MainSwiper';

// Create navigation stack
const Stack = createNativeStackNavigator();

export default function App() {
  // Manage login state (null if not logged in)
  const [user, setUser] = useState(null);

  // Handler to update user state on login
  const handleLogin = (username) => {
    setUser(username);
  };

  // Handler to reset user state on logout
  const handleLogout = () => {
    setUser(null);
  };

  return (
    <NavigationContainer>
      <Stack.Navigator screenOptions={{ headerShown: false }}>
        {!user ? (
          // Show login and registration if user is not logged in
          <>
            <Stack.Screen name="Login">
              {props => <LoginScreen {...props} onLogin={handleLogin} />}
            </Stack.Screen>
            <Stack.Screen name="Register" component={RegisterScreen} />
          </>
        ) : (
          // Show main swiper navigation once logged in
          <Stack.Screen name="Main">
            {props => (
              <MainSwiper {...props} username={user} onLogout={handleLogout} />
            )}
          </Stack.Screen>
        )}
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

index.js: Registers app components.

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Entry point for the React Native app.
// Registers the App component with Expo to ensure proper environment setup
// for both Expo Go and native builds.

import { registerRootComponent } from 'expo';

import App from './App';

// registerRootComponent calls AppRegistry.registerComponent('main', () => App);
// It also ensures that whether you load the app in Expo Go or in a native build,
// the environment is set up appropriately
registerRootComponent(App);
```

MainSwiper.js: Swiper navigation.

```
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Swiper navigation for SkyView app. Includes welcome/logout header on all screens except MapScreen. DisplayScreen added before Map.

import React from 'react';
import { View, Text, StyleSheet, TouchableOpacity, SafeAreaView } from 'react-native';
import Swiper from 'react-native-swiper';

// Screens
import AddScreen from './screens/AddScreen';
import SearchScreen from './screens/SearchScreen';
import DisplayScreen from './screens/DisplayScreen';  // NEW screen!
import MapScreen from './screens/MapScreen';
import FeedbackScreen from './screens/FeedbackScreen';
import ChartScreen from './screens/ChartScreen';

export default function MainSwiper({ username, onLogout }) {
  // Wrapper component to display header unless suppressed
  const ScreenWrapper = ({ children, showHeader = true }) => (
    <SafeAreaView style={styles.safe}>
      {showHeader && (
        <View style={styles.header}>
          <Text style={styles.welcome}>Welcome, {username}</Text>
          <TouchableOpacity onPress={onLogout}>
            <Text style={styles.logout}>Logout</Text>
          </TouchableOpacity>
        </View>
      )}
      <View style={styles.content}>
        {children}
      </View>
    </SafeAreaView>
  );

  return (
    <Swiper
      loop={false}
      showsPagination={true}
      showsButtons={true}
      scrollEnabled={true}
      scrollAnimationDuration={600}
      dotStyle={{ backgroundColor: '#bbb', width: 8, height: 8, borderRadius: 4 }}
      activeDotStyle={{ backgroundColor: '#007AFF', width: 10, height: 10, borderRadius: 5 }}
      nextButton={<Text style={styles.arrow}>›</Text>}
      prevButton={<Text style={styles.arrow}>‹</Text>}
    >
      {/* Each screen wrapped to include consistent header layout */}
      <View style={styles.slide}>
        <ScreenWrapper><AddScreen /></ScreenWrapper>
      </View>

      <View style={styles.slide}>
        <ScreenWrapper><SearchScreen /></ScreenWrapper>
      </View>
```

```
        <View style={styles.slide}>
          <ScreenWrapper><DisplayScreen /></ScreenWrapper>
        </View>

        <View style={{ flex: 1 }}>
          <MapScreen />
        </View>

        <View style={styles.slide}>
          <ScreenWrapper><FeedbackScreen /></ScreenWrapper>
        </View>

        <View style={styles.slide}>
          <ScreenWrapper><ChartScreen /></ScreenWrapper>
        </View>
      </Swiper>
    );
}

const styles = StyleSheet.create({
  safe: {
    flex: 1,
    backgroundColor: '#fff',
  },
  content: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    paddingHorizontal: 15,
  },
  slide: {
    flex: 1,
    backgroundColor: '#fff',
  },
  header: {
    paddingTop: 10,
    paddingBottom: 10,
    paddingHorizontal: 20,
    backgroundColor: '#f2f2f2',
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
  },
  welcome: {
    fontSize: 16,
    color: '#333',
    fontWeight: '500',
  },
  logout: {
    color: '#007AFF',
    fontWeight: 'bold',
    fontSize: 16,
  },
  arrow: {
    color: '#007AFF',
    fontSize: 36,
    fontWeight: 'bold',
    paddingHorizontal: 20,
  },
});
```

AnimatedHeader.js: Animated header for Lottie.

```
// Author: Thomas Hoerger
// Group 9 -
//Date: 4-24-2025
// Description: Reusable animated header with SkyView title and Lottie.

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import LottieView from 'lottie-react-native';

export default function AnimatedHeader({ username, onLogout }) {
  return (
    <View style={styles.header}>
      {/* Sun animation at the top */}
      <LottieView
        source={require('../assets/animations/sun.json')}
        autoPlay
        loop
        style={styles.animation}
      />

      {/* Main app title */}
      <Text style={styles.title}>SkyView</Text>

      {/* Optional welcome message if user is logged in */}
      {username && (
        <Text style={styles.welcome}>Welcome, {username}</Text>
      )}
    </View>
  );
}

// Styles for header layout
const styles = StyleSheet.create({
  header: {
    alignItems: 'center',
    paddingTop: 20,
  },
  animation: {
    width: 100,
    height: 100,
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    marginTop: -10,     // Moves title closer to the animation
    marginBottom: 8,
  },
  welcome: {
    fontSize: 16,
    color: '#444',
  },
});
```

LoginScreen.js: Sends user and pass to server.

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Login screen. Sends username/password to server and logs in if valid.

import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Alert } from 'react-native';

export default function LoginScreen({ navigation, onLogin }) {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  // Called when user presses Login button
  const handleLogin = async () => {
    try {
      const response = await fetch('http://34.123.143.201/login_user.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password }),
      });

      const data = await response.json();

      // Login success
      if (data.status === 'success') {
        onLogin(data.username);
      } else {
        Alert.alert('Login Failed', data.message || 'Invalid credentials.');
      }
    } catch (err) {
      Alert.alert('Error', 'Could not connect to server.');
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>SkyView Login</Text>

      {/* Username input */}
      <TextInput
        placeholder="Username"
        value={username}
        onChangeText={setUsername}
        style={styles.input}
        autoCapitalize="none"
      />

      {/* Password input */}
      <TextInput
        placeholder="Password"
        value={password}
        onChangeText={setPassword}
        secureTextEntry
        style={styles.input}
      />

      {/* Login button */}
      <Button title="Login" onPress={handleLogin} />

      {/* Navigation link to registration screen */}
      <Text style={styles.link} onPress={() => navigation.navigate('Register')}>
        Don't have an account? Register
      </Text>
    </View>
  );
}

// Layout styles
const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: 'center', padding: 20 },
  title: { fontSize: 28, fontWeight: 'bold', marginBottom: 20, textAlign: 'center', color: '#007AFF' },
  input: { borderWidth: 1, padding: 10, marginBottom: 12, borderRadius: 6, borderColor: '#aaa' },
  link: { marginTop: 12, color: '#007AFF', textAlign: 'center' },
});
```

RegisterScreen.js: Allows registration.

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Allows users to register by sending username, email, and password to the server. Validates duplicate users.

import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Alert } from 'react-native';

export default function RegisterScreen({ navigation }) {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  // Called when Register button is pressed
  const handleRegister = async () => {
    // Check for missing fields
    if (!username || !email || !password) {
      Alert.alert('Missing Fields', 'Please fill in all fields.');
      return;
    }

    try {
      const response = await fetch('http://34.123.143.201/register_user.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, email, password }),
      });

      const data = await response.json();

      // If registration succeeded
      if (data.status === 'success') {
        Alert.alert('✅ Registration Successful', data.message);
        navigation.navigate('Login');
      } else {
        Alert.alert('❌ Registration Failed', data.message || 'Unknown error');
      }
    } catch (error) {
      Alert.alert('⚠️ Error', 'Could not connect to server.');
      console.error('Registration error:', error);
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>SkyView Register</Text>

      {/* Username input */}
      <TextInput
        placeholder="Username"
        value={username}
        onChangeText={setUsername}
        style={styles.input}
        autoCapitalize="none"
      />
```

```
      />

      {/* Email input */}
      <TextInput
        placeholder="Email"
        value={email}
        onChangeText={setEmail}
        style={styles.input}
        keyboardType="email-address"
        autoCapitalize="none"
      />

      {/* Password input */}
      <TextInput
        placeholder="Password"
        value={password}
        onChangeText={setPassword}
        secureTextEntry
        style={styles.input}
      />

      {/* Register button */}
      <Button title="Register" onPress={handleRegister} />

      {/* Navigation link to login screen */}
      <Text style={styles.link} onPress={() => navigation.navigate('Login')}>
        Already have an account? Login
      </Text>
    </View>
  );
}

// Layout styles
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    padding: 20,
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    marginBottom: 20,
    textAlign: 'center',
  },
  input: {
    borderWidth: 1,
    borderColor: '#aaa',
    padding: 10,
    marginBottom: 12,
    borderRadius: 6,
  },
```

AddScreen.js: Form to submit weather.

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Add Screen. Lets user add data to the database.

import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet } from 'react-native';
import LottieView from 'lottie-react-native';

export default function AddScreen() {
  // State to store form input values
  const [city, setCity] = useState('');
  const [temperature, setTemp] = useState('');
  const [humidity, setHumidity] = useState('');

  // Submits weather data to backend
  const handleSubmit = async () => {
    try {
      await fetch('http://34.123.143.201/add_weather.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ city, temperature, humidity }),
      });

      alert('✅ Weather report added!');
      setCity('');
      setTemp('');
      setHumidity('');
    } catch (err) {
      alert("⚠️ Failed to add weather. Check connection.");
    }
  };

  return (
    <View style={styles.container}>
      {/* Floating sun animation */}
      <View style={styles.lottieWrapper}>
        <LottieView
          source={require('../assets/animations/sun.json')}
          autoPlay
          loop
          style={styles.lottie}
        />
      </View>

      {/* Input fields and submit button */}
      <View style={styles.content}>
        <Text style={styles.appTitle}>SkyView</Text>
        <Text style={styles.header}>Add Weather Report</Text>

        <TextInput placeholder="City" value={city} onChangeText={setCity} style={styles.input} />
        <TextInput placeholder="Temperature (°F)" value={temperature} onChangeText={setTemp} keyboardType="numeric" style={styles.input} />
        <TextInput placeholder="Humidity (%)" value={humidity} onChangeText={setHumidity} keyboardType="numeric" style={styles.input} />

        <Button title="Submit" onPress={handleSubmit} />
      </View>
```

```
    </View>
  );
}

// Styles
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#fff',
  },
  lottieWrapper: {
    position: 'absolute',
    top: 40,
    zIndex: 1,
    alignItems: 'center',
    width: '100%',
  },
  lottie: {
    width: 100,
    height: 100,
  },
  content: {
    alignItems: 'center',
    width: '100%',
    paddingHorizontal: 16,
    maxWidth: 400,
  },
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    textAlign: 'center',
    marginBottom: 10,
  },
  header: {
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },
  input: {
    borderWidth: 1,
    borderColor: '#ccc',
    marginBottom: 12,
    padding: 10,
    borderRadius: 5,
    width: 300,
  },
});
```

SearchScreen.js: Uses FlatList to show search results.

```
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Search screen. Lets user search a city and retrieve data for it.

import React, { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  Button,
  FlatList,
  StyleSheet,
  TouchableWithoutFeedback,
  Keyboard,
} from 'react-native';
import LottieView from 'lottie-react-native';

const SearchScreen = () => {
  const [searchTerm, setSearchTerm] = useState('');
  const [results, setResults] = useState([]);

  // Fetch weather data from the server for a given city
  const handleSearch = async () => {
    if (!searchTerm.trim()) {
      alert('Please enter a city name.');
      return;
    }

    try {
      const response = await fetch(`http://34.123.143.201/search_weather.php?city=${searchTerm}`);
      const data = await response.json();
      setResults(data); // store result in state
    } catch (error) {
      console.error('Error fetching weather data:', error);
      setResults([]); // clear results if error
    }
  };

  return (
    <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
      <View style={styles.container}>
        {/* Floating sun animation */}
        <View style={styles.lottieWrapper}>
          <LottieView
            source={require('../assets/animations/sun.json')}
            autoPlay
            loop
            style={styles.lottie}
          />
        </View>

        {/* Title and input */}
        <View style={styles.centeredContent}>
          <Text style={styles.appTitle}>SkyView</Text>
          <Text style={styles.title}>Search Weather</Text>
```

```
          <TextInput
            style={styles.input}
            placeholder="Enter city name"
            value={searchTerm}
            onChangeText={setSearchTerm}
          />
          <Button title="Search" onPress={handleSearch} />
        </View>

        {/* Results output */}
        <View style={styles.resultsSection}>
          {results.length > 0 ? (
            <FlatList
              data={results}
              keyExtractor={(item, index) => index.toString()}
              keyboardShouldPersistTaps="handled"
              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>
              )}
            />
          ) : (
            searchTerm !== '' && (
              <Text style={styles.noMatch}>No match found for “{searchTerm}”</Text>
            )
          )}
        </View>
      </View>
    </TouchableWithoutFeedback>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 16,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff',
  },
  lottieWrapper: {
    position: 'absolute',
    top: 40,
    zIndex: 1,
    alignItems: 'center',
    width: '100%',
  },
  lottie: {
    width: 100,
    height: 100,
  },
  centeredContent: {
    alignItems: 'center',
    justifyContent: 'center',
    marginBottom: 20,
    width: '100%',
    maxWidth: 400,
  },
  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: 180, // fixed input width
  },
  resultsSection: {
    width: '100%',
    maxWidth: 400,
    flexShrink: 1,
  },
  resultBox: {
    backgroundColor: '#f0f0f0',
    padding: 10,
    borderRadius: 6,
    marginBottom: 12,
  },
  city: {
    fontSize: 18,
    fontWeight: 'bold',
  },
  noMatch: {
    marginTop: 20,
    fontStyle: 'italic',
    color: 'gray',
    textAlign: 'center',
  },
});

export default SearchScreen;
```

DisplayScreen.js: Select and display data from database.

```
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Select and display data from any database table using dropdown picker and restored flat list layout.

import React, { useState, useEffect } from 'react';
import { View, Text, FlatList, StyleSheet, ActivityIndicator, Pressable } from 'react-native';
import DropDownPicker from 'react-native-dropdown-picker';

export default function DisplayScreen() {
  const [tab, setTab] = useState('Display'); // current tab
  const [tables, setTables] = useState([]); // available table names
  const [selectedTable, setSelectedTable] = useState(''); // current table
  const [data, setData] = useState([]); // rows from table
  const [loading, setLoading] = useState(false); // loading state

  // dropdown state
  const [open, setOpen] = useState(false);
  const [items, setItems] = useState([]); // dropdown options

  // Load table names
  const fetchTables = async () => {
    try {
      const res = await fetch('http://34.123.143.201/get_table_list.php');
      const json = await res.json();
      setTables(json);
      setItems(json.map(tbl => ({ label: tbl, value: tbl })));
      setSelectedTable(json[0]);
    } catch (err) {
      console.error('Failed to load table list:', err);
    }
  };

  // Load data from selected table
  const fetchData = async () => {
    if (!selectedTable) return;
    try {
      setLoading(true);
      const res = await fetch(`http://34.123.143.201/get_table_data.php?table=${selectedTable}`);
      const json = await res.json();
      if (json.rows) setData(json.rows);
    } catch (err) {
      console.error('Data fetch failed:', err);
    } finally {
      setLoading(false);
    }
  };

  // On component mount: get tables
  useEffect(() => {
    fetchTables();
  }, []);
```

```
// On table change or tab switch to "Display": load data
useEffect(() => {
  if (tab === 'Display' && selectedTable) {
    fetchData();
  }
}, [tab, selectedTable]);

return (
  <View style={styles.container}>
    {/* Tab bar to switch between views */}
    <View style={styles.tabContainer}>
      <Pressable onPress={() => setTab('Select')}>
        <Text style={[styles.tab, tab === 'Select' && styles.activeTab]}>Select</Text>
      </Pressable>
      <Pressable onPress={() => setTab('Display')}>
        <Text style={[styles.tab, tab === 'Display' && styles.activeTab]}>Display</Text>
      </Pressable>
    </View>

    {/* Table selector dropdown */}
    {tab === 'Select' && (
      <View style={styles.dropdownWrapper}>
        <Text style={styles.label}>Choose a table:</Text>
        <DropDownPicker
          open={open}
          setOpen={setOpen}
          items={items}
          setItems={setItems}
          value={selectedTable}
          setValue={setSelectedTable}
          placeholder="Select a table"
          style={styles.dropdown}
          dropDownContainerStyle={styles.dropdownBox}
        />
      </View>
    )}

    {/* Display fetched rows */}
    {tab === 'Display' && (
      <>
        <Text style={styles.selected}>Showing: {selectedTable}</Text>
        {loading ? (
          <ActivityIndicator size="large" color="#007AFF" />
        ) : (
          <FlatList
            data={data}
            keyExtractor={(item, index) => index.toString()}
            contentContainerStyle={styles.list}
            renderItem={({ item }) => (
              <View style={styles.row}>
                {Object.entries(item).map(([key, value]) => (
                  <Text key={key} style={styles.item}>
                    <Text style={styles.bold}>{key}:</Text> {String(value)}
                  </Text>
                ))}
              </View>
            )}
```

```
            ListEmptyComponent={<Text style={styles.empty}>No data found.</Text>}
          />
        )}
      </>
    )}
  </View>
 );
}

// Styles
const styles = StyleSheet.create({
 container: {
  flex: 1,
  paddingTop: 40,
  paddingHorizontal: 16,
  backgroundColor: '#fff',
 },
 tabContainer: {
  flexDirection: 'row',
  justifyContent: 'center',
  marginBottom: 16,
 },
 tab: {
  fontSize: 16,
  padding: 10,
  marginHorizontal: 12,
  color: '#444',
 },
 activeTab: {
  borderBottomWidth: 2,
  borderColor: '#007AFF',
  fontWeight: 'bold',
  color: '#007AFF',
 },
 dropdownWrapper: {
  marginTop: 20,
  zIndex: 999,
 },
 label: {
  fontSize: 18,
  marginBottom: 10,
  textAlign: 'center',
 },
 dropdown: {
  backgroundColor: '#f1f1f1',
 },
 dropdownBox: {
  backgroundColor: '#f1f1f1',
  zIndex: 999,
 },
 selected: {
  fontSize: 16,
  marginBottom: 10,
  textAlign: 'center',
 },
 list: {
  paddingBottom: 100,
 },
 row: {
  backgroundColor: '#f5f5f5',
  marginBottom: 10,
  padding: 10,
  borderRadius: 8,
 },
 item: {
  fontSize: 14,
  marginBottom: 4,
 },
 bold: {
  fontWeight: 'bold',
 },
 empty: {
  textAlign: 'center',
  fontStyle: 'italic',
  marginTop: 20,
  color: '#888',
 },
});
```

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

```
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Select and display data from any database table using dropdown picker and restored flat list layout.

import React, { useState, useEffect } from 'react';
import { View, Text, FlatList, StyleSheet, ActivityIndicator, Pressable } from 'react-native';
import DropDownPicker from 'react-native-dropdown-picker';

export default function DisplayScreen() {
  const [tab, setTab] = useState('Display'); // current tab
  const [tables, setTables] = useState([]); // available table names
  const [selectedTable, setSelectedTable] = useState(''); // current table
  const [data, setData] = useState([]); // rows from table
  const [loading, setLoading] = useState(false); // loading state

  // dropdown state
  const [open, setOpen] = useState(false);
  const [items, setItems] = useState([]); // dropdown options

  // Load table names
  const fetchTables = async () => {
    try {
      const res = await fetch('http://34.123.143.201/get_table_list.php');
      const json = await res.json();
      setTables(json);
      setItems(json.map(tbl => ({ label: tbl, value: tbl })));
      setSelectedTable(json[0]);
    } catch (err) {
      console.error('Failed to load table list:', err);
    }
  };

  // Load data from selected table
  const fetchData = async () => {
    if (!selectedTable) return;
    try {
      setLoading(true);
      const res = await fetch(`http://34.123.143.201/get_table_data.php?table=${selectedTable}`);
      const json = await res.json();
      if (json.rows) setData(json.rows);
    } catch (err) {
      console.error('Data fetch failed:', err);
    } finally {
      setLoading(false);
    }
  };

  // On component mount: get tables
  useEffect(() => {
    fetchTables();
  }, []);
```

```
        </MapView>

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

// Styling for layout and map
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  map: {
    width: Dimensions.get('window').width,
    height: Dimensions.get('window').height,
  },
  searchBar: {
    position: 'absolute',
    top: 40,
    left: 20,
    right: 20,
    flexDirection: 'row',
    backgroundColor: '#fff',
    borderRadius: 8,
    padding: 8,

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

    // Android shadow
    elevation: 4,
  },
  input: {
    flex: 1,
    paddingHorizontal: 10,
  },
});
```

FeedbackScreen.js: Like/dislike UI and submits.

```
// Author: Thomas Hoerger
// Group 9
// 4-24-2025
// Description: Feedback screen. Lets user submit feedback to the database.

import React, { useState } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';
import LottieView from 'lottie-react-native'; // 🌞 Lottie animation

export default function FeedbackScreen() {
  const [submitted, setSubmitted] = useState(false); // tracks if feedback is sent

  // Sends feedback to the backend
  const handleFeedback = async (liked) => {
    try {
      await fetch('http://34.123.143.201/submit_feedback.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          username: 'guest_user',
          liked: liked ? 1 : 0, // send 1 for like, 0 for dislike
        }),
      });

      setSubmitted(true); // flag as submitted
    } catch (error) {
      alert('Failed to submit feedback.');
    }
  };

  return (
    <View style={styles.container}>
      {/* Floating sun animation at top */}
      <View style={styles.lottieWrapper}>
        <LottieView
          source={require('../assets/animations/sun.json')}
          autoPlay
          loop
          style={styles.lottie}
        />
      </View>

      {/* Content block */}
      <View style={styles.content}>
        <Text style={styles.appTitle}>SkyView</Text>

        {/* Conditional feedback buttons or thank-you message */}
        <View style={styles.feedbackBlock}>
          {!submitted ? (
            <>
              <Text style={styles.question}>Do you like this app?</Text>
              <Button title="👍 I Like It" onPress={() => handleFeedback(true)} />
              <View style={{ height: 10 }} />
              <Button title="👎 I Don't Like It" onPress={() => handleFeedback(false)} />
            </>
          ) : (
            <Text style={styles.thanks}>Thanks for your feedback! ✅</Text>
          )}
        </View>
      </View>
    </View>
  );
}

// Layout styling
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingHorizontal: 16,
    backgroundColor: '#fff',
  },
  lottieWrapper: {
    position: 'absolute',
    top: 40,
    zIndex: 1,
    alignItems: 'center',
    width: '100%',
  },
  lottie: {
    width: 100,
    height: 100,
  },
  content: {
    alignItems: 'center',
    justifyContent: 'center',
    width: '100%',
    maxWidth: 400,
    paddingHorizontal: 16,
  },
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    marginBottom: 20,
    textAlign: 'center',
  },
  question: {
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },
  feedbackBlock: {
    height: 140, // fixed block height for consistent layout
    justifyContent: 'center',
    alignItems: 'center',
  },
```

```
  thanks: {
    fontSize: 18,
    color: 'green',
    textAlign: 'center',
  },
});
```

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

```
// Author: Thomas Hoerger
// Group 9
// 4-23-2025
// Description: Displays user feedback as a pie chart using react-native-chart-kit.

import React, { useEffect, useState } from 'react';
import { View, Text, Dimensions, StyleSheet, Alert, Button } from 'react-native';
import { PieChart } from 'react-native-chart-kit';
import LottieView from 'lottie-react-native';

// Functional component to render the chart screen
export default function ChartScreen() {
  const [chartData, setChartData] = useState(null); // State to hold pie chart data

  // Function to fetch feedback stats from the backend
  const loadChartData = () => {
    fetch('http://34.123.143.201/get_feedback_stats.php') // Fetch data from PHP endpoint
      .then((res) => res.json()) // Parse JSON response
      .then((data) => {
        const liked = parseInt(data.liked);       // Parse liked count as integer
        const disliked = parseInt(data.disliked); // Parse disliked count as integer

        // Validate data
        if (isNaN(liked) || isNaN(disliked)) {
          Alert.alert('Error', 'Invalid feedback data');
          return;
        }

        // Set pie chart data
        setChartData([
          {
            name: 'Liked',
            population: liked,
            color: 'green',
            legendFontColor: '#000',
            legendFontSize: 14,
          },
          {
            name: 'Disliked',
            population: disliked,
            color: 'red',
            legendFontColor: '#000',
            legendFontSize: 14,
          },
        ]);
      })
      .catch((err) => {
        console.error('Chart fetch error:', err);
        Alert.alert('Error', 'Could not load feedback data');
      });
  };

  // Load chart data once when the component mounts
  useEffect(() => {
    loadChartData();
  }, []);
```

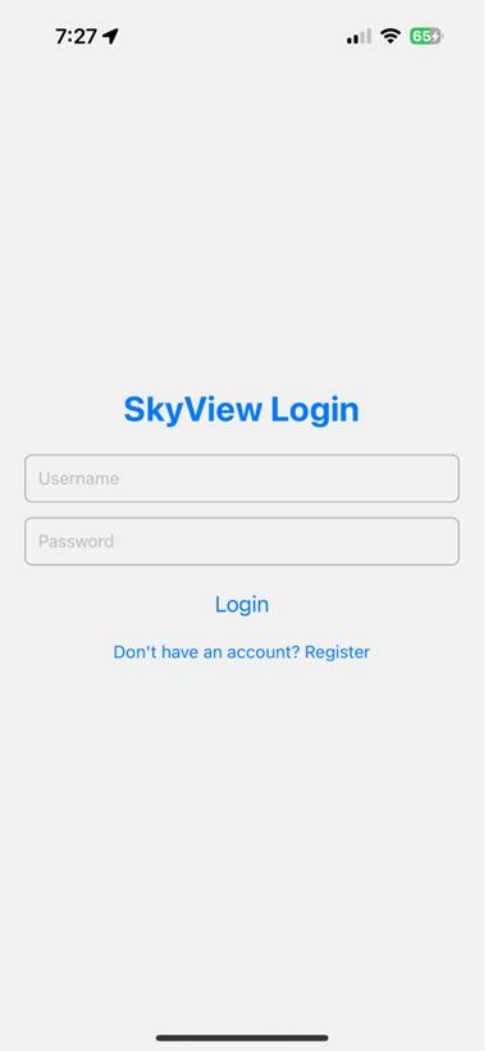
```
  return (
    <View style={styles.container}>
      {/* Lottie animation displayed above the chart */}
      <View style={styles.lottieWrapper}>
        <LottieView
          source={require('../assets/animations/sun.json')}
          autoPlay
          loop
          style={styles.lottie}
        />
      </View>

      {/* App title and screen title */}
      <Text style={styles.appTitle}>SkyView</Text>
      <Text style={styles.title}>User Feedback</Text>

      {/* Refresh button to reload chart data */}
      <View style={{ marginBottom: 20 }}>
        <Button title="Refresh Chart" onPress={loadChartData} />
      </View>

      {/* Display pie chart if data is available, else show loading message */}
      {chartData ? (
        <PieChart
          data={chartData}
          width={Dimensions.get('window').width - 20}
          height={220}
          chartConfig={{
            backgroundColor: '#fff',
            backgroundGradientFrom: '#fff',
            backgroundGradientTo: '#fff',
            color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
            labelColor: () => '#000',
          }}
          accessor="population"
          backgroundColor="transparent"
          paddingLeft="15"
          absolute
        />
      ) : (
        <Text style={styles.loading}>Loading chart...</Text>
      )}
    </View>
  );
}

// Styles for the chart screen layout and elements
const styles = StyleSheet.create({
  container: {
    marginTop: 40,
    alignItems: 'center',
    paddingHorizontal: 10,
  },
```

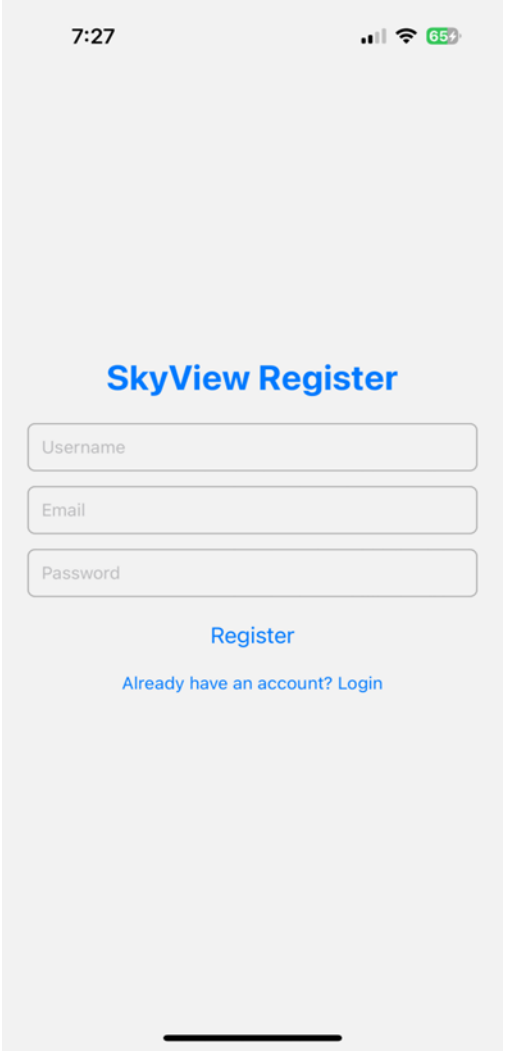
```
  lottieWrapper: {
    position: 'absolute',
    top: -160,
    zIndex: 1,
    width: '100%',
    alignItems: 'center',
  },
  lottie: {
    width: 100,
    height: 100,
  },
  appTitle: {
    fontSize: 28,
    fontWeight: 'bold',
    color: '#007AFF',
    marginBottom: 10,
    textAlign: 'center',
  },
  title: {
    fontSize: 20,
    fontWeight: 'bold',
    marginBottom: 10,
  },
  loading: {
    fontStyle: 'italic',
    color: '#555',
  },
});
```

Included Features:

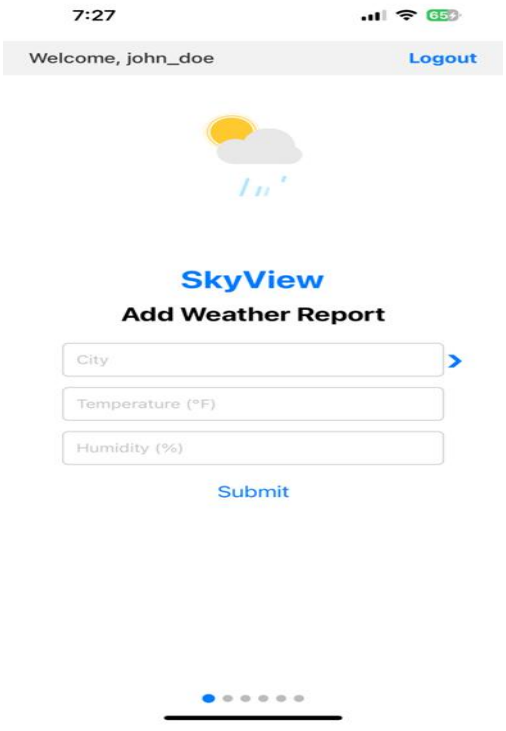
- Installed react-native-swiper for navigation.
- Added custom dot and arrow indicators.

- Login/Register Screen

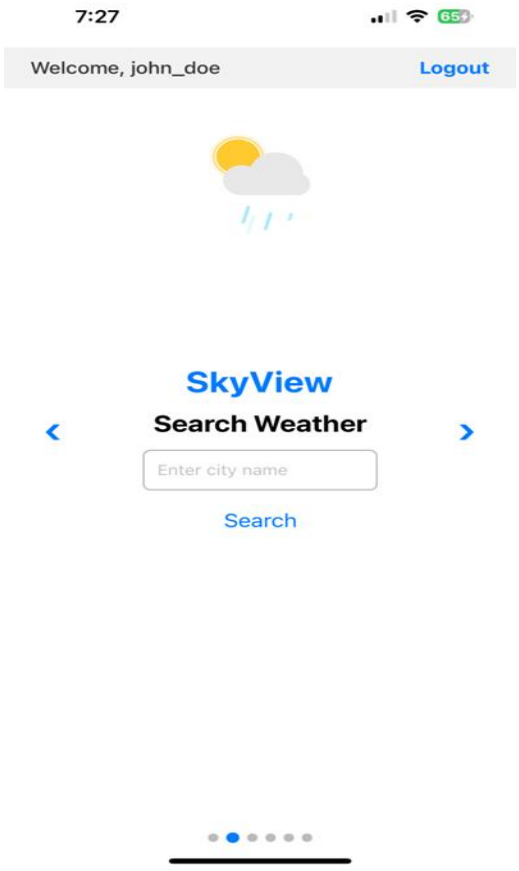




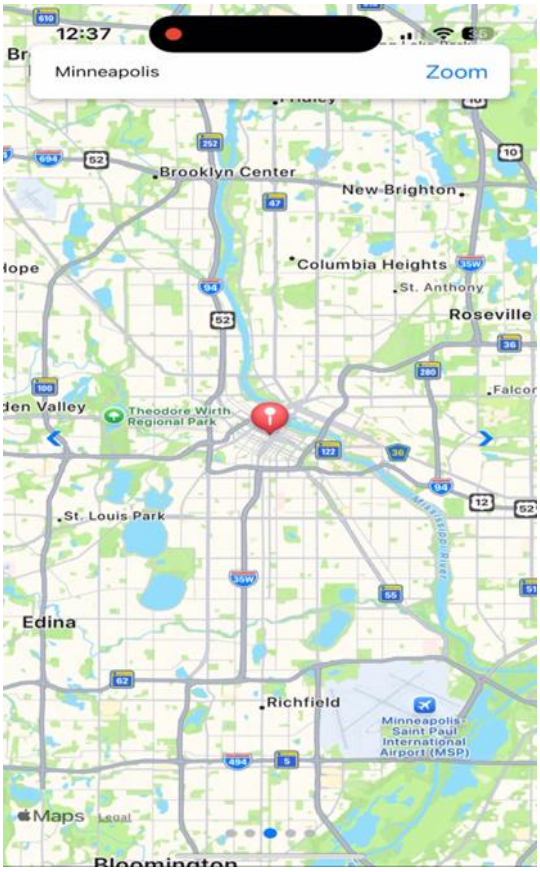
- Implemented add feature so that the user can add to the database via the app.



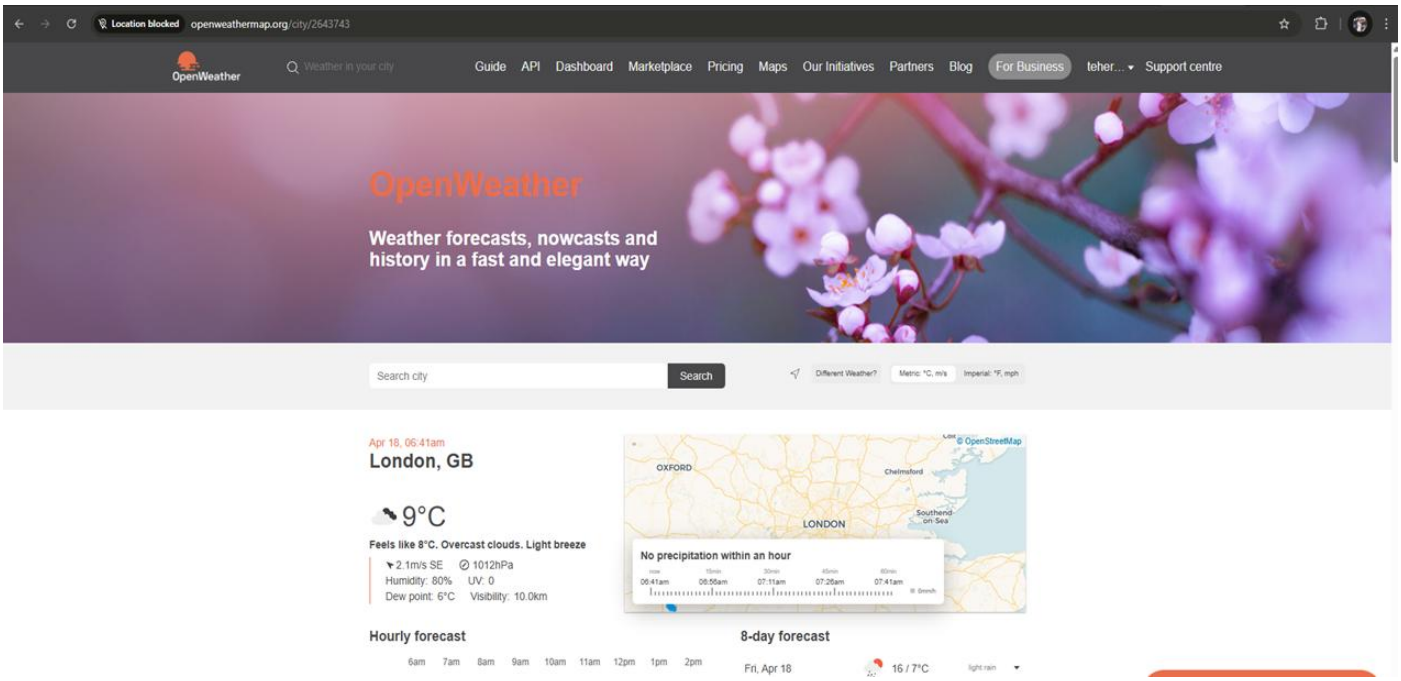
- Added search so that the user can display weather reports from the searched city.



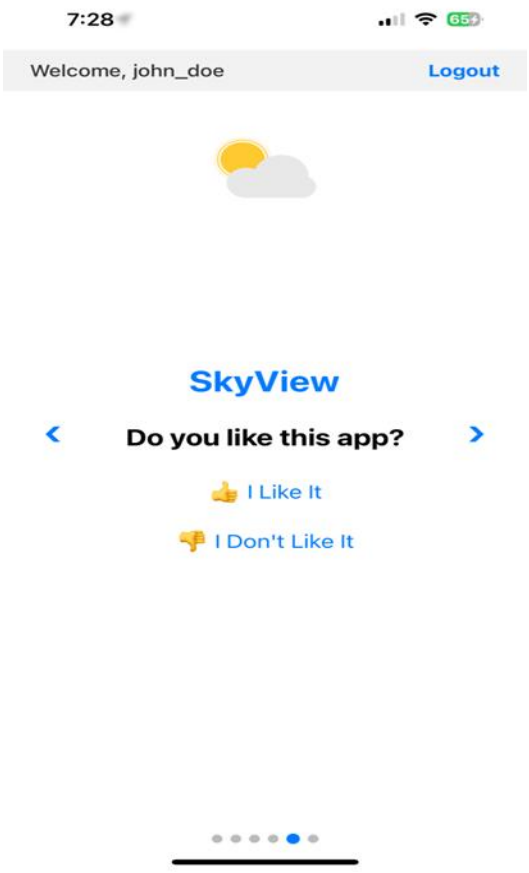
- Added map feature so that the user can search a city and display it on a map.

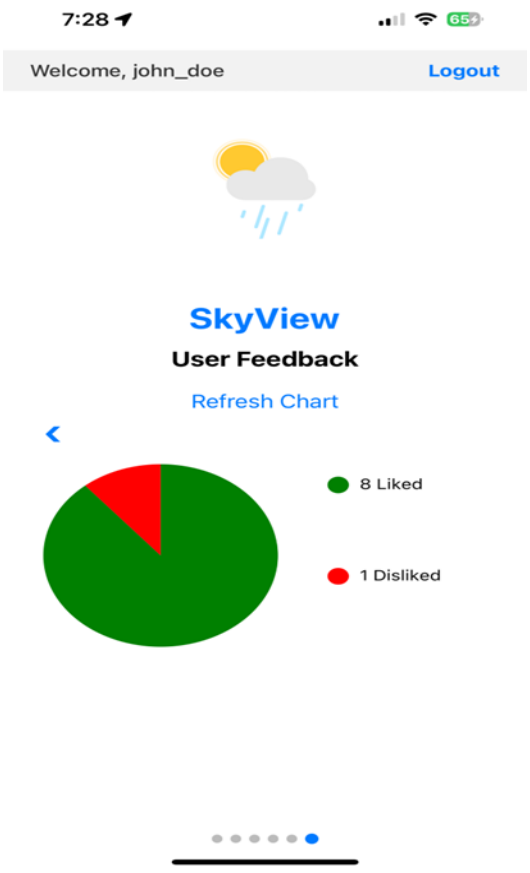


- Integrated OpenWeatherMap API (Geo API) into get_coords.php. So if the city isn't in the database, script fetches coordinates live from API

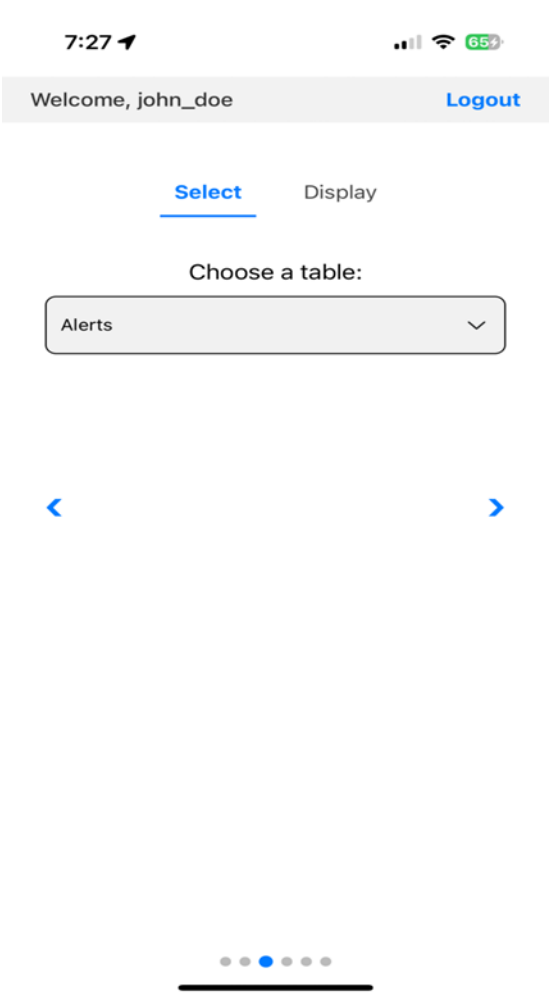


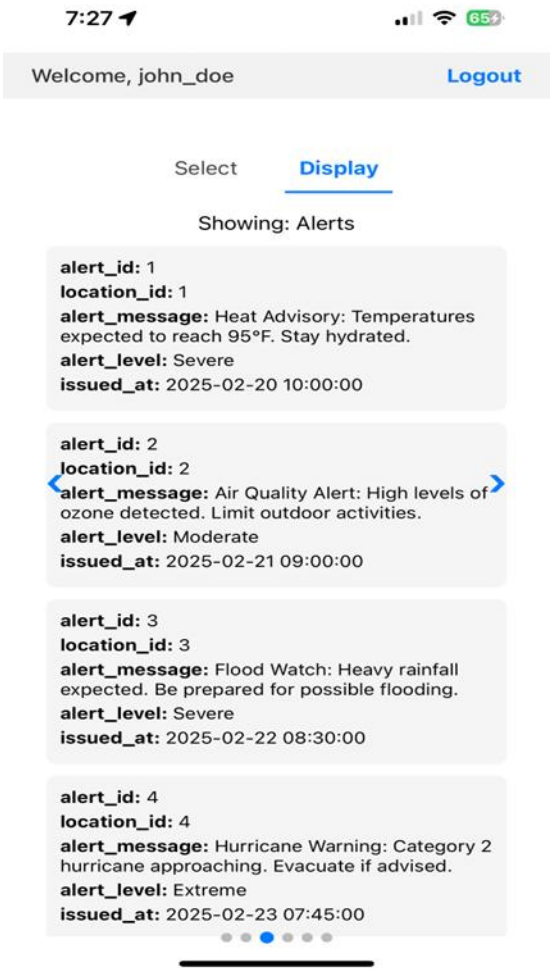
- Feedback + Chart System
  - Users select like/dislike and submit.
  - Chart displays live data from database using a refresh button.





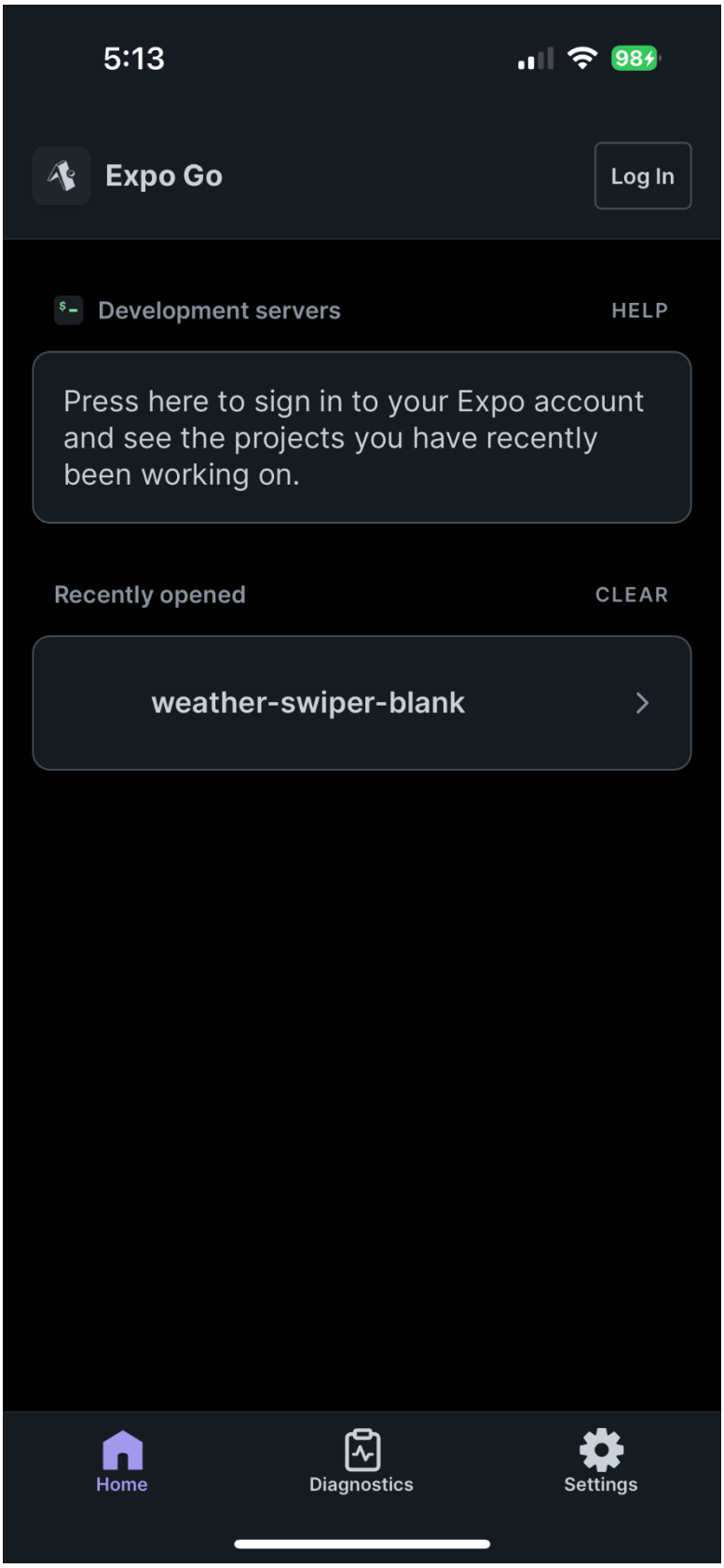
- Display Screen: User can select which table from the database they would like and display it





- Used responsive styles to support both mobile and laptop views.
- Components such as maps, charts, and form fields scale based on screen size.
- Added Lottie animation
- Added welcome (user) and logout button

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



Summary:

The updated SkyView app is a full-stack, swipe-based weather platform that allows users to register, log in, and interact with real-time weather data. Built with React Native and connected to a PHP/MySQL backend hosted on a VPS, the app lets users add and search city-specific reports, select/display database tables, view map locations with dynamic zoom, and submit feedback visualized in a live pie chart. Lottie animations enhance the UI across screens, and the system supports real-time data refresh and table browsing. This project showcases my ability to develop and deploy a complete, secure, and animated mobile application from backend to frontend.