**Capstone Project Documentation**  
  
GITHUB REPOSITORY -<https://github.com/adetayo10/capstone-project-team2>

**Project Overview**

The Capstone Weather App is a full-stack web application that provides real-time weather updates. Users can sign up, log in, and view weather conditions for different cities. This project integrates frontend, backend, and database management, utilizing an external weather API to fetch weather data.

## Day 1 - Capstone

**1. Understanding the Capstone Plan**

The Capstone project introduces real-world development practices, requiring the team to work collaboratively to build an application from scratch. The project follows a structured plan where team members assume different roles, complete assigned tasks, and ensure that each step builds upon the previous one.

**2. Project Introduction: The Weather App**

The **Capstone Weather App** is designed to provide real-time weather updates for selected cities. It incorporates authentication, weather data retrieval, and historical tracking. The project is divided into several stages, starting with database modeling and backend development, then moving towards frontend integration and testing.

**3. Functional and Non-Functional Requirements**

**Functional Requirements:**

* User authentication (sign-up, login, email verification, password reset, logout)
* Ability to search for weather data by city
* Display weather conditions such as temperature, humidity, and wind speed
* Store and retrieve historical weather searches

**Non-Functional Requirements:**

* Fast response times with minimal latency
* Secure authentication with token-based authorization
* High availability and cloud-based data storage
* Efficient historical data retrieval (limited to 10 recent searches)

**4. Database Design with PostgreSQL**

The database structure was designed with the following tables:

* **User Table**: Stores user credentials and authentication details.
* **Country Table**: Stores country codes and country information.
* **City Table**: Stores details about cities including geographical data.
* **Weather Table**: Stores weather conditions linked to users and cities.

The data is hosted on a **cloud instance**, ensuring accessibility and performance.

**5. Entity Relationship Diagram (ERD)**

The ERD was developed to illustrate how the different database tables interact:

* One-to-many relationship between **City** and **Weather**.
* Many-to-one relationship between **City** and **Country**.
* One-to-many relationship between **User** and **Weather**.

**6. Connecting pgAdmin to Cloud Database**

To ensure seamless data access, we connected **pgAdmin** to the cloud-hosted PostgreSQL database using unique credentials. The setup was tested for accessibility, confirming that all tables and relationships were correctly established.

**7. Running Additional Queries**

After creating the tables, we executed several SQL queries to:

* Insert sample data into the **User, Country, City, and Weather** tables.
* Retrieve specific records based on conditions (e.g., filtering verified users, searching by city ID).
* Update existing records to reflect changes in weather data or user details.
* Delete specific entries to test data integrity and constraints.

### Summary of Day 1

* **Capstone Plan Discussion**: Understanding project structure and team collaboration.
* **Weather App Introduction**: Identified functional and non-functional requirements.
* **Database Design**: Created User, Country, City, and Weather tables.
* **Entity Relationship Diagram (ERD)**: Mapped out database relationships.
* **Cloud Database Setup**: Connected pgAdmin to a cloud-hosted PostgreSQL instance.
* **Query Execution**: Inserted, updated, retrieved, and deleted records to test database operations.

## Day 2 - Capstone

**1. Performing SQL Join Queries**

We executed multiple **JOIN** queries to extract useful information:

* Retrieved city names and their corresponding weather descriptions.
* Retrieved city names and temperatures.
* Retrieved city names and timezones where humidity is greater than 50%.
* Retrieved city names and sunrise times where temperature is below 20°C.
* Retrieved city names and maximum temperatures where wind speed is less than 10.

**2. Creating a Stored Procedure**

We created a **stored procedure (insertUser)** to insert new users dynamically into the **User** table.

**3. Creating a View**

A **database view (WeatherView)** was implemented to streamline weather data retrieval by joining **City**, **Country**, and **Weather** tables.

**4. Creating a Function**

A function **totalRecordsInUser** was developed to return the total number of records in the **User** table.

**5. Implementing Bulk Insertion**

A **stored procedure (insertBulkUser)** was created to insert **50** random users into the **User** table for testing purposes.

**6. Running Additional Queries**

We executed SQL queries to:

* Filter users by first name, last name, email verification, and phone number patterns.
* Count and group users based on the month of creation.
* Use **subqueries** to retrieve user details dynamically.

**7. Java Integration with PostgreSQL**

We developed a **Java program** that:

* Establishes a database connection to PostgreSQL.
* Inserts new users into the **User** table dynamically.
* Retrieves user data and performs queries using **JDBC**.

### Summary of Day 2

* **Executed SQL JOIN queries** to retrieve weather and city data.
* **Created a stored procedure (insertUser)** for dynamic user insertion.
* **Implemented a database view (WeatherView)** for efficient weather queries.
* **Developed a function (totalRecordsInUser)** to count records in the User table.
* **Performed bulk data insertion** with the stored procedure (insertBulkUser).
* **Executed advanced SQL queries** including filtering, aggregation, and subqueries.
* **Integrated Java with PostgreSQL** to automate database interactions using JDBC..