SOFTWARE REQUIREMENTS SPECIFICATION

**For**

**Weather App Using DB**

**Prepared by:-**

*1.22EC140-****SANJEEV.RM***

*2.22EC175-****SUMAN.T***

*3.2EC307-****VIJAY ANAND.A***

# Introduction

Creating a weather app using an API and a database involves fetching weather data from a third-party API, storing it in a database, and then presenting it to users through a user interface

## Purpose

## Document Conventions

* + - Entire document should be justified.
    - Convention for Main title

Font face: Times New Roman Font style: Bold

Font Size: 14

* + - Convention for Sub title

Font face: Times New Roman Font style: Bold

Font Size: 12

* + - Convention for body

Font face: Times New Roman Font Size: 12

## Scope of Development Project

### The scope of a weather app using an API integrated with a database can be extensive, providing a variety of features and functionalities to meet the needs of different user groups.

**Scope and features of weather app:**

**Current Weather Information**

**Extended Forecast**

**Historical Weather Data**

**Location-based Services**

**Weather Alerts and Notifications**

## Definitions, Acronyms and Abbreviations

JAVA -> platform independence SQL-> Structured query Language ER-> Entity Relationship

UML -> Unified Modeling Language

IDE-> Integrated Development Environment SRS-> Software Requirement Specification

## References

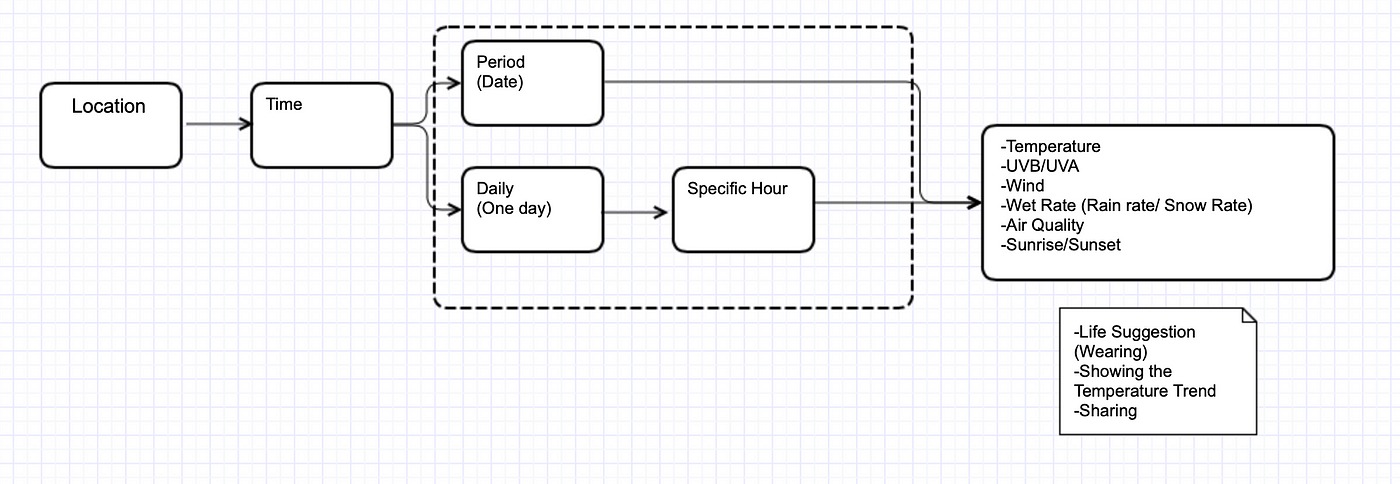
* + - Books

 Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices (ACM Press) by Michael Jackson

Software Requirements (Microsoft) Second EditionBy Karl E. Wiegers

Software Engineering: A Practitioner’s Approach Fifth Edition By Roger S. Pressman

1. **2.overall descriptions**
2. **2.1 product perpective**

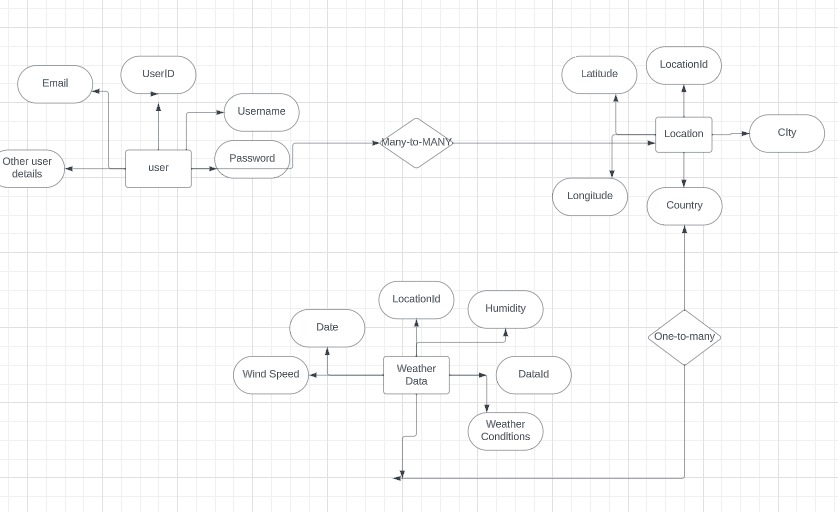


This is a broad level diagram of the project showing a basic overview. The users can be either staff or student.. This System will provide a search functionality to facilitate the search of resources.

The users of the system can request the weather/temperature/humitity for which they would have to follow certain criteria.

## Product Function

Entity Relationship Diagram of Library Management System



## User Classes and Characteristics

## 

## To design a weather app with a database, you can define several user classes and their characteristics. User classes represent different types of users who will interact with your weather app. Here are some common user classes and their characteristics for a weather app:

## Guest Users:

## Characteristics:

## Access the app without creating an account.

## Limited features and personalization.

## Basic weather information for a default location.

## Registered Users:

## Characteristics:

## Create an account with a unique username and password.

## Save favorite locations for quick access.

## Set default location.

## Receive personalized weather notifications.

## Premium Users:

## Characteristics:

## Subscribe to a premium plan.

## Access advanced features such as extended forecasts, historical weather data, and radar imagery.

## Ad-free experience.

## Priority customer support.

## Business Users:

## Characteristics:

## Access to weather data for business planning and decision-making.

## Integration with other business applications.

## Operating Environment:

## The operating environment for weather app development using a database involves various components, technologies, and considerations. Here's an overview of the key elements in the operating environment:

## Front-end Development:

## Frameworks and Libraries: Choose front-end frameworks like React, Angular, or Vue.js for building the user interface. Utilize libraries such as D3.js or Chart.js for data visualization.

## Responsive Design: Ensure the app is responsive and accessible across various devices and screen sizes.

## Back-end Development:

## Server-Side Language: Select a server-side language such as Node.js, Python (Django or Flask), Ruby (Ruby on Rails), or Java for handling server-side logic.

## Web Framework: Use a web framework that facilitates server-side development, routing, and handling requests.

## Assumptions and Dependencies:

The assumptions are:-

**Data Accuracy:**

Assumption: Weather data obtained from external APIs or sources is assumed to be accurate and reliable.

Rationale: The app relies on external weather services, and any inaccuracies in their data may affect the app's accuracy.

**User Internet Connection:**

Assumption: Users are assumed to have a stable internet connection for real-time weather updates.

Rationale: The app fetches live weather data from online sources, and a reliable internet connection is necessary for timely updates.

**Device Compatibility:**

Assumption: The app is assumed to be used on devices with modern web browsers or specific mobile platforms.

**Dependencies:**

Weather APIs:

Dependency: The app depends on external weather APIs to fetch current and forecasted weather data.

Mitigation: Regularly monitor the performance of chosen APIs and have contingency plans if there are disruptions.

Database System:

Dependency: The app relies on a specific database system (e.g., MySQL, PostgreSQL) to store user data and preferences.

Mitigation: Ensure proper configuration, backups, and scalability of the chosen database system.

Internet Connectivity:

Dependency: The app depends on users having a stable internet connection to fetch real-time weather updates.

Mitigation: Implement offline capabilities for cached data and provide user-friendly notifications for connectivity issues.

## Requirement

**Software Configuration:-**

Integrated Development Environment (IDE): Use a suitable IDE for coding, such as Visual Studio Code, PyCharm, or Atom.

Version Control: Implement version control using Git for collaborative development and code management.

Database Selection: Choose a database system based on your needs (e.g., MySQL, PostgreSQL, MongoDB for NoSQL).

Database Connectivity: Implement a connection layer to interact with the chosen DBMS.

**Hardware Configuration:-**

Development Workstations:

High-performance CPUs (e.g., multi-core processors).

Sufficient RAM (8GB or more).

SSD storage for faster development and compilation.

## Data Requirement

Geolocation Data:

**Latitude and Longitude:**

Store the geographical coordinates (latitude and longitude) for various locations.

Location Names:

Maintain a database of location names for user-friendly display.

Weather Data:

**Current Conditions:**

Temperature, humidity, wind speed, wind direction, visibility, and other relevant parameters.

**Forecast Data:**

Hourly and daily forecasts for a specified period (e.g., 7-day forecast).

Historical Weather Data:

Archive past weather conditions for users who want to view historical data.

# 3.External Interface Requirement

## GUI

User Interface Design Principles: The Weather App With Database will present users with a visually appealing and intuitive graphical interface. The design adheres to standard principles, ensuring an enhanced user experience and easy navigation.

**Customization:**

Users will have the flexibility to customize the interface according to their preferences.

**Modules Integration:**

All modules within the software, such as user management, weather information display, and date selection, seamlessly integrate into the graphical user interface.

**Login Interface:**

Users are required to log in using a username and password. New users can register by entering their details, and incorrect login attempts trigger error messages.

**Search Feature:**

Users can search for weather information by entering the name of the city and date.

**Categories View:**

The app displays categories of weather information, including Temperature, Humidity, Pressure, Visibility, Wind, DewPoint, UV Index, Moon Phase, and Air Quality.

# 4.System Features

# 

The Weather App With Database focuses on user account security and delivering real-time weather updates with the following features:

User Authentication and Validation: Users securely access the app with unique credentials and member ID verification, ensuring controlled and verified access.

Monitoring and Weather Alerts: The system actively monitors user activity and provides real-time weather alerts. Users receive pop-up notifications for imminent weather changes or events, such as rain, allowing them to stay informed about the current conditions.

Real-time Weather Display: The app prioritizes real-time weather data, offering users the latest and most accurate information about current weather conditions.

Accountability and Privacy: The system maintains proper accountability by limiting access to individual accounts. Users can securely view and manage their real-time weather information, ensuring privacy and data security. The absence of an administrator role contributes to a straightforward and secure user experience

**5.1** **Performance Requirement:**

Performance requirements for a weather app with a database depend on various factors, including the scale of the application, the number of users, the frequency of data updates, and the complexity of queries. Here are some general performance considerations for a weather app with a database:

**Scalability:**

Plan for scalability to handle an increasing number of users and data.

Use horizontal scaling by distributing the workload across multiple servers or cloud instances.

Database Indexing:

Implement proper indexing on the fields used in queries (e.g., city names) to improve search performance.

Caching:

Implement caching mechanisms to store frequently accessed data in memory. This can reduce the load on the database, especially for read-heavy operations.

Asynchronous Processing:

Consider using asynchronous processing for tasks such as data updates and background

## Safety Requirement

## 

## 

## When it comes to the security requirements for a weather app's database, you'll want to consider various aspects to ensure the confidentiality, integrity, and availability of the data. Here are specific security requirements for the database component of a weather app

## Access Control:

## Enforce strict access controls to the database. Use role-based access control (RBAC) to ensure that users have the minimum necessary privileges.

## Regularly review and update user permissions to reflect personnel changes and ensure that only authorized individuals can access the database.

## Authentication:

## Implement strong authentication mechanisms for accessing the database. Enforce the use of strong, unique passwords, and consider multi-factor authentication for added security.

## Avoid using default credentials, and regularly audit and monitor login attempts for any suspicious activity.

## Audit Logging:

## Implement comprehensive audit logging to record all database activities. This includes login attempts, data modifications, and any other relevant actions.

## Regularly review and analyze audit logs to detect and respond to potential security incidents.

## Data Validation:

## Implement input validation to prevent SQL injection attacks. Use parameterized queries or prepared statements to ensure that user input is treated as data and not executable code.

## Validate and sanitize user inputs to protect against other types of injection attacks.

## Security Requirement

When it comes to the security requirements for a weather app's database, you'll want to consider various aspects to ensure the confidentiality, integrity, and availability of the data. Here are specific security requirements for the database component of a weather app:

**Access Control:**

Enforce strict access controls to the database. Use role-based access control (RBAC) to ensure that users have the minimum necessary privileges.

Regularly review and update user permissions to reflect personnel changes and ensure that only authorized individuals can access the database.

**Authentication:**

Implement strong authentication mechanisms for accessing the database. Enforce the use of strong, unique passwords, and consider multi-factor authentication for added security.

Avoid using default credentials, and regularly audit and monitor login attempts for any suspicious activity.

**Audit Logging:**

Implement comprehensive audit logging to record all database activities. This includes login attempts, data modifications, and any other relevant actions.

Regularly review and analyze audit logs to detect and respond to potential security incidents.

**Data Validation:**

Implement input validation to prevent SQL injection attacks. Use parameterized queries or prepared statements to ensure that user input is treated as data and not executable code.

Validate and sanitize user inputs to protect against other types of injection attacks.

## Requirement attributes

## Business Rules

## Business rules for developing a weather app with a database help guide the decision-making process, ensure consistency, and align the development efforts with the overall objectives of the business. Here are some business rules that might be relevant for a weather app with a database:

## Data Accuracy:

## Rule: Weather data must be accurate and sourced from reliable meteorological sources.

## Rationale: Ensures that users receive trustworthy and up-to-date weather information.

## Real-time Updates:

## Rule: The app should provide real-time weather updates at predefined intervals.

## Rationale: Enhances user experience by delivering the most current weather information.

## User Location Privacy:

## 5.6 user Requirement

## User requirements for a weather app with a database are critical in shaping the app's features and functionality. Here are some user requirements to consider:

## Real-Time Updates:

## Users expect real-time updates for weather conditions, ensuring that the information is current and relevant.

## Hourly and Daily Forecasts:

## The app was provide both hourly and daily forecasts, giving users a comprehensive view of the upcoming weather.

## User-Friendly Interface:

## The app was feature an intuitive and user-friendly interface, making it easy for users to access and understand weather information.

## Weather Maps and Visualizations:

## Interactive maps, radars, and visualizations can enhance the user experience by providing a clear and dynamic representation of weather patterns.

## Temperature Units and Preferences:

## The app allow users to choose their preferred units for temperature (Celsius or Fahrenheit) and other relevant preferences

## .

## Accessible Design:

## Considerations for accessibility, such as text-to-speech features and high-contrast modes, can make the app inclusive for users with varying needs.

# Other Requirements

## Data and Category Requirement

## 

## There are several different attributes of weather data that are commonly collected and are purchasable. Generally, data providers will provide data points that include localized or global information on: temperature, precipitation, humidity, wind speed, wind direction, and cloud coverage.

## Appendix

## 

## Database Schema:

## List of tables with their fields and data types.

## Relationships between tables

## Document essential database functions or stored procedures.

## Include parameters, return values, and a brief description.

## Glossary

## Location ID:

## Unique identifier assigned to each location in the database.

## City:

## Name of the city for which weather data is recorded.

## Country:

## Name of the country associated with the location.

## Weather Data:

## Recorded information about atmospheric conditions, including temperature, humidity, wind speed, etc.

## Data ID:

## Unique identifier for each set of weather data in the database.

## Date:

## The specific date for which weather data is recorded.

## Temperature:

## Measurement of the warmth or coldness of the atmosphere.

## Humidity:

## The amount of water vapor present in the air.

## Class Diagram

## 

Creating a case diagram involves identifying the actors, use cases, and the relationships between them. In the context of a weather app with a database, you might have actors like "User" and "Weather Data Provider" interacting with various usecases.

