WeatherPlus: Your Personalized Weather Companion

Keyur Patel - 16010421073
Ronit Mehta - 16010421056
Batch - A2
Exp-5(Software Design Document)

April 2024

Aim:

To prepare Software Design Document (SDD)

Resources:

Internet Explorer, LaTex Editor

Results: Software Design Document (SDD) in given format

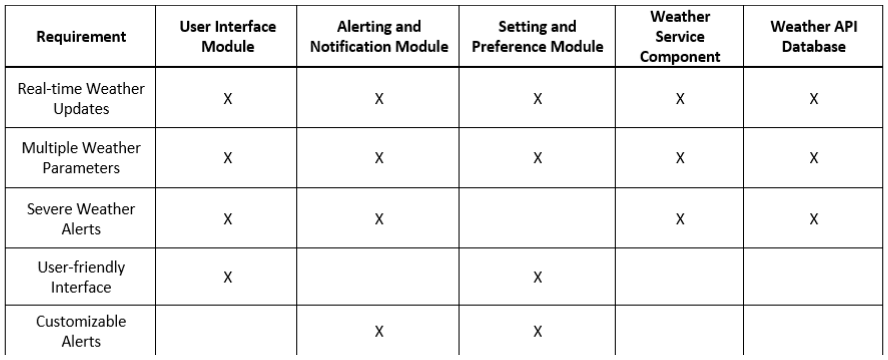
1 Introduction

1.1 Design Overview The design of our weather forecasting application encompasses a robust and user-centric approach aimed at delivering accurate and timely weather information to our users. By leveraging cutting-edge technologies and adhering to industry best practices, our design ensures reliability, scalability, and ease of use.

Essential Features

- **Data Aggregation and Processing:** Our application integrates data from multiple reliable sources, including meteorological agencies, satellites, and weather stations. Advanced algorithms are employed to process this data, ensuring accuracy and reliability in forecasting.
- **User Interface Design:** We prioritize an intuitive and visually appealing user interface to enhance user experience. Through careful design considerations, we aim to provide users with easy access to weather information, forecasts, and interactive features.

1.2 Requirements Traceability Matrix .



Requirement	User Interface Module	Alerting and Notification Module	Setting and Preference Module	Weather Service Component	Weather API Database
Real-time Weather Updates	X	X	X	X	X
Multiple Weather Parameters	X	X	X	X	X
Severe Weather Alerts	X	X		X	X
User-friendly Interface	X		X		
Customizable Alerts		X	X		

Figure 1: Requirements Traceability Matrix

2 System Architecture Design

2.1 Chosen System Architecture

The chosen system architecture for the Weather Forecasting App is a three-tier architecture: Presentation Layer, Application Layer, and Data Layer. This architecture provides scalability, modularity, and separation of concerns.

1. **Presentation Layer:**
   - The front-end of the application where the user interacts.
   - Responsible for rendering user interfaces, such as weather forecasts, location search, and user settings.
   - Implemented using HTML, CSS, and JavaScript .
2. **Application Layer:**
   - Serves as the business logic layer, handling user requests, processing data, and managing application flow.
   - Implements the "Favorites" and "Search Location" features.
   - Developed using Node.js for the server-side logic, Express.js for RESTful API routing, and business logic modules.
3. **Data Layer:**
   - Stores and manages weather data, user preferences, and location information.
   - Utilizes a relational database (such as PostgreSQL) to store user favorites, historical weather data, and settings.
   - The database is accessed through an Object-Relational Mapping (ORM) library like Sequelize in Node.js.

Technical Risks and Contingency Plans:

Risk 1: API Downtime Contingency: Implement caching mechanisms to store frequently accessed data locally.

Risk 2: Data Security Contingency: Implement encryption for sensitive user data in the database and during transmission.

2.2 Discussion of Alternative Designs

1. **Monolithic Architecture:** Considered but not chosen due to potential difficulties in scaling and maintaining a large, tightly-coupled codebase.
2. **Microservices Architecture:** Discussed for its modularity and scalability benefits but deemed too complex for the current scope of the project.

2.3 System Interface Description:

1. **Operating System Interface (OS):** The application is designed to be platform-independent, running on any OS supporting Node.js.
2. **Files:** Local file storage is used for temporary caching of user settings and session data.
3. **Networking:** Utilizes HTTP/HTTPS protocols for client-server communication.
4. **Graphics Libraries:** Uses built-in CSS and React.js components for UI design.

3 Detailed Description of Components

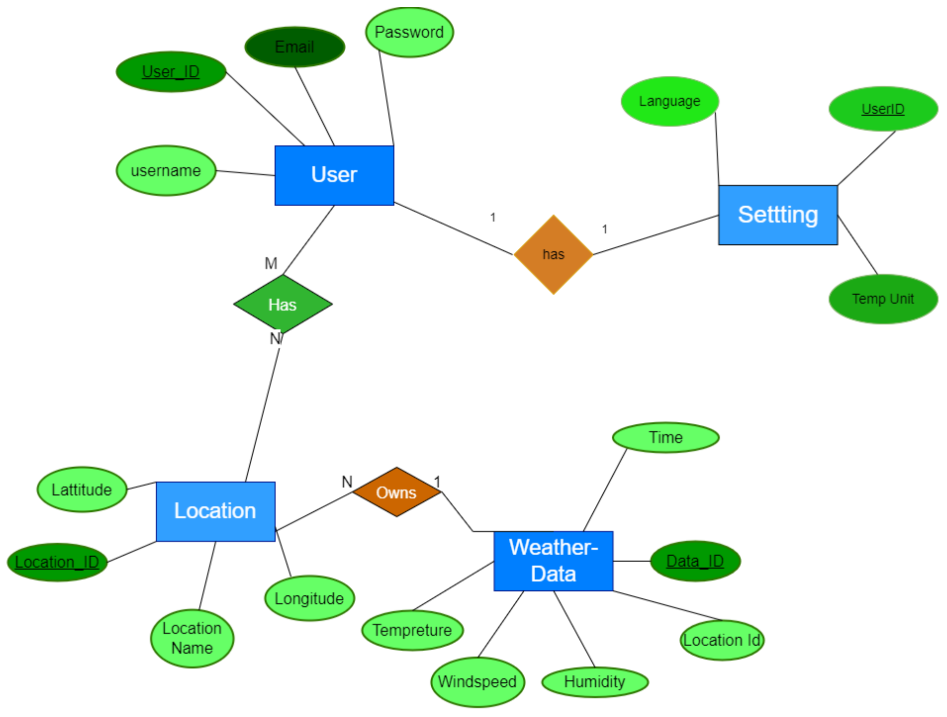


Figure 2: ER-Diagram

3.1 User

1. **Responsibilities:**
   - Manages user authentication and authorization.
   - Stores user preferences, such as favorite locations and settings.
   - Provides an interface for users to interact with the app, including login/logout functionality.
2. **Constraints:**
   - User data must be securely stored and encrypted.
   - User interface components must be responsive and accessible across devices.
3. **Composition:**
   - Includes user profile data (name, email, etc.).
   - Contains user-generated settings and preferences.
   - Interfaces with the authentication service for login/logout operations.

4. **Interactions:**
    - Interacts with the Location and Setting components to manage user preferences.
    - Communicates with the Authentication Service for user login and authentication.
    - Provides data to the Presentation Layer for rendering user-specific content.
5. **Resources:**
    - Database table for user profiles and settings.
    - Front-end UI components for user login and settings management.
    - Authentication service for verifying user credentials.

3.2 Location

1. **Responsibilities:**
    - Manages location data for weather forecasts.
    - Provides functionality to search for and save locations as favorites.
    - Retrieves geographic coordinates for specified locations.
2. **Constraints:**
    - Must handle various location formats (city names, zip codes, etc.).
    - Accuracy of location data retrieval is critical for weather accuracy.
3. **Composition:**
    - Contains a database table for storing favorite locations per user.
    - Utilizes geocoding services to convert location inputs to coordinates.
    - Interfaces with the User component to retrieve user-specific favorite locations.
4. **Interactions:**
    - Communicates with external geocoding APIs for location data.
    - Provides location data to the Weather Data component for fetching weather information.
    - Interfaces with the User component to manage user-specific favorites.
5. **Resources:**
    - Database table for storing favorite locations.
    - External geocoding API services (e.g., Google Maps Geocoding API).
    - Front-end components for location search and favorite management.

3.3 Settings

1. **Responsibilities:**
    - Manages application settings and preferences.
    - Allows users to customize units (e.g., temperature in Celsius or Fahrenheit).
    - Handles user notifications and alert preferences.
2. **Constraints:**
    - Must provide a user-friendly interface for setting customization.
    - Settings must persist across user sessions.
3. **Composition:**

- Contains a database table for storing user-specific settings.
- Includes options for unit selection (temperature, wind speed, etc.).
- Interfaces with the User component to retrieve and update user settings.

4. **Interactions:**
   - Communicates with the User component to fetch and update user settings.
   - Provides settings data to the Presentation Layer for user interface customization.
   - Interfaces with the Weather Data component to apply user preferences to weather data.
5. **Resources:**
   - Database table for storing user settings.
   - Front-end UI components for setting customization.
   - Integration with the Notification Service for managing user alerts.

3.4 Weather Data

1. **Responsibilities:**
   - Fetches and processes weather data from external APIs.
   - Provides weather forecasts and current conditions for specified locations.
   - Updates weather information periodically based on user preferences.
2. **Constraints:**
   - Relies on external weather APIs for accurate and up-to-date data.
   - Must handle various data formats returned by different weather services.
3. **Composition:**
   - Utilizes external weather APIs (e.g., OpenWeatherMap, WeatherAPI) for data retrieval.
   - Contains data models for storing weather forecasts and conditions.
   - Interfaces with the Location and Setting components to fetch user-specific weather data.
4. **Interactions:**
   - Communicates with external weather APIs to fetch weather data.
   - Provides weather information to the Presentation Layer for display.
   - Interfaces with the Location component to retrieve weather data for specified locations.
5. **Resources:**
   - External weather APIs for data retrieval.
   - Database tables for storing weather forecasts and conditions.
   - Background processes or services for periodic data updates.

4 User Interface Design

4.1 Description of the User Interface

4.1.1 Screen Images

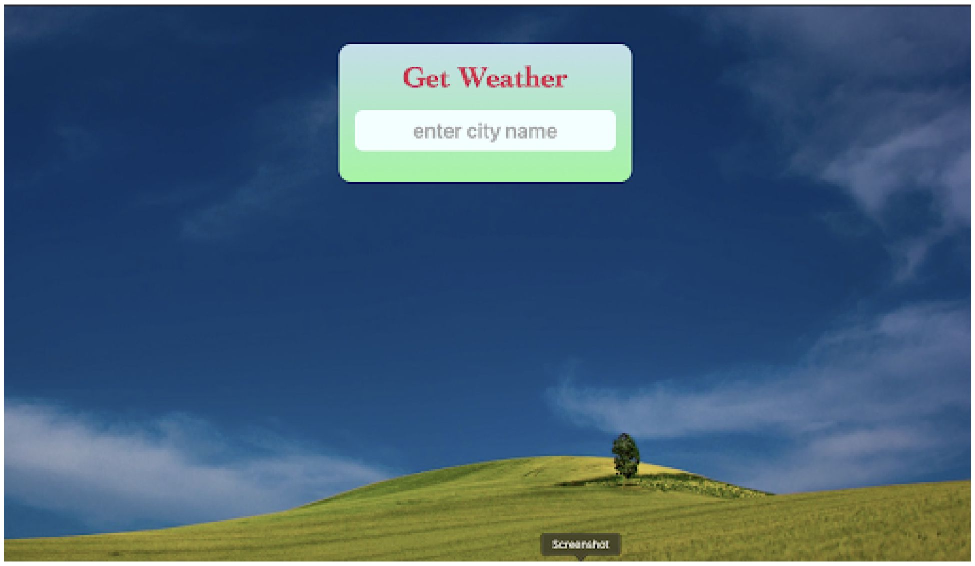


Figure 3: UI-Interface

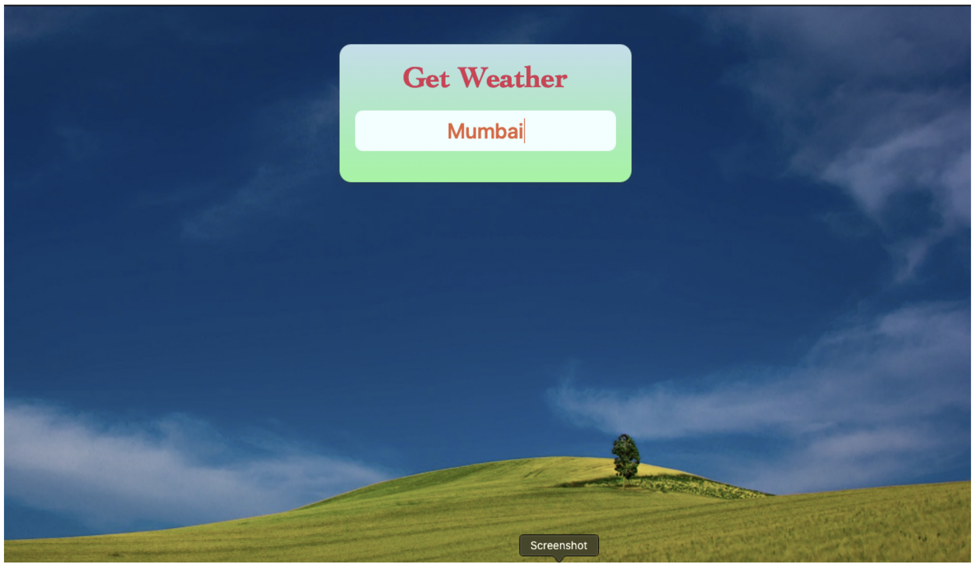
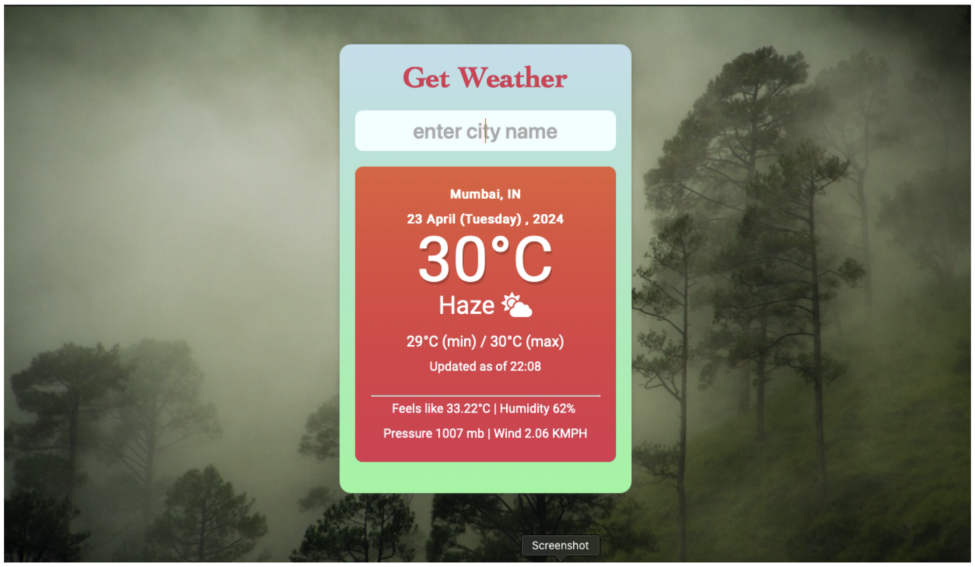
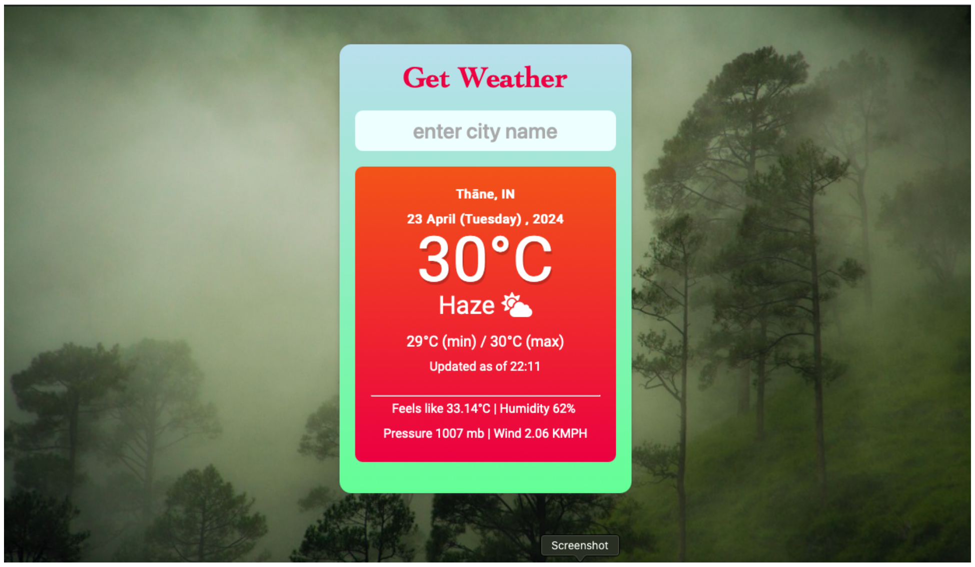


Figure 4: UI-Interface



Mumbai, IN
23 April (Tuesday) , 2024
30°C
Haze
29°C (min) / 30°C (max)
Updated as of 22:08
Feels like 33.22°C | Humidity 62%
Pressure 1007 mb | Wind 2.06 KMPH

Figure 5: UI-Interface for displaying weather



Thāne, IN
23 April (Tuesday) , 2024
30°C
Haze
29°C (min) / 30°C (max)
Updated as of 22:11
Feels like 33.14°C | Humidity 62%
Pressure 1007 mb | Wind 2.06 KMPH

Figure 6: UI-Interface for displaying weather

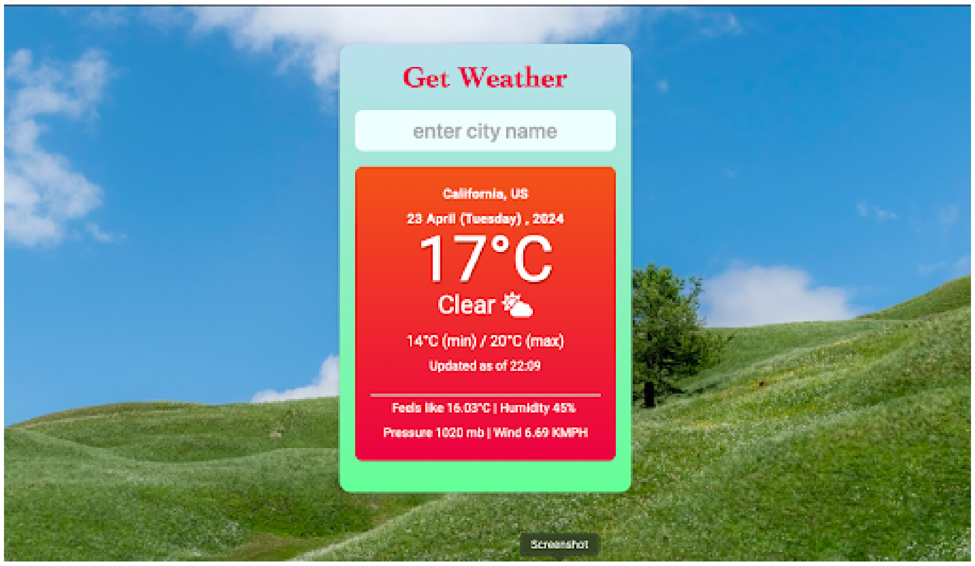


Figure 7: UI-Interface for displaying weather

4.1.2 Objects and Actions

(a) **Home Screen:**

i. ***Objects:*** Search bar, Current Location button, Settings icon, Favorites list.

ii. ***Actions:***

- *Search bar:* Allows users to input location for weather search.
- *Current Location button:* Retrieves weather for user's current location.
- *Settings icon:* Opens settings menu for unit preferences and notifications.
- *Favorites list:* Displays user's saved favorite locations for quick access.

(b) **Settings Screen:**

i. ***Objects:*** Units dropdown menu, Notifications toggle.

ii. ***Actions:***

- *Units dropdown menu:* Lets users select temperature and wind speed units.
- *Notifications toggle:* Enables or disables weather alerts and notifications.

Weather Details Screen:

i. ***Objects:*** Detailed weather information (temperature, humidity, etc.), Share button.

ii. ***Actions:*** Detailed weather info: Displays in-depth weather data for a specific location.

Location Permission Screen:

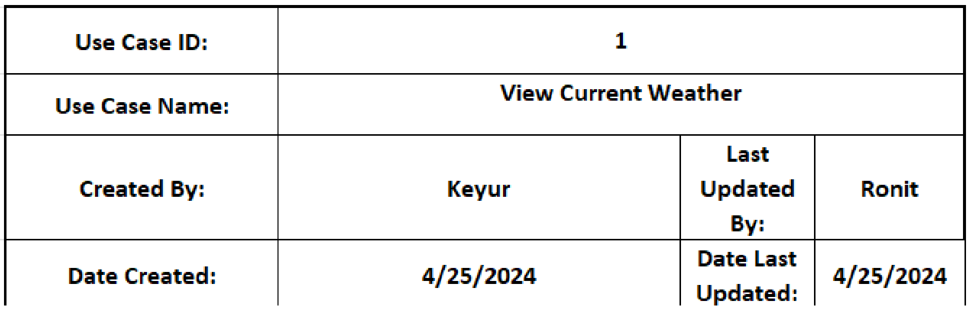
i. ***Objects:*** Permission request message, Allow/Deny buttons.

ii. ***Actions:***

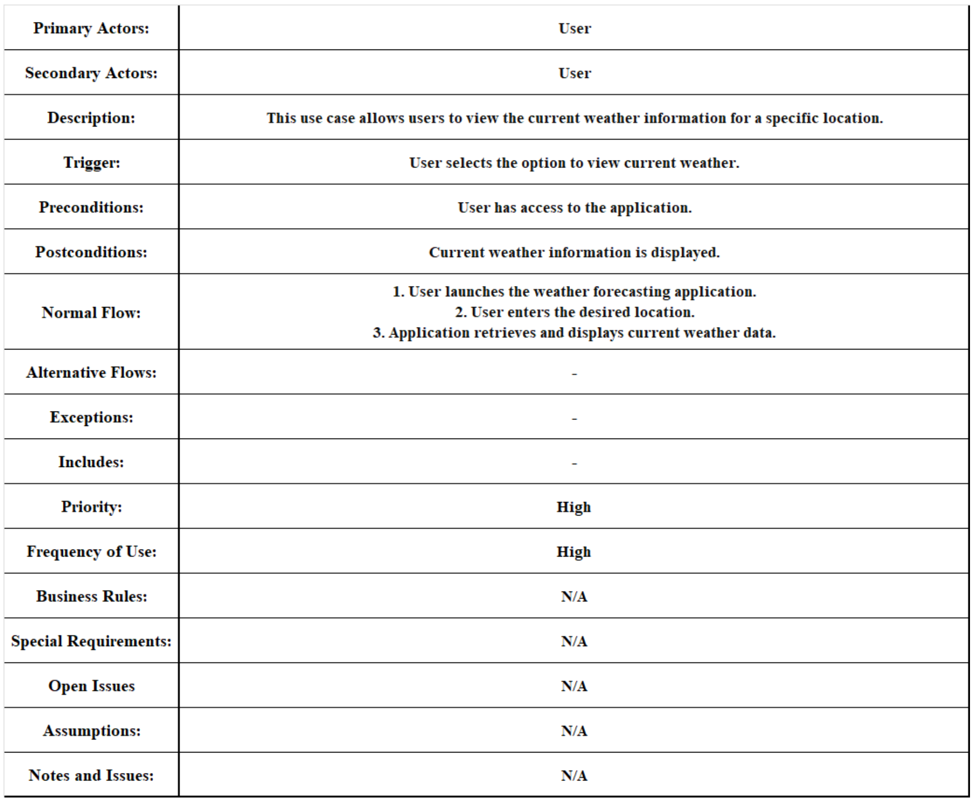
- *Permission request message:* Asks users to grant location access for current weather.
- *Allow/Deny buttons:* Enables users to grant or deny location permission.

5 System Architecture

5.1 Use Case-1 .

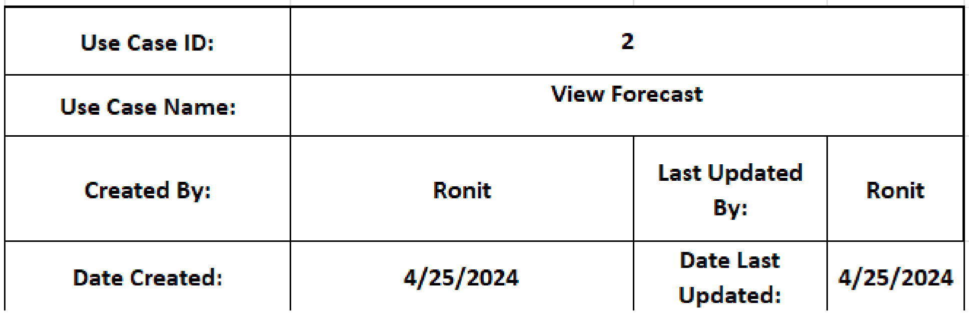


Use Case ID:	1		
Use Case Name:	View Current Weather		
Created By:	Keyur	Last Updated By:	Ronit
Date Created:	4/25/2024	Date Last Updated:	4/25/2024

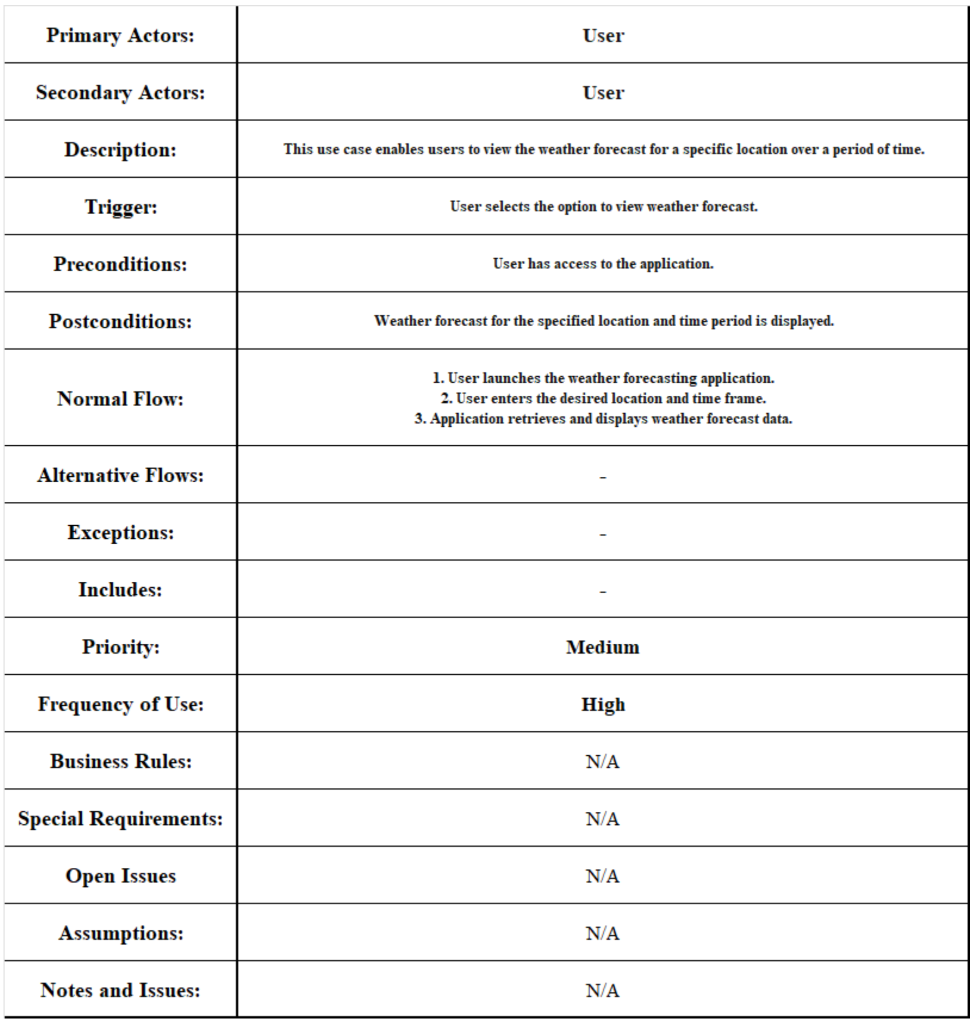


Primary Actors:	User
Secondary Actors:	User
Description:	This use case allows users to view the current weather information for a specific location.
Trigger:	User selects the option to view current weather.
Preconditions:	User has access to the application.
Postconditions:	Current weather information is displayed.
Normal Flow:	1. User launches the weather forecasting application. 2. User enters the desired location. 3. Application retrieves and displays current weather data.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	High
Frequency of Use:	High
Business Rules:	N/A
Special Requirements:	N/A
Open Issues	N/A
Assumptions:	N/A
Notes and Issues:	N/A

5.2 Use Case-2 .

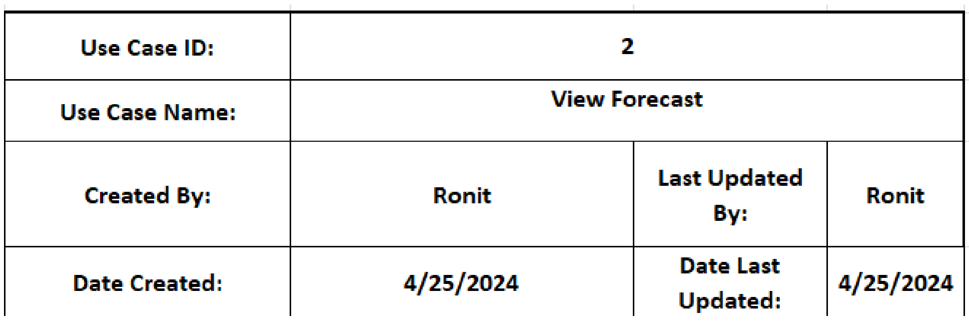


Use Case ID:	2		
Use Case Name:	View Forecast		
Created By:	Ronit	Last Updated By:	Ronit
Date Created:	4/25/2024	Date Last Updated:	4/25/2024



Primary Actors:	**User**
Secondary Actors:	**User**
Description:	**This use case enables users to view the weather forecast for a specific location over a period of time.**
Trigger:	**User selects the option to view weather forecast.**
Preconditions:	**User has access to the application.**
Postconditions:	**Weather forecast for the specified location and time period is displayed.**
Normal Flow:	**1. User launches the weather forecasting application.** **2. User enters the desired location and time frame.** **3. Application retrieves and displays weather forecast data.**
Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	**Medium**
Frequency of Use:	**High**
Business Rules:	N/A
Special Requirements:	N/A
Open Issues	N/A
Assumptions:	N/A
Notes and Issues:	N/A

5.3 Use Case-3 .

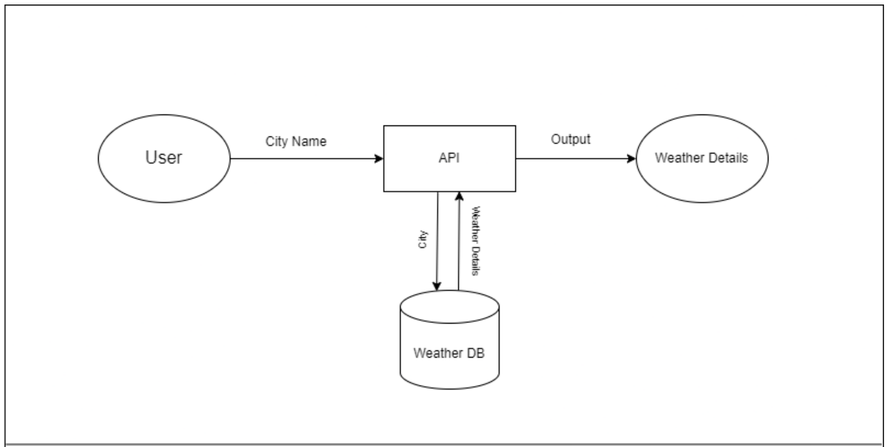


Use Case ID:	**2**		
Use Case Name:	**View Forecast**		
Created By:	**Ronit**	**Last Updated By:**	**Ronit**
Date Created:	**4/25/2024**	**Date Last Updated:**	**4/25/2024**

Primary Actors:	**User**
Secondary Actors:	**User**
Description:	**This use case allows users to set up custom weather alerts based on specific criteria such as temperature, precipitation, etc.**
Trigger:	**User selects the option to set weather alerts.**
Preconditions:	**User has access to the application.**
Postconditions:	**Weather alerts are set up according to user-defined criteria.**
Normal Flow:	**1. User launches the weather forecasting application. 2. User navigates to the weather alerts section. 3. User defines the criteria for the alert (e.g., temperature threshold). 4. Application sets up the alert based on user input.**
Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	**Low**
Frequency of Use:	**Medium**
Business Rules:	N/A
Special Requirements:	N/A
Open Issues	N/A
Assumptions:	N/A
Notes and Issues:	N/A

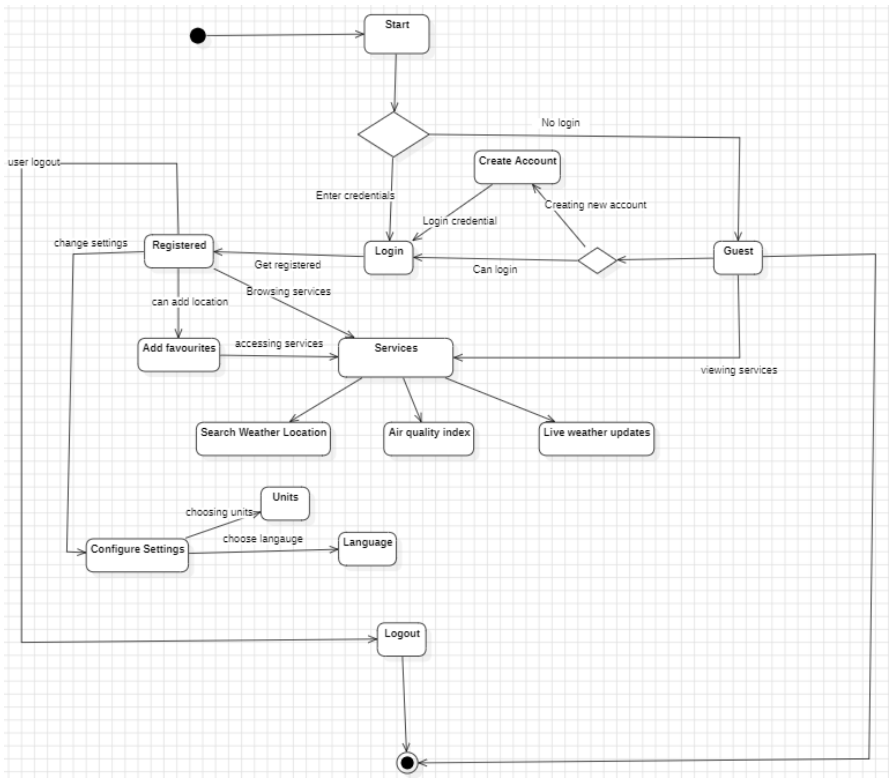
6 Data Flow Specfications

6.1 Level 0 DFD with description .



Data Flow Diagram of Weather App

6.2 Level 1 DFD with description .



Questions:

1. **Explain Architecture design patterns and styles with examples.**

Ans: (a) **Layered Architecture**

   i. ***Description:*** Divides an application into layers (Presentation, Business Logic, Data) where each layer has a specific responsibility. ***Example:***

      A. ***Presentation Layer:*** Handles user interaction (UI).

      B. ***Business Logic Layer:*** Contains the application's business rules and logic.

      C. ***Data Layer:*** Manages data persistence and storage.

   ii. ***Benefits:*** Separation of concerns, easier maintenance, and scalability.

   iii. ***Example Framework:*** Java EE

(b) **Model-View-Controller (MVC)**

   i. ***Description:*** Separates an application into three interconnected components: Model (data), View (UI), and Controller (logic).

   ii. ***Example:***
      Model: Manages data and business logic. ***View:*** Renders the UI. ***Controller:*** Handles user input, interacts with the model, updates the view.

   iii. ***Benefits:*** Modularity, easier maintenance, and parallel development.

   iv. ***Example Framework:*** Ruby on Rails, Django

(c) **Microservices Architecture**

   i. ***Description:*** Decomposes an application into smaller, independently deployable services.

   ii. ***Example:*** Each service is responsible for a specific function (e.g., user authentication, order processing). Services communicate over the network, often via APIs.

   iii. ***Benefits:*** Scalability, independent deployment, technology diversity.

   iv. ***Example:*** Netflix, Amazon

(d) **Event-Driven Architecture (EDA)**

   i. ***Description:*** Components communicate through events, enabling loose coupling and asynchronous processing.

   ii. ***Example:*** Events are produced and consumed by different parts of the system. Pub/Sub systems (like Apache Kafka or RabbitMQ) are often used.

   iii. ***Benefits:*** Scalability, flexibility, and responsiveness.

   iv. ***Example:*** IoT applications, real-time analytics

Outcomes

CO3 Demonstrate requirements, modeling and design of a system

Conclusion

After completing this experiment we have successfully created Software Design Document for my Weather app forecasting project.

References

Books:

1. Roger S. Pressman, Software Engineering: A practitioners Approach, 7th Edition, McGraw Hill, 2010.

2. Technical report on Guidelines for Documents Produced by Student Projects In Software Engineering based on IEEE standards

3. Timothy C. Lethbridge, Robert Laganiere " Object-Oriented Software Engineering – A practical software development using UML and Java", Second Edition, Tata McGraw-Hill, New Delhi,2004