Group: CS32_4

Project: Air Quality Monitoring System

I. Feasibility Study Analysis

1. Technical Feasibility

- **Tools & Frameworks:**
  - **Frontend:** React.js (for Web), React Native (for Mobile)
  - **Backend:** FastAPI / Flask (Python-based REST API)
  - **Database:** PostgreSQL / MongoDB (to store historical data)
  - **APIs:** AQICN API / OpenWeatherMap API (to fetch air quality data)
  - **Cloud Deployment:** AWS, Firebase, or Heroku
- **Scalability:**
  - The system should handle multiple users accessing real-time air quality data.
  - Load balancing techniques can be used for high-traffic scenarios.

2. Schedule Feasibility

Week	Task	Duration	Milestone
1-2	Requirement Analysis, API Research	2 weeks	Finalized project scope
3-4	System Design, UML diagrams, Database Schema	2 weeks	Architecture ready
5-6	Backend & API Development	2 weeks	Core API functional
7-8	Frontend Development	2 weeks	Basic UI functional
9	Testing (Unit & Integration)	1 week	Test cases executed
10	Deployment & Final Report	1 week	Project Submission

3. Quality Feasibility

- **Testing:** Unit Testing, Integration Testing, and Load Testing.
- **Maintenance:** Regular updates to API endpoints, bug fixes, and security patches.

4. Market Feasibility

- **Target Customers:**
  - General public concerned with air pollution.
  - Organizations monitoring environmental conditions.
  - Government bodies for policy-making.

- **Demand Assessment:**
    - With increasing air pollution concerns, users want an easy-to-access platform.
    - The project can integrate ML for pollution prediction, making it valuable.

II. Project Requirements and Requirement Analysis

1. Customer-Provided Requirements

- A platform (web & mobile) to display real-time AQI data.
- Option to check historical data and view air pollution trends.
- Notifications for high pollution levels.
- User authentication for personalized alerts.

2. Requirement Analysis

- **Functional Requirements:**
    - Fetch air quality data from external APIs.
    - Store historical air quality data for trend analysis.
    - User registration for notifications.
- **Non-Functional Requirements:**
    - **Performance:** Should fetch and display data within 2 seconds.
    - **Security:** API keys should be encrypted; authentication for personal features.
    - **Scalability:** The system should support thousands of users simultaneously.

III. Software Requirements Specification (SRS)

(a) Introduction

- **Purpose:** To provide real-time and historical air quality data, helping users make informed decisions about environmental conditions.
- **Scope:** The system will offer AQI visualization, notifications, and historical data analysis via a web and mobile app.

(b) Overall Description

- **Product Features:**
    - Real-time AQI monitoring.
    - Historical data analysis and trends.
    - Personalized notifications for high pollution levels.
- **User Classes:**
    - **Guest Users:** View AQI data for different locations.
    - **Registered Users:** Get alerts, save locations, and track historical data.

- **Operating Environment:**
    - Web-based and mobile-based.
    - Compatible with Windows, macOS, Android, and iOS.

(c) System Features and Requirements

Functional Requirements

Feature	Description	Inputs	Processing	Outputs
Fetch AQI Data	Retrieve data from APIs	Location	Call API	Display AQI value
Historical Data	Store & analyse past AQI values	API Data	Database storage	Graphs & charts
User Authentication	Register/login users	User details	Validate credentials	Access to features
Notifications	Alert users about high AQI	AQI Threshold	Send notification	Push alerts

Non-Functional Requirements

- **Performance:** The system should handle at least **10,000 requests per second**.
- **Security:** Use JWT for authentication, encrypt API keys.
- **Reliability:** Ensure **99.9% uptime** with backup servers.
- **Maintainability:** Modular code for easy updates and scalability.

(d) System Attributes

Attribute	Details
Availability	99.9% uptime with cloud hosting
Scalability	Supports thousands of users
Backup & Recovery	Daily database backups
Compatibility	Works on Web, Android, iOS

IV. Chosen SDLC Model & Justification

Model Chosen: Agile Model

Justification:

Incremental Development – Features can be added in phases (e.g., AQI fetching first, then historical trends, then ML predictions).

User Feedback Integration – Allows for quick modifications based on testing results.

Risk Management – Reduces risk since testing happens in every iteration.

Continuous Improvement – Easy to update APIs or UI based on new pollution trends.

Flexibility – If APIs change or we want to add ML predictions, Agile allows modifications without redoing the entire system.