AIR QUALITY MONITORING SYSTEM

01. Project Requirements:

Define the Scope: Determine the specific air quality parameters to monitor (e.g., PM2.5, PM10, CO2, etc.).

Hardware: Choose appropriate sensors and microcontrollers for data collection.

Software: Plan the backend and frontend technologies for data processing and user interface.

User Features: Decide on user authentication, real-time data visualization, historical data analysis, and alerts.

02. **Design and Architecture**:

Hardware Setup: Integrate sensors (e.g., particulate matter sensor, gas sensor) with microcontrollers (Arduino, Raspberry Pi).

Backend: Choose a backend framework (e.g., Node.js with Express) for data processing and storage.

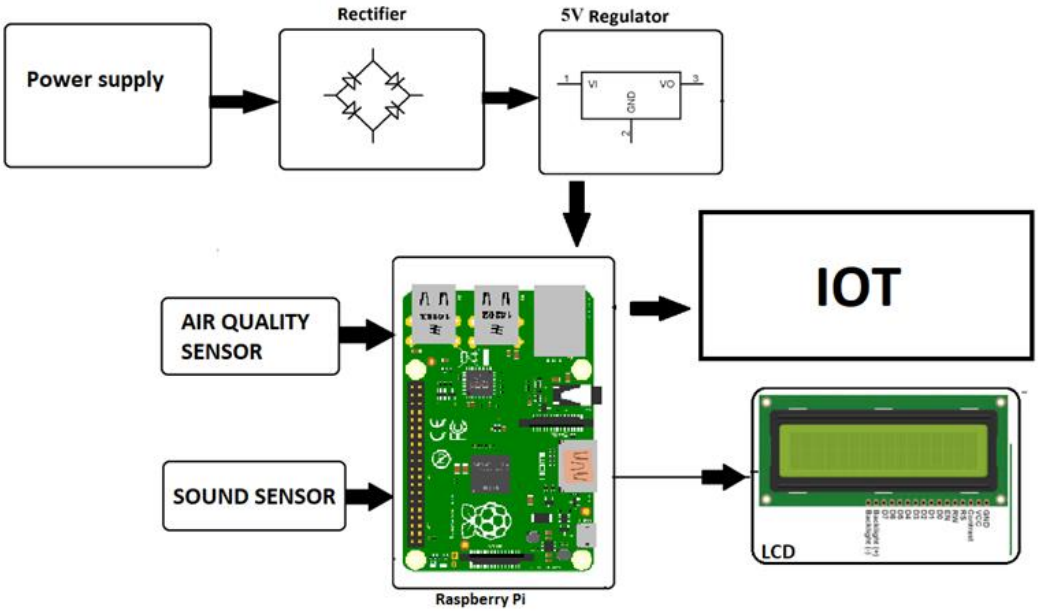
Database: Use a database system (e.g., MongoDB) to store sensor data.

Frontend: Select frontend technologies (e.g., HTML, CSS, JavaScript, React.js) for building the user interface.

Real-time Communication: Implement WebSocket for real-time data updates to the frontend.

System uses air sensors to sense presence of harmful gases/compounds in the air and constantly transmit this data to microcontroller. Also system keeps measuring sound level and reports it to the online server over IOT. The sensors interact with microcontroller which processes this data and transmits it over internet. This allows authorities to monitor air pollution in different areas and take action against it. Also authorities can keep a watch on the noise pollution near schools, hospitals and no honking areas, and if system detects air quality and noise issues it alerts authorities so they can take measures to control the issue.

Block Diagram:



3. **Development**:

Sensor Integration: Write code to collect data from sensors and send it to the backend.

Backend Development: Develop APIs to receive sensor data, process it, and store it in the database.

Database Integration: Store sensor data in the database for historical analysis.

Frontend Development: Create interactive charts and graphs to visualize real-time and historical data.

User Authentication: Implement user authentication and authorization mechanisms.

Alert System: Set up alerts for specific air quality thresholds.

4. **Testing**:

Unit Testing: Test individual components/modules to ensure they work as expected.

Integration Testing: Test the communication between sensors, backend, and frontend.

User Acceptance Testing (UAT): Allow users to test the system for usability and provide feedback.

5. **Documentation**:

Technical Documentation: Document the hardware setup, software architecture, and algorithms used.

User Manual: Create a user manual explaining how to access the system, interpret data, and set up alerts.

Code Documentation: Comment your code thoroughly for future reference.

Error Handling: Document common errors and their troubleshooting methods.

6. **Deployment**:

Choose Hosting: Select a hosting service (e.g., AWS, Heroku) for deploying your web application.

Deployment: Deploy your application, database, and necessary services on the chosen hosting platform.

7. **Security**:

Data Encryption: Implement encryption techniques to secure sensitive data transmission.

Authentication and Authorization: Ensure secure user authentication and authorization mechanisms are in place.

Regular Security Audits: Regularly audit the platform for security vulnerabilities.

8. **Performance Optimization**:

Code Optimization: Optimize your code for better performance.

Database Optimization: Optimize database queries and indexes for faster data retrieval.

9. **Maintenance and Support**:

Bug Fixing: Address any bugs or issues reported by users.

Feature Updates: Implement new features or enhancements based on user feedback.

24/7 Support: Provide continuous support to users, especially for critical issues.