KINGS ENGINEERING COLLEGE

PROJECT TITTLE: Air quality analysis

DEPARTMENT: Bio Medical Engineering

MENTOR: Mary Lalitha

TEAM MEMBERS:

Veera Ganesh S

Vishwa M

Vignesh S

Vigneshwaran N

Vetrivel N

Continue building the project by developing the data sharing platform.

Database Setup:

Choose a suitable database system to store the collected air quality data. Options include PostgreSQL, MySQL, or cloud-based databases like Firebase or AWS DynamoDB.

API Design:

Design RESTful APIs or Graph QL endpoints to allow data access and sharing. These APIs will enable communication between your database and the user interface.

User Authentication:

Implement user authentication to control access to the platform. Common methods include username/password, OAuth, or API tokens.

User Roles and Permissions:

Define user roles (e.g., admin, regular user) and set permissions for each role to manage data access and sharing.

Web or Mobile Interface:

Develop a user-friendly web or mobile interface that allows users to view real-time air quality data and historical trends.

Real-Time Data Updates:

Integrate mechanisms for real-time data updates. Consider using technologies like Web Sockets or server-sent events (SSE) for instant data delivery.

Data Visualization:

Create interactive and informative data visualizations, such as charts, graphs, and maps, to present air quality data in an understandable format.

Data Filters and Search:

Implement features that enable users to filter and search for specific data points based on location, time, or air quality parameters.

Data Sharing Options:

Allow users to share specific data sets or visualizations with others through shareable links or social media integrations.

Notifications and Alerts:

Set up a notification system that informs users of important updates or when air quality levels reach predefined thresholds.

Data Export:

Provide options for users to export data in various formats (e.g., CSV, PDF) for further analysis or reporting.

Use web development technologies (e.g., HTML, CSS, JavaScript) to create a platform that displays real-time air quality data.

here web application to display real-time air quality data using HTML, CSS, and JavaScript:

HTML (index.html):

```html
<!DOCTYPE html>
<html lang="e n">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link rel="stylesheet" href="styles.css">
 <title>Real-Time Air Quality</title>
</head>
<body>
 <div class="container">
  <h1>Real-Time Air Quality</h1>
  <div class="data">
   <h2>Air Quality Index (AQI): <span id="aqi">Loading...</span></h2>
   <p>Last Update: <span id="last-update">-</span></p>
   </div>
  </div>
 <script src ="app.js"></script>
</body>
</html>
```

CSS (styles.css):

```css
css
body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
    text-align: center;
}

.container {
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
    padding: 20px;
    margin: 50px auto;
    width: 80%;
    max-width: 400px;
}

h1 {
    color: #333;
}

.data {
    margin-top: 20px;
}
```

3. **JavaScript (app.js)**:

```javascript
javascript
const aqiElement = document.getElementById('aqi');
const lastUpdateElement = document.getElementById('last-update');

function fetchData() {
    // Replace this URL with the source of your real-time air quality data (API endpoint).
    const apiUrl = 'https://your-air-quality-api-url-here';

    fetch(apiUrl)
        .then(response => response.json())
        .then(data => {
            const aqi = data.aqi;
            const lastUpdate = new Date(data.time);
```
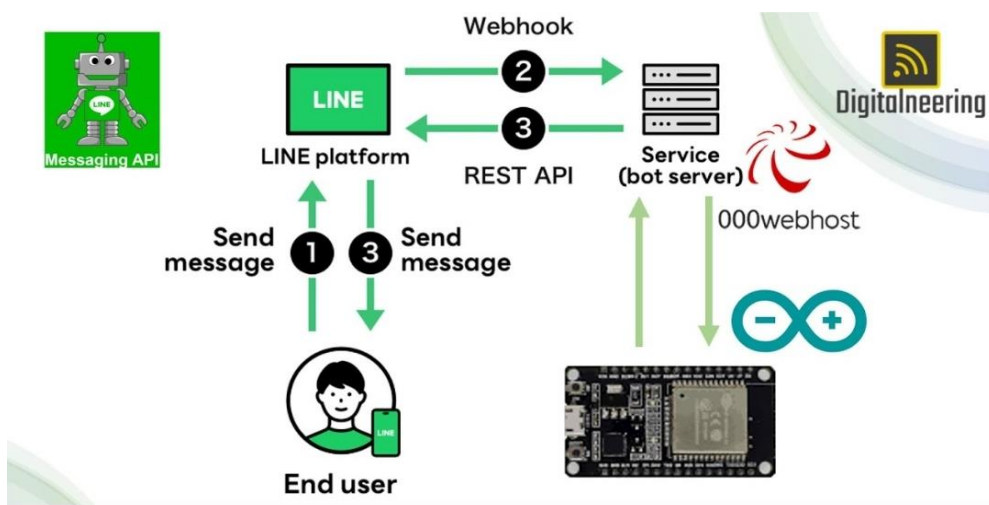
```
      aqiElement.textContent = aqi;
      lastUpdateElement.textContent = lastUpdate.toTimeString();
    })
    .catch(error => {
      console.error('Error fetching data:', error);
      aqiElement.textContent = 'N/A';
      lastUpdateElement.textContent = 'Error';
    });
}
```


Design the platform to receive and display air quality data sent by the IoT devices.

IoT Devices:

Develop or integrate IoT sensors that measure air quality parameters (e.g., PM2.5, PM10, CO2, temperature, humidity).
Ensure devices are connected to the internet (Wi-Fi, cellular, LoRa, etc.) for data transmission.

Data Collection and Ingestion:

Set up a data ingestion layer to receive data from IoT devices.
Use MQTT, HTTP, or other IoT protocols to collect data.
Implement data validation and cleansing to ensure accuracy.

Database:

 Store incoming data in a database (e.g., SQL or NoSQL) for historical reference.
 Design the schema to efficiently store time-series data.

Cloud :

Host the platform on a cloud service (e.g., AWS, Azure, Google Cloud) or on-premises servers.
Ensure scalability, reliability, and security.

API:

Develop RESTful or Graph QL APIs for IoT devices to send data to the platform.
Implement authentication and authorization mechanisms for device access.

Data Processing:

Use data processing tools like Apache Kafka, Spark, or Flink for real-time processing and analysis.
Calculate air quality indices (AQI) from raw sensor data.

User Interface:

Build a web or mobile interface for users to access air quality data.
Display real-time and historical air quality data, including visualizations (charts, maps, etc.).
Implement user authentication and role-based access control.

Notifications:

Set up alerts and notifications for users based on predefined air quality thresholds.
Notifications can be sent via email, SMS, or push notifications.

Data Analytics:

Use machine learning models to predict air quality trends or detect anomalies.
Provide insights and recommendations for users.

Security:

Implement security measures to protect data in transit and at rest.
Regularly update and patch the system to guard against vulnerabilities.

Compliance:

Ensure compliance with data privacy regulations (e.g., GDPR) and industry standards.

Maintenance and Monitoring:

Monitor system health and performance using tools like Prometheus or Grafana.
Plan for regular maintenance and updates to keep the platform running smoothly.
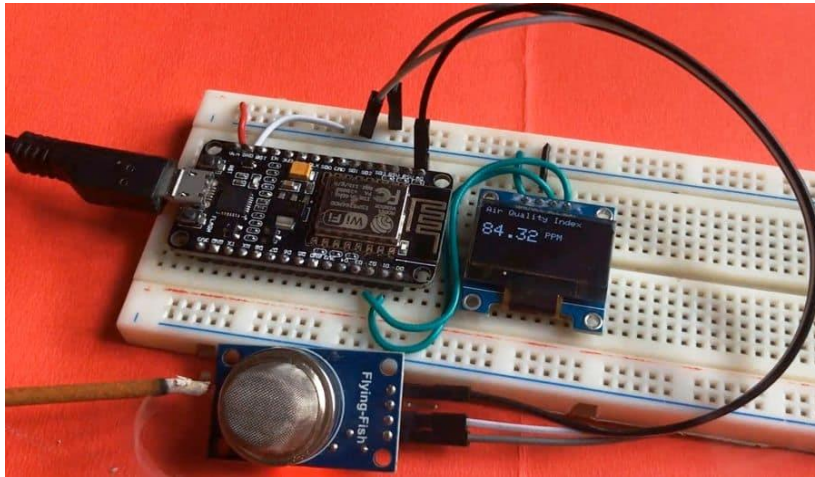
Designing mobile apps for iOS and Android platforms that provide users with access to real-time air quality level updates requires careful planning and a user-friendly interface. Below, I'll outline the design considerations for creating these mobile apps:

```
Flutter code example:
flutter create air_quality_app
cd air_quality_app
dependencies:
 flutter:
 sdk: flutter
http: ^0.13.3
 charts_flutter: ^0.11.0
main.dart: Main entry point for the app.
screens/home_screen.dart: Home screen displaying air quality information.
widgets/air_quality_chart.dart: Widget for displaying charts.
services/api_service.dart: Service for making API calls.
import 'package:flutter/material.dart';
import 'screens/home_screen.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
```

```dart
  @override
  Widget build(BuildContext context) {
  return MaterialApp(
  title: 'Air Quality App',
  theme: ThemeData(
  primarySwatch: Colors.blue,
  ),
  home: HomeScreen(),
  );
  }
}
import 'package:flutter/material.dart';
import 'widgets/air_quality_chart.dart';
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
  return Scaffold(
  appBar: AppBar(
  title: Text('Air Quality App'),
  ),
body: Center(
  child: AirQualityChart(),
  ),
  );
  }
}
import 'package:flutter/material.dart';
import 'package:charts_flutter/flutter.dart' as charts;
class AirQualityChart extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
  // Implement your chart logic here using the 'charts' package
  return Container(
  height: 300,
  child: Placeholder(), // Replace with your chart widget
  );
  }
}
import 'dart:convert';
import 'package:http/http.dart' as http;
class ApiService {
  final String apiUrl; // Replace with your API endpoint
  ApiService(this.apiUrl);
  Future<Map<String, dynamic>> getAirQualityData() async {
  final response = await http.get(Uri.parse(apiUrl));
  if (response.statusCode == 200) {
  return json.decode(response.body);
  } else {
  throw Exception('Failed to load air quality data');
  }
```
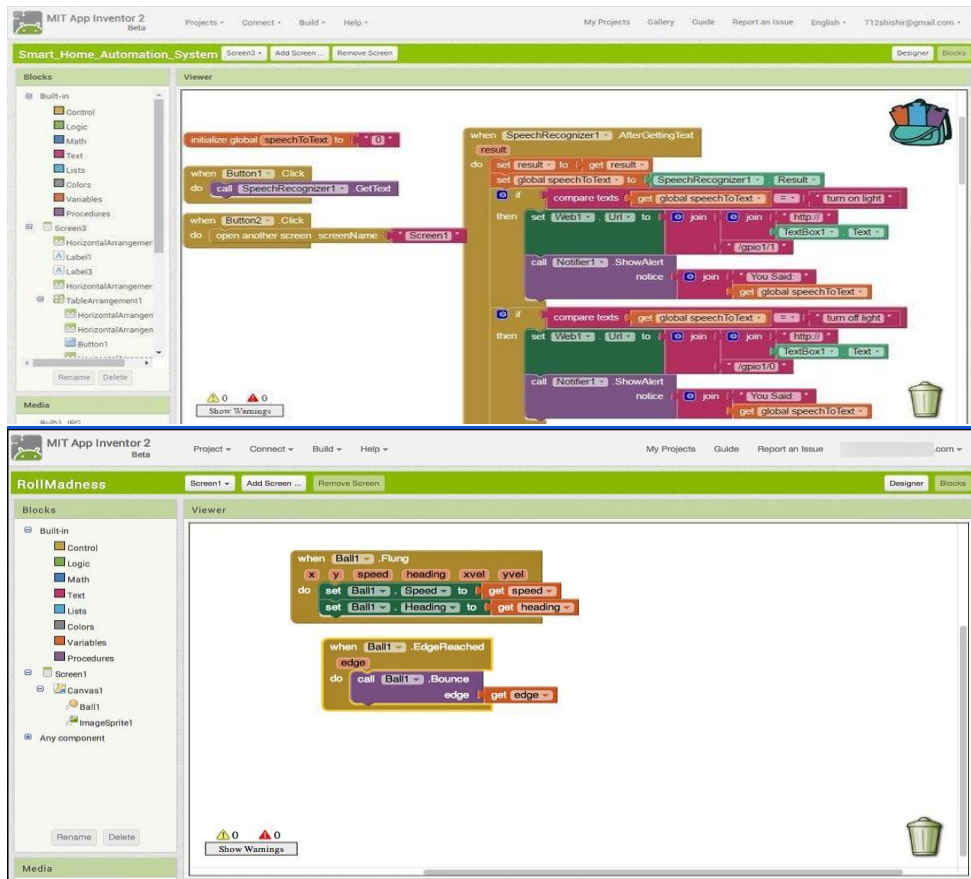
```
 }
}
```
Flutter run

This Flutter code provides a simple app that simulates real-time air quality level updates. To create a complete

app, you would need to Creating a complete, production-ready mobile app involves a significant amount of

work, and it's often a multi-stage development process. You may want to consider working with a mobile app

development team or hiring experienced developers to build the app to your specifications.

Mit app wireframe:

https://appinventor.mit.edu.