Phase 4:
Development of smart public restroom – Part_2

The document consists of two parts: the first part describes the development of an IoT-based Smart Restroom Status web page, outlining the HTML structure, CSS styling, and JavaScript code to display occupancy and cleanliness data, while the second part introduces the creation of a mobile app for real-time restroom information using Flutter. It provides instructions to set up a Flutter project, defines a simple user interface, and demonstrates data simulation. The document serves as a high-level guide for developers looking to build both a web page and a mobile app to monitor and display real-time restroom status information, offering a foundation for real data integration in these applications.

This code is for an IoT-based Smart Restroom Status web page. It combines HTML, CSS, and JavaScript to create a webpage that displays the availability and cleanliness status of a restroom. The web page has a title, a header, and a main content area that displays the restroom status.

HTML (index.html):

- The HTML document starts with the usual document type declaration and sets the language to English.
- In the <head> section, there are meta tags for character set and viewport settings, a title for the webpage, and a link to an external CSS file ("styles.css").
- The content of the page is wrapped in the <body> tag.
- Inside the <body>, there is a <header> element with the title of "Smart Restroom Status."
- The main content is in a <main> element, which contains a <section> with the id "restroom-status."
- Within the section, there are two elements: "availability" and "cleanliness." Each contains a heading and a paragraph element with placeholders ("Loading...").

- Finally, there is a <script> tag at the end that includes an external JavaScript file ("script.js").

CSS (styles.css):

- The CSS file provides styling for the webpage.
- The body has a background color, and the font is set to Arial or sans-serif. Margins and padding are set to zero to remove default spacing.
- The header has a dark background color, white text, and is centered with padding.
- The main content has a maximum width, margin, and padding to center it on the page.
- Various elements are styled using CSS selectors, including headings and specific IDs like "restroom-status," "availability," and "cleanliness."

JavaScript (script.js):

- This JavaScript code runs after the DOM (Document Object Model) is fully loaded, as it's wrapped in a 'DOMContentLoaded' event listener. This ensures that the JavaScript code doesn't execute until the HTML is ready.
- It selects the HTML elements with the IDs "occupancy" and "cleanliness-status" and stores them in variables.
- There is a function called 'updateUI' that takes two parameters, "occupancy" and "cleanliness." This function updates the text content of the corresponding HTML elements with the provided data.

- A simulated data object is created, mimicking data that might be obtained from IoT sensors. In a real scenario, you would fetch this data from sensors using AJAX or fetch requests to your server.

- The simulated data is used to call the 'updateUI' function after a 2-second delay, updating the occupancy and cleanliness status on the webpage.

In a real-world scenario, you would replace the simulated data with actual data from IoT sensors in your restroom, making this webpage a real-time representation of restroom status.

Program:

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Restroom Status</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Smart Restroom Status</h1>
  </header>
  <main>
    <section id="restroom-status">
      <h2>Restroom Availability and Cleanliness</h2>
      <div id="availability">
        <h3>Availability</h3>
        <p id="occupancy">Loading...</p>
      </div>
      <div id="cleanliness">
        <h3>Cleanliness</h3>
        <p id="cleanliness-status">Loading...</p>
```

```
        </div>
      </section>
    </main>
    <script src="script.js"></script>
</body>
</html>
```

CSS

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f0f0f0;
}

header {
    background-color: #333;
    color: white;
    text-align: center;
    padding: 1rem;
}

main {
    max-width: 800px;
    margin: 0 auto;
    padding: 2rem;
}

h2 {
    font-size: 24px;
}

#restroom-status {
    background-color: white;
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 1rem;
```

```
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

#availability, #cleanliness {
  margin: 1rem 0;
}

#occupancy, #cleanliness-status {
  font-size: 18px;
  font-weight: bold;
}
```

JavaScript

```
document.addEventListener("DOMContentLoaded", () => {
  const occupancyElement = document.getElementById("occupancy");
  const cleanlinessElement = document.getElementById("cleanliness-status");

  // Function to update the UI with sensor data
  const updateUI = (occupancy, cleanliness) => {
    occupancyElement.textContent = `Occupancy: ${occupancy}`;
    cleanlinessElement.textContent = `Cleanliness: ${cleanliness}`;
  };

  // Simulate fetching sensor data (replace with real data)
  // In a real scenario, you would make an AJAX or fetch request to your server to get sensor
data.
  setTimeout(() => {
    const simulatedData = {
      occupancy: "Occupied",
      cleanliness: "Clean",
    };
    updateUI(simulatedData.occupancy, simulatedData.cleanliness);
  }, 2000); // Simulate a 2-second delay for data retrieval
});
```

Mobile App

Building a mobile app for both iOS and Android to provide users with access to real-time restroom information requires using a cross-platform framework or developing separate apps for each platform. Here's a high-level overview of how you can create a cross-platform app using a popular framework called Flutter, developed by Google:

Prerequisites:
1. Install Flutter: Follow the official [Flutter installation guide](https://flutter.dev/docs/get-started/install) to set up your development environment.

1. Create a New Flutter Project:
Use the following commands to create a new Flutter project:

```bash
flutter create restroom_info_app
cd restroom_info_app
```

2. Define the User Interface:
Open the `lib/main.dart` file and define the user interface. For this example, we'll create a simple app with a button to refresh restroom information and display the data.

```dart
import 'package:flutter/material.dart';

void main() {
 runApp(RestroomInfoApp());
}

class RestroomInfoApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   home: RestroomInfoScreen(),
  );
 }
}
```

```
class RestroomInfoScreen extends StatefulWidget {
 @override
 _RestroomInfoScreenState createState() => _RestroomInfoScreenState();
}

class _RestroomInfoScreenState extends State<RestroomInfoScreen> {
 String occupancyStatus = "Loading...";
 String cleanlinessStatus = "Loading...";

 // Function to fetch and update restroom information
 void refreshRestroomInfo() {
  // Simulate fetching data from your server or IoT sensors
  setState(() {
   occupancyStatus = "Occupied"; // Replace with real data
   cleanlinessStatus = "Clean"; // Replace with real data
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Restroom Info App'),
   ),
   body: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: <Widget>[
      Text('Restroom Availability: $occupancyStatus'),
      Text('Cleanliness Status: $cleanlinessStatus'),
      ElevatedButton(
       onPressed: refreshRestroomInfo,
       child: Text('Refresh'),
      ),
     ],
    ),
   ),
```

```
  );
 }
}
```

3. Test the App:
Connect your Android or iOS device or use an emulator/simulator. Run the app using the following command:

```bash
flutter run
```

This will launch the app on your connected device or emulator.

4. Real Data Integration:
In a production scenario, you would replace the simulated data in the `refreshRestroomInfo` function with actual data fetched from your server or IoT sensors. You may use HTTP requests or WebSocket connections to receive real-time data updates.

5. Build for iOS and Android:
You can build your Flutter app for both iOS and Android using the following commands:

- For Android:
  ```bash
  flutter build apk
  ```

- For iOS:
  ```bash
  flutter build ios
  ```

You'll find the generated APK file for Android in the `build/app/outputs/flutter-apk` directory and the iOS project in the `ios` directory.

Program:

```
import 'package:flutter/material.dart';

void main() {
 runApp(RestroomInfoApp());
}

class RestroomInfoApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   home: RestroomInfoScreen(),
  );
 }
}

class RestroomInfoScreen extends StatefulWidget {
 @override
 _RestroomInfoScreenState createState() => _RestroomInfoScreenState();
}

class _RestroomInfoScreenState extends State<RestroomInfoScreen> {
 String occupancyStatus = "Loading...";
 String cleanlinessStatus = "Loading...";

 // Function to fetch and update restroom information
 void refreshRestroomInfo() {
  // Simulate fetching data from your server or IoT sensors
  setState(() {
   occupancyStatus = "Occupied"; // Replace with real data
   cleanlinessStatus = "Clean"; // Replace with real data
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Restroom Info App'),
```

```
    ),
    body: Center(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
       Text('Restroom Availability: $occupancyStatus'),
       Text('Cleanliness Status: $cleanlinessStatus'),
       ElevatedButton(
        onPressed: refreshRestroomInfo,
        child: Text('Refresh'),
       ),
      ],
     ),
    ),
   );
  }
}
```