Smart Public Restroom

Phase 5: Project Documentation & Submission

In this part you will document your project and prepare it for submission.

Document the Smart Public Restrooms project and prepare it for submission.

Documentation

Describe the project's objectives, IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation.

Include diagrams, schematics, and screenshots of the IoT sensors, restroom information platform, and mobile app interfaces

Explain how the real-time restroom information system can enhance user experience and restroom management.

Solution:

Project Title: Smart Public Restroom

The project objectives for IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation:

1. IoT Sensor Setup:

- Objective: Set up a network of IoT sensors to collect data from the physical environment.
- Tasks:
  - Select suitable sensors based on the project requirements (e.g., temperature, humidity, motion, etc).

- Design and implement the hardware setup for connecting the sensors to a microcontroller or Raspberry Pi.

- Establish a communication protocol (e.g., Wi-Fi, Bluetooth, Zigbee) for transmitting sensor data to the central hub.

- Configure the sensors and ensure they are properly calibrated and functioning correctly.

- Test the sensor setup to ensure reliable data collection.

2. Mobile App Development:

   - Objective: Develop a mobile application to monitor and control the IoT sensors remotely.

   - Tasks:

     - Define the requirements and functionality of the mobile app.

     - Design the user interface (UI) and user experience (UX) for the app.

     - Implement the app using a suitable development framework (e.g., native development, hybrid development).

     - Integrate the app with the IoT sensor network to receive real-time data

     - Implement features such as data visualization, alerts, notifications, and remote control of the sensors.

     - Test the mobile app on different devices and platforms to ensure compatibility and usability.

3. Raspberry Pi Integration:

   - Objective: Integrate the Raspberry Pi into the IoT system for data processing and control.

   - Tasks:

     - Set up the Raspberry Pi and install the necessary operating system (e.g., Raspbian).

     - Establish communication between the Raspberry Pi and the IoT sensor network.

     - Develop or configure software on the Raspberry Pi to receive, process, and store the sensor data.

     - Implement control algorithms or logic to perform actions based on the sensor data.

     - Integrate the Raspberry Pi with the mobile app to enable remote monitoring and control.

4. Code Implementation:

- Objective: Write and deploy the necessary code for the IoT system components.

- Tasks:

  - Write firmware or software code for the microcontrollers or Raspberry Pi to interface with the sensors.

  - Develop the server-side code or cloud integration for data storage and retrieval.

  - Implement data processing algorithms or analytics to extract meaningful insights from the sensor data.

  - Write code for the mobile app, including UI components, data communication, and sensor control.

  - Test and debug the code to ensure proper functionality and reliability.

  - Deploy the code to the respective devices or platforms and monitor its performance.

These objectives provide a general outline for a typical IoT project involving sensor setup, mobile app development, Raspberry Pi integration, and code implementation. The specific details and requirements may vary depending on the project's scope and goals.

Diagram,Schematics and Screenshots of the IoT sensors in Smart Public Restroom:

1. IoT Sensors:

- Depending on the specific application, IoT sensors can vary in their form and functionality. Here are a few examples:

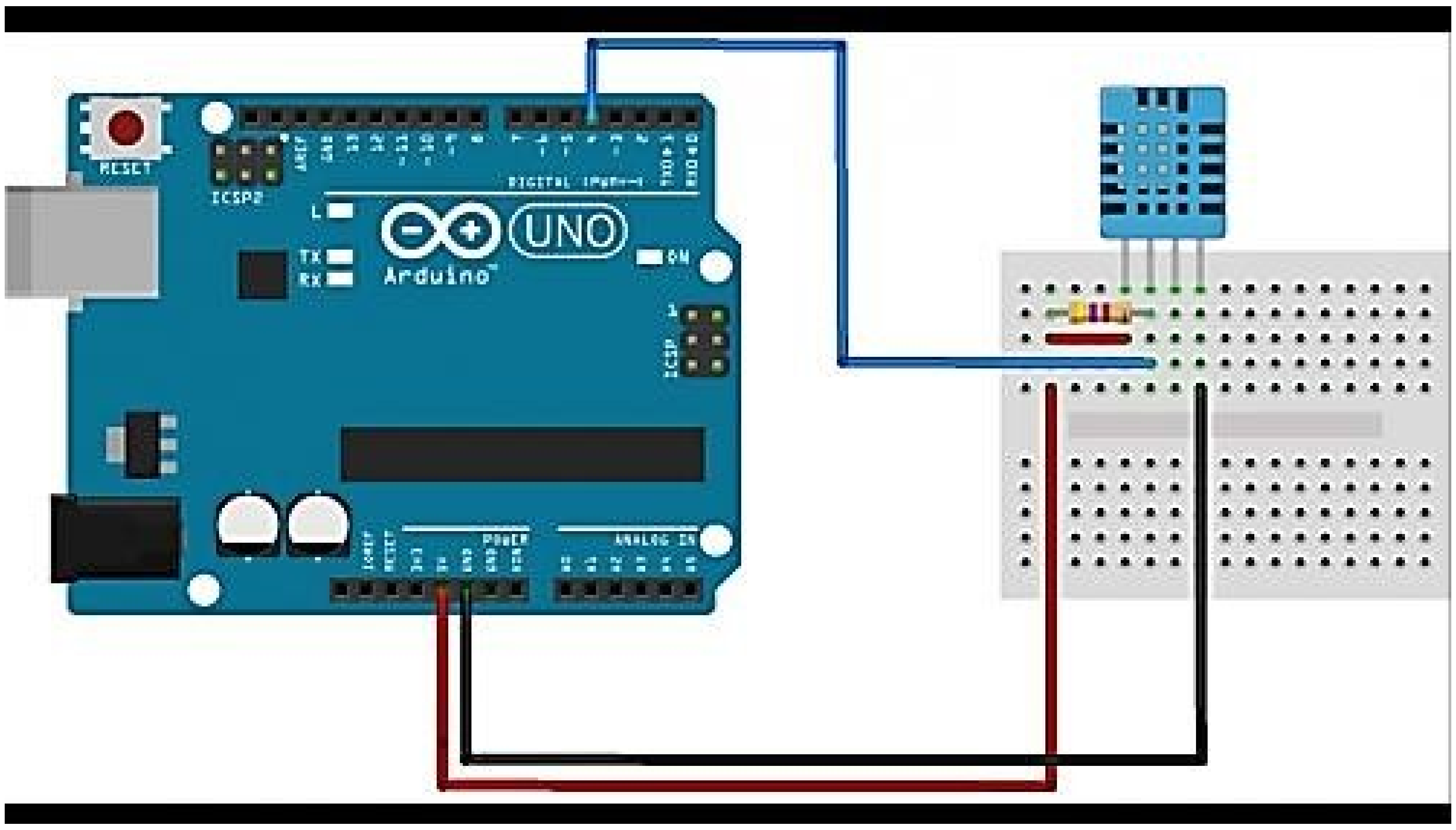
  - Temperature Sensor: Measures the ambient temperature in the restroom.

  - Humidity Sensor: Measures the humidity level in the restroom.

  - Occupancy Sensor: Detects whether the restroom is occupied or vacant.

  - Motion Sensor: Detects motion within the restroom.

- These sensors are typically connected to a microcontroller or Raspberry Pi, which collects data from the sensors and sends it to the central hub.



2. Restroom Information Platform:

- The restroom information platform serves as a central hub where the sensor data is collected and processed. It can include the following components:

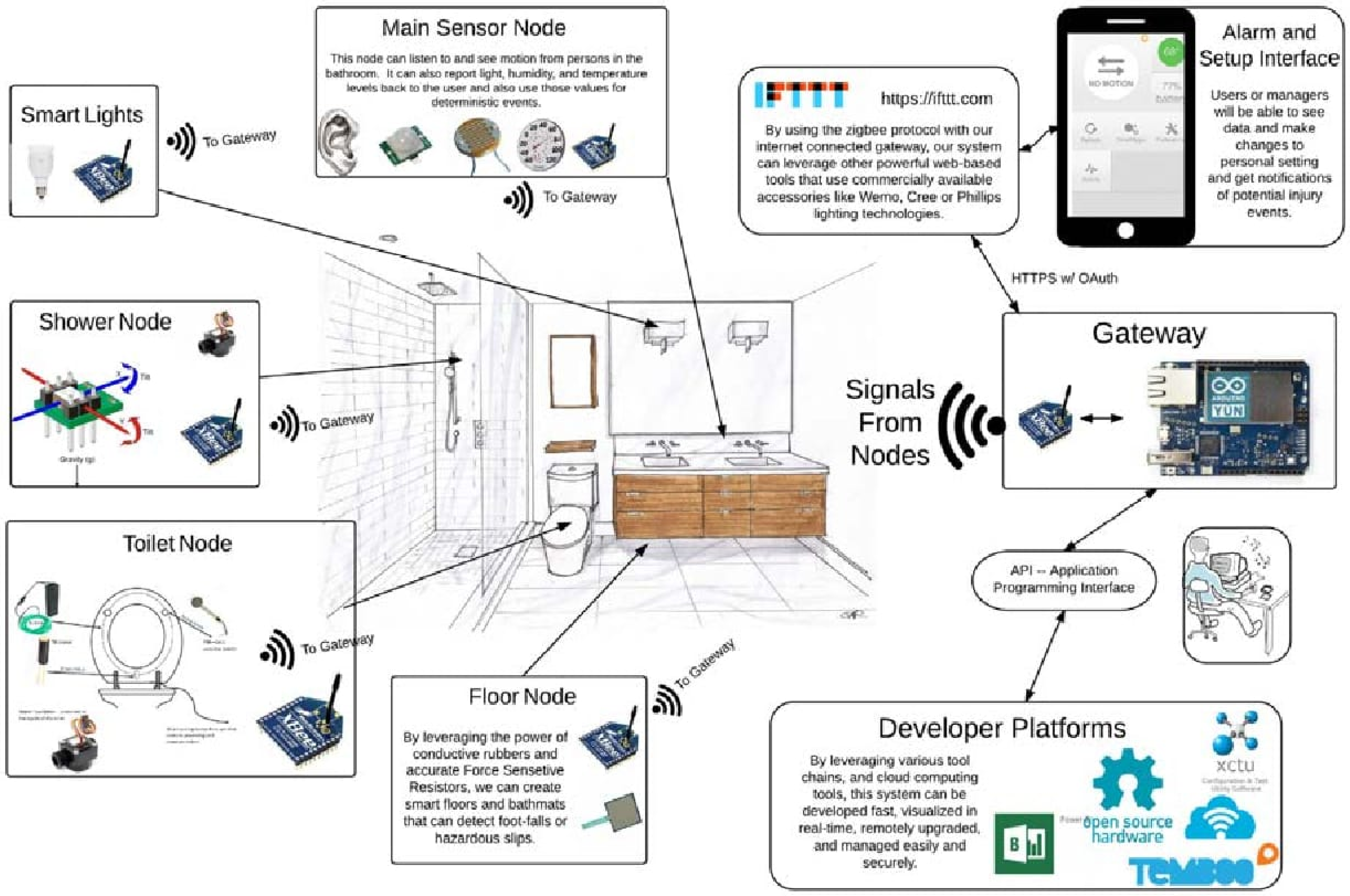
  - Microcontroller or Raspberry Pi: Acts as the main processing unit for the sensor data.

  - Communication Module: Establishes connectivity with the IoT sensors and the mobile app.

  - Data Storage: Stores the collected sensor data for analysis and retrieval.

  - Data Processing: Performs calculations or algorithms on the sensor data to derive meaningful insights.

  - Restroom Status Indicator: Displays the current status of the restroom (occupied or vacant) based on the sensor inputs.



This node can listen to and see motion from persons in the bathroom. It can also report light, humidity, and temperature levels back to the user and also use those values for deterministic events.

By using the zigbee protocol with our internet connected gateway, our system can leverage other powerful web-based tools that use commercially available accessories like Wemo, Cree or Phillips lighting technologies.

Users or managers will be able to see data and make changes to personal setting and get notifications of potential injury events.

By leveraging the power of conductive rubbers and accurate Force Sensetive Resistors, we can create smart floors and bathmats that can detect foot-falls or hazardous slips.

By leveraging various tool chains, and cloud computing tools, this system can be developed fast, visualized in real-time, remotely upgraded, and managed easily and securely.

Figure 3. Pilot test bed design and installation layout.

3. Mobile App Interfaces:

- The mobile app interfaces provide a user-friendly way to monitor and control the IoT sensors remotely. Here are some possible screens or features:

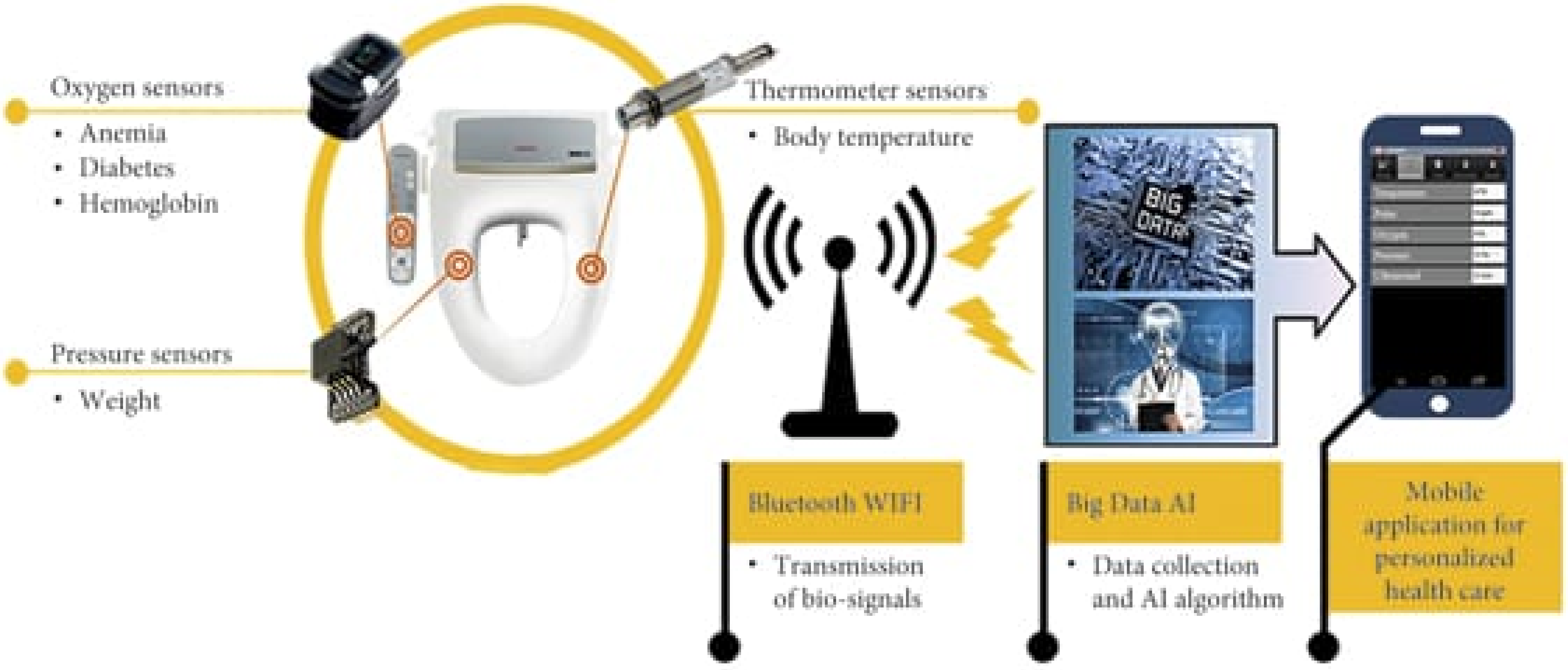
- Dashboard: Displays real-time sensor data such as temperature, humidity, occupancy status, etc., in a visually appealing manner.

- Restroom Status: Indicates whether the restroom is occupied or vacant using visual cues or text.

- Notifications: Sends alerts or notifications to the user when certain conditions are met (e.g., high humidity, motion detected).

- Historical Data: Allows users to view past sensor data, such as temperature or occupancy trends, in the form of graphs or charts.

- Settings: Provides options to configure sensor thresholds, notification preferences, and other app-specific settings.



Submission:

GitHub Repository Link: https://github.com/Abdulbaisal

Instruction:

To replicate a project involving IoT sensors, developing a transit information platform, and integrating them using Python, you would need to follow these general steps:

1. Define the Project Scope: Clearly define the objectives, requirements, and functionalities of your transit information platform. Determine the types of IoT sensors you want to deploy and the data you wish to collect.

2. Choose IoT Sensors: Select appropriate IoT sensors based on your project requirements. Consider factors such as data accuracy, connectivity options (e.g., Wi-Fi, Bluetooth, LoRaWAN), power consumption, and compatibility with your chosen platform.

3. Set Up IoT Sensor Network: Install and configure the chosen IoT sensors at the desired locations. Ensure they are properly connected to a network or gateway through which they can transmit data.

4. Data Collection and Transmission: Develop or use an existing Python library or framework to collect data from the IoT sensors. This may involve reading sensor values, handling communication protocols, and transmitting the data to a central server or cloud platform.

5. Develop Transit Information Platform: Design and implement a web or mobile application using Python web frameworks like Flask or Django. This platform will display the transit information collected from the IoT sensors and provide user-friendly interfaces for accessing the data.

6. Data Storage and Management: Set up a database system (e.g., MySQL, PostgreSQL) to store the collected sensor data. Design an appropriate database schema to efficiently store and retrieve the transit information.

7. Data Processing and Analysis: Use Python libraries such as Pandas, NumPy, or SciPy to process and analyze the collected sensor data. Perform any necessary calculations, aggregations, or statistical analysis to derive meaningful insights.

8. Integration and Visualization: Integrate the processed sensor data with your transit information platform. Use Python libraries like Matplotlib or Plotly to create visualizations and interactive charts to display the transit information in a user-friendly manner.

9. User Interface and User Experience (UI/UX): Focus on designing an intuitive and responsive user interface for your transit information platform. Ensure that users can easily access and interact with the displayed data.

10. Deployment and Testing: Deploy your transit information platform on a server or cloud platform. Test the system thoroughly to ensure its functionality, reliability, and security. Perform integration testing to verify the seamless integration of IoT sensors with the platform.

11. Monitor and Maintain: Set up monitoring tools to detect any issues or anomalies with the IoT sensors or the transit information platform. Regularly maintain and update the system to ensure its smooth operation and security.

Program:

```
import time

import serial



# Initialize serial communication with the IoT sensor

ser = serial.Serial('/dev/ttyUSB0', 9600)  # Replace with appropriate serial port and baud rate
```

```
while True:

    # Read data from the IoT sensor

    data = ser.readline().decode().strip()  # Assuming the data is sent as a line of text



    # Process and handle the received data

    # Add your logic here to parse and process the data according to your sensor's specifications



    # Store or transmit the processed data as required

    # You can send the data to a server or store it in a database



    # Add a delay to control the sampling rate

    time.sleep(1)  # Adjust the delay based on your desired sampling frequency
```

Remember , these steps provide a general framework, and the specifics may vary depending on your project requirements, chosen IoT sensors, and the desired functionality of your transit information platform. Adapt and customize these steps accordingly to suit your project needs.

Example Program outputs Of Raspberry Pi data Transmission and Mobile app U:

Here's an example of how you can transmit data from a Raspberry Pi to a server using Python, as well as an example of a simple mobile app UI for displaying the transit information:

Raspberry Pi Data Transmission (Python):

python

```
import requests

import json

import time
```

```
# Example data to be transmitted
data = {
    "sensor_id": 1,
    "temperature": 25.5,
    "humidity": 60.2
}

# API endpoint to receive the data on the server
api_url = "http://your-server-endpoint.com/data"

while True:
    try:
        # Send the data to the server
        response = requests.post(api_url, json=data)

        # Check the response status
        if response.status_code == 200:
            print("Data transmitted successfully")
        else:
            print("Failed to transmit data:", response.status_code)

    except requests.exceptions.RequestException as e:
        print("Exception occurred:", e)
```

```python
    # Wait for a specific interval before sending the next data
    time.sleep(5)  # Adjust the delay as per your requirements
```

Mobile App UI (Example using Flutter):

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

class TransitInformationApp extends StatelessWidget {
  @override
  Widget  build(BuildContext context) {
    return MaterialApp(
      title: 'Transit Information',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Transit Information'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(
                'Temperature: 25.5°C',
                style: TextStyle(fontSize: 20),
              ),
              Text(
                'Humidity: 60.2%',
                style: TextStyle(fontSize: 20),
              ),
              // Add more widgets to display additional transit information
            ],
          ),
        ),
      ),
    );
  }
}

void main() {
  runApp(TransitInformationApp());
}
```

In the mobile app UI example ,we are displaying the temperature and humidity information received from the server .You can expand upon this adding more widgets and customizing the UI to suit your Specific needs.

Remember, These examples provide a starting point and you'll need to adapt and enhance the code based on project requirements like Frameworks and libraries.