

SMART PUBLIC RESTROOM

Phase-5: Project Documentation

Introduction:

The "Smart Public Restroom Management System" is an innovative solution designed to enhance the efficiency and user experience of public restrooms. Leveraging the power of IOT (Internet of Things) technology and a user-friendly web application built with React, this project offers a comprehensive and intelligent restroom management system.

In today's fast-paced world, the demand for clean and accessible public restrooms is ever-increasing. However, managing these facilities can be a challenging task. This project addresses these challenges by providing real-time monitoring, control, and user feedback through web application.

Key Features:

- 1. Real-time Monitoring:** Sensors within each restroom continuously monitor occupancy, cleanliness, and supply levels. This data is instantly accessible through the web application.
- 2. Smart Occupancy Management:** Users can check the availability of restrooms through the web application before visiting a location, minimizing waiting times.
- 3. Automated Maintenance Alerts:** The system sends maintenance alerts when restrooms require cleaning or supply replenishment, ensuring a consistent level of service.
- 4. User Feedback:** Visitors can rate and review the cleanliness and overall experience of each restroom, allowing for immediate improvements.
- 5. Sustainability:** The system promotes eco-friendliness by reducing water and energy wastage through smart controls.

- 6. Web Application:** The React-based web application offers an intuitive interface for users to find and review restrooms, while administrators can manage and monitor multiple locations seamlessly.

Smart Public Restroom Information Platform:

1. Dashboard:

- Provides an overview of multiple public restrooms.
- Lists the available restrooms, their status, cleanliness, and supply levels.

2. Individual Restroom Details:

- Users can click on a specific restroom for more information.
- Real-time status (e.g., open or closed) and occupancy information.

3. Cleanliness Monitoring:

- Visualizes the cleanliness of each restroom, with real-time updates.
- May include ratings from previous users.

4. Supply Management:

- Shows supply levels (e.g., toilet paper, soap, hand sanitizer).
- Alerts facility managers when supplies are low.

5. User Feedback:

- Allows users to rate and comment on the cleanliness and service.
- Provides a feedback loop for continuous improvement.

MOBILE APPS:

1. Real-time Restroom Status:

- Displays the current status of nearby public restrooms (open/closed).
- Utilizes IOT occupancy sensors for real-time updates.

2. Cleanliness Information:

- Shows cleanliness ratings and information for each restroom.
- Allows users to provide ratings and comments.

3. Supply Levels:

- Indicates the availability of essential supplies (toilet paper, soap, hand sanitizer).
- Notifies users if supplies are running low.

4. Navigation:

- Offers directions to the nearest smart public restroom.

5. IOT Integration:

- Integrates with IOT sensors for occupancy and cleanliness data.
- Uses sensors for real-time monitoring.

6. Push Notifications:

- Notifies users about restroom status changes, cleanliness updates, or low supply alerts.

7. Security and Privacy:

- Implements user authentication and data encryption to protect user information.

8. Encryption:

- Use strong encryption protocols (e.g., HTTPS) to protect data in transit between IOT devices, the app, and the back-end server.

9. Data Analytics:

- Collects and analyses historical data for insights and trends.
- Helps facility managers make informed decisions.

10. Accessibility:

- Ensures that the app is accessible to people with disabilities.

11. Social Sharing:

- Allows users to share their restroom experience on social media.

12. Multiple Languages:

- Supports multiple languages for a diverse user base.

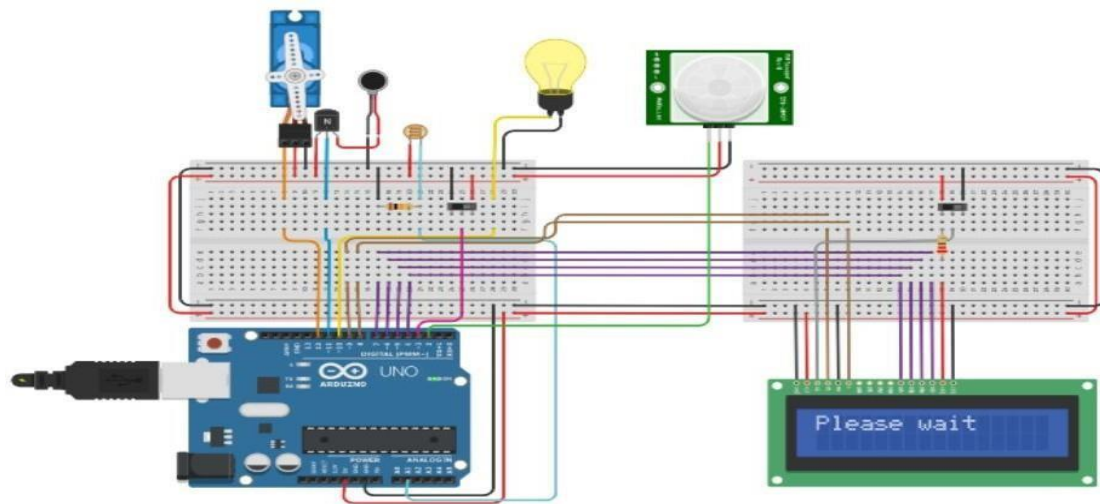
Hardware Implementation:

1. **Simulator used:** Wokwi Simulator

2. **Components used:**

Name	Quantity	Component
U2	1	Arduino Uno R3
PIR1	1	-39.42235927868387 , -193.319557022722 , -235.36724515923254 PIR Sensor
L1	1	Light bulb
R1	1	Photoresistor
M3	1	Vibration Motor
SERVO1	1	Positional Micro Servo
R2	1	10 k Ω Resistor
U3	1	LCD 16 x 2
R3	1	220 Ω Resistor
S1 S2	2	Slideswitch
T1	1	NPN Transistor (BJT)

3. Circuit Diagram:



4. Working :

The provided Python code is designed to simulate a simple system using a Raspberry Pi that controls a servo motor and an LED based on the status of a push-button switch, simulating a restroom's occupancy status. Here's how the code works:

Initialization (Setup):

- It imports the necessary libraries: RPi.GPIO and time.
- It defines the GPIO (General-Purpose Input/Output) pins used for the button, LED, and servo motor.
- It sets the GPIO mode to use the Broadcom SOC channel numbering.

- It configures the buttonPin as an input and ledPin as an output.
- It initializes the servo motor by configuring the servoPin and setting up a PWM (Pulse Width Modulation) object with a frequency of 50 Hz. The PWM signal will be used to control the servo motor's position.
- It starts the servo motor at its initial position (0 degrees)

Loop:

- The code enters a continuous loop using a while loop with True as the condition.

Button State Detection:

- Inside the loop, it reads the state of the buttonPin using GPIO.input(buttonPin) and stores it in the buttonState variable.

Restroom Status Control:

- If the button is pressed (buttonState is HIGH), it considers the restroom as occupied.
- It turns on the LED (sets ledPin to HIGH) to indicate that the restroom is occupied.
- It uses PWM to set the servo motor to a specific duty cycle (7.5%) to open the door (rotate the servo to 90 degrees).

Restroom Vacant:

- If the button is not pressed (buttonState is LOW), it considers the restroom as vacant.
- It turns off the LED (sets ledPin to LOW) to indicate that the restroom is vacant.
- It uses PWM to set the servo motor to a different duty cycle (2.5%) to close the door (return the servo to 0 degrees).

Delay:

- The code includes a small delay of 0.1 seconds using `time.sleep(0.1)` to prevent rapid toggling of the servo and LED due to switch bounce.

Cleanup on KeyboardInterrupt:

- The code is designed to handle a KeyboardInterrupt (Ctrl+C) gracefully. When the user interrupts the program, it stops the servomotor and cleans up the GPIO pins using `servo.stop()` and `GPIO.cleanup()`, respectively.
- In summary, the code continuously monitors the state of a button switch. When the button is pressed, it turns on an LED and rotates a servo to open a door, simulating an occupied restroom. When the button is released, it turns off the LED and closes the door by returning the servo to its initial position, simulating a vacant restroom. The code keeps running in a loop until the user interrupts it, and it ensures proper cleanup of resources upon exit.

USING A WEB DEVELOPMENT:

Creating a complete smart public restroom using IOT webpage requires a substantial amount of code, and it can be quite complex. However, I can provide you with a simplified example of an HTML, CSS, and JavaScript template for a webpage that displays restroom information. You would need to integrate this with IOT devices and server-side logic for real-world functionality.

Sample Code:**HTML:**

We have a simple interface for viewing real-time traffic information.

```
<!DOCTYPE html>

<html>

<head>

<title>Smart Restroom Dashboard</title>

<style>

/* Add your CSS styling here */

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 800px;

margin: 0 auto;

padding: 20px;

}

/* Add more CSS as needed */

</style>

</head>

<body>

<div class="container">

<h1>Smart Restroom Dashboard</h1>

<h2>Restroom Information</h2>
```



```
<p>Status: <span id="restroom Status">Loading...</span></p>
<p>Cleanliness: <span id="cleanliness">Loading...</span></p>
<p>Supply Levels: <span id="supply Levels">Loading...</span></p>
</div>
```

```
<script>
```

```
// Add your JavaScript code to fetch IoT data and update the webpage
```

```
function updateRestroomData() {
```

```
// Simulated IOT data (replace with actual data retrieval)
```

```
const restroomData = {
```

```
status: 'Open',
```

```
cleanliness: 'Good',
```

```
supplyLevels: 'Adequate'
```

```
};
```

```
// Update the webpage elements with IoT data
```

```
document.getElementById('restroomStatus').textContent      =
restroomData.status;
```

```
document.getElementById('cleanliness').textContent          =
restroomData.cleanliness;
```

```
document.getElementById('supplyLevels').textContent          =
restroomData.supplyLevels;
```

```
}
```

```
// Periodically update the data (e.g., every 30 seconds)
```

```
setInterval(updateRestroomData, 30000);
```

```
// Initial data update
```

```
updateRestroomData();
```

```
</script>
```

```
</body>
```

```
</html>
```

CSS:

Create a stylesheet (styles.css) to define the layout and styling of your web page.

```
/* Reset some default browser styles */
```

```
body, h1, h2, p {
```

```
margin: 0;
```

```
padding: 0;
```

```
}
```

```
/* Set a background color and text color */
```

```
body {
```

```
background-color: #f0f0f0;
```

```
color: #333;
```

```
}
```

```
/* Center the content */
```

```
.container {  
max-width: 800px;  
margin: 0 auto;  
padding: 20px;  
background-color: #fff;  
box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);  
border-radius: 5px;  
}
```

```
/* Style headings */
```

```
h1 {  
font-size: 24px;  
margin-bottom: 10px;  
}
```

```
h2 {  
font-size: 20px;  
margin-bottom: 10px;  
}
```

```
/* Style data elements */

p {
  font-size: 16px;
  margin-bottom: 15px;
}

/* Update link styles if needed */

a {
  color: #0077cc;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}
```

JavaScript:

JavaScript code snippet that updates the restroom information on your smart public restroom webpage.

```
Javascript function updateRestroomData() {
  // Simulated IoT data (replace with actual data retrieval)
  const restroomData = {
```

```
status: 'Open',
cleanliness: 'Good',
supplyLevels: 'Adequate'
};

// Update the webpage elements with IOT data

document.getElementById('restroomStatus').textContent =
restroomData.status;

document.getElementById('cleanliness').textContent =
restroomData.cleanliness;

document.getElementById('supplyLevels').textContent =
restroomData.supplyLevels;
}

// Periodically update the data (e.g., every 30 seconds)
setInterval(updateRestroomData, 30000);

// Initial data update
updateRestroomData();
```

DESIGNING A MOBILE APP:

Designing a mobile app for a smart public restroom using IoT involves several key considerations. Here's a high-level guide to get you started:

1. Define the Purpose and Features:

- Clearly define the objectives of the mobile app, such as remote restroom monitoring, user experience enhancement, or resource management.

2. User Research:

- Understand the needs and preferences of the restroom users, facility managers, and other stakeholders. This will help you design a user-centered app.

3. IOT Integration:

- Identify the IOT devices and sensors you'll use in the restroom (e.g., occupancy sensors, water quality monitors, or smart dispensers). Ensure they are compatible with your app.

4. Wireframing and Prototyping:

- Create wireframes and prototypes of the app's user interface, considering ease of use and accessibility. Test the user flow and make improvements based on feedback.

5. Real-time Monitoring:

- Design features that allow real-time monitoring of restroom conditions. Users can check restroom availability, cleanliness, and supply levels through the app.

6. Resource Management:

- Implement features for managing restroom resources efficiently. For instance, automatic alerts for refilling supplies or scheduling cleaning based on usage patterns.

7. User Feedback and Ratings:

- Include a feature for users to provide feedback or rate the restroom's cleanliness and service, helping facility managers improve.

8. Security and Privacy:

- Prioritize security to protect IOT data and user information. Implement encryption and authentication measures.

9. Cross-Platform Considerations:

- Decide whether you'll develop native apps for iOS and Android or use cross-platform development tools.

10. IOT Data Visualization:

- Create clear and visually appealing ways to present data from IoT devices in the app. Charts, graphs, and maps can be useful.

11. App Testing:

- Rigorously test the app, ensuring that it functions smoothly with IoT devices. Check for any connectivity issues.

12. Compliance and Regulations:

- Ensure that your app complies with relevant regulations and standards, especially concerning data privacy and IOT device safety.

13. App Launch and Promotion:

- Launch the app on app stores and promote it to users. Consider partnerships with facility owners or local authorities to gain visibility.

14. Feedback and Updates:

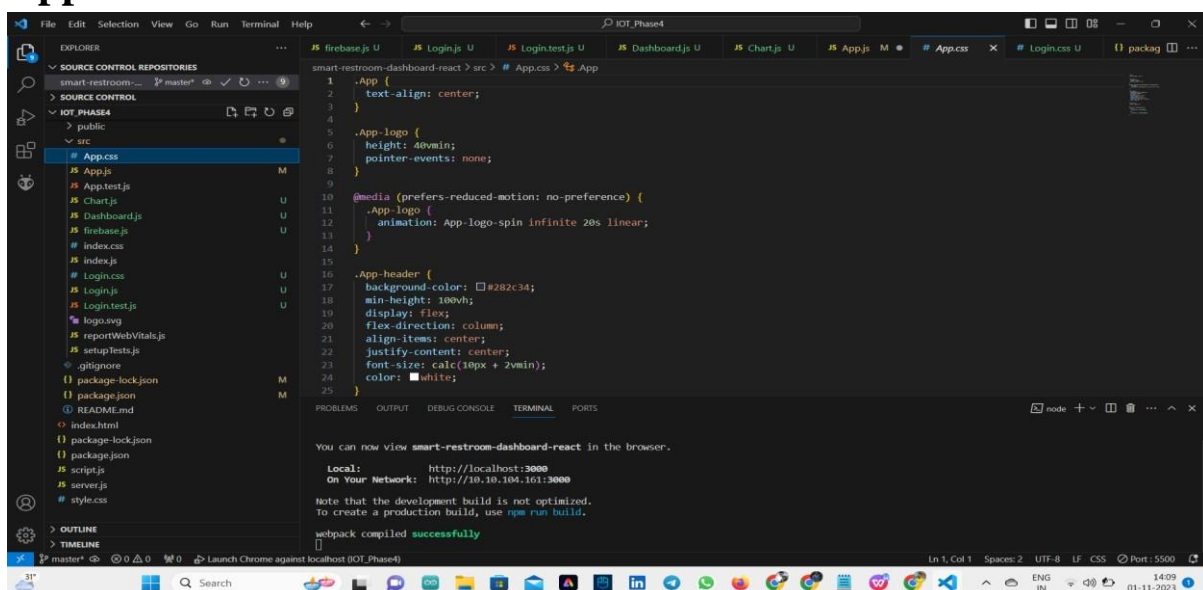
- Continuously gather user feedback and make updates to enhance the app's performance and usability.

15. Maintenance and Support:

- Provide ongoing support and maintenance to address issues and keep the app up to date.

Screenshots and Output of my project:

App.css



```
1 .App {
2   text-align: center;
3 }
4
5 .App-logo {
6   height: 40vmin;
7   pointer-events: none;
8 }
9
10 @media (prefers-reduced-motion: no-preference) {
11   .App-logo {
12     animation: App-logo-spin infinite 20s linear;
13   }
14 }
15
16 .App-header {
17   background-color: #282c34;
18   min-height: 100vh;
19   display: flex;
20   flex-direction: column;
21   align-items: center;
22   justify-content: center;
23   font-size: calc(10px + 2vmin);
24   color: white;
25 }
```

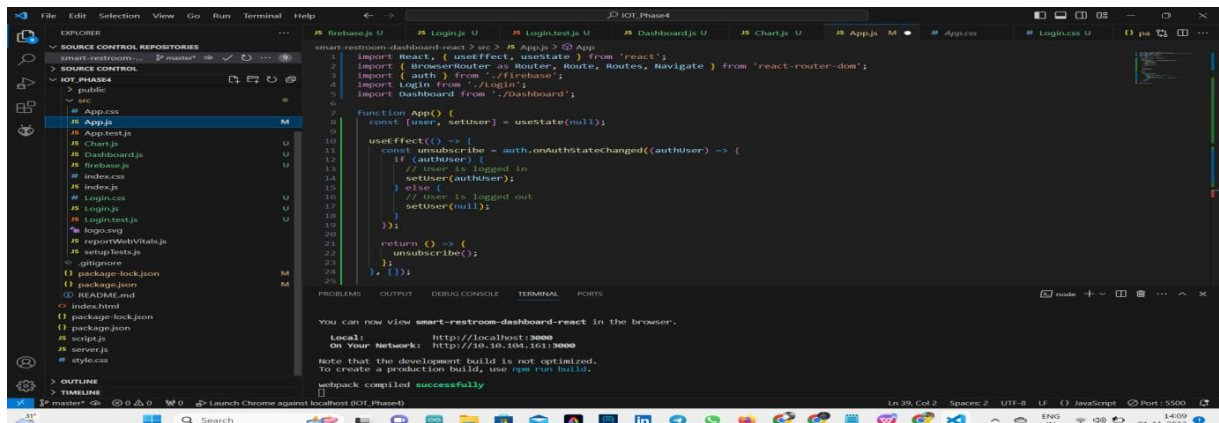
You can now view smart-restroom-dashboard-react in the browser.

Local: <http://localhost:3000>
On Your Network: <http://10.10.104.161:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

App.js



App.test.js

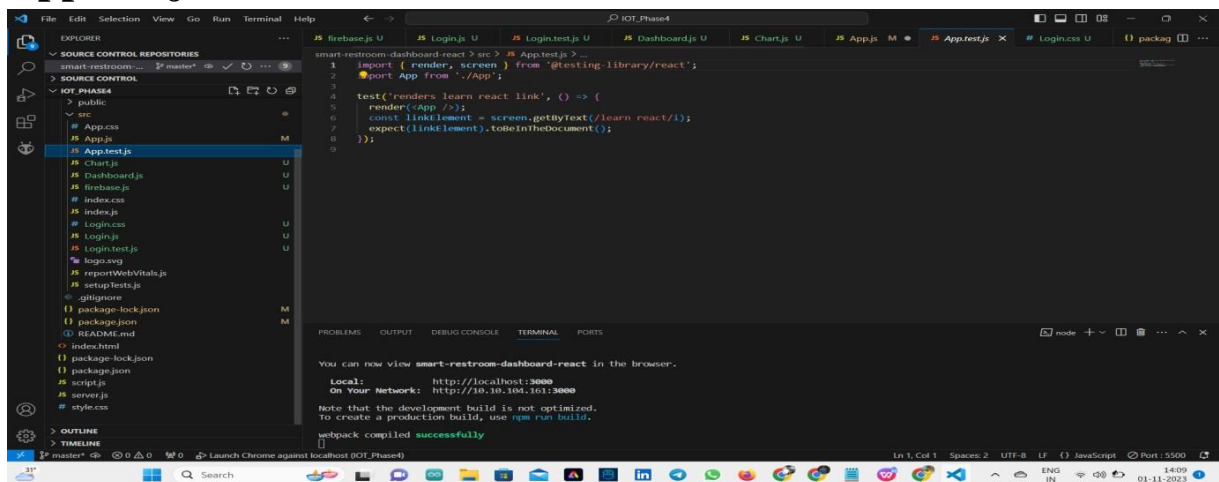
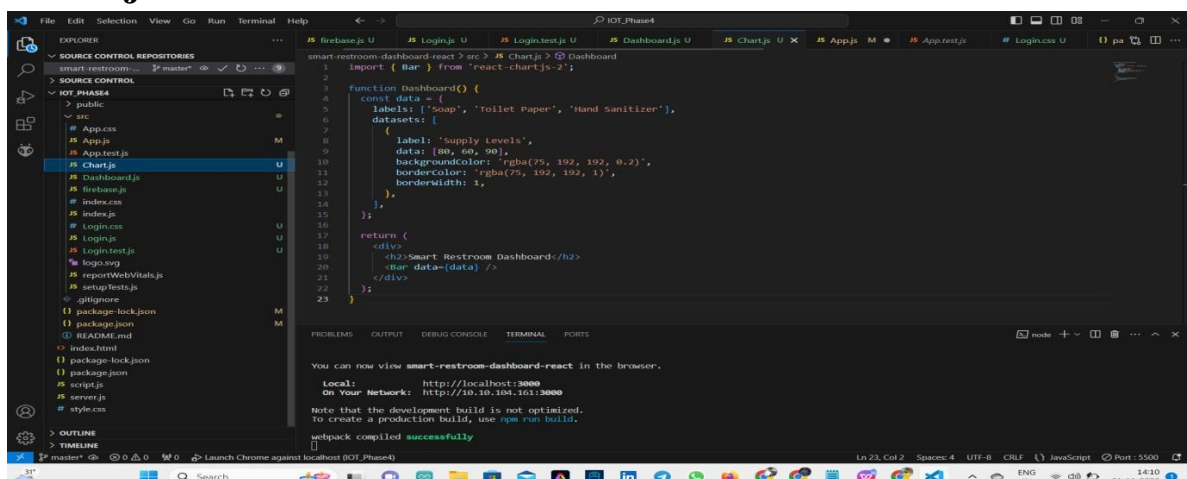


Chart.js



Dashboard.js

```
1 // src/dashboard.js
2 import React from 'react';
3 import { auth } from './firebase';
4
5
6 Function Dashboard() {
7   return (
8     <div>
9       <h2>Smart Restroom Dashboard</h2>
10      <div className="restroom-info">
11        <p>Status: Open</p>
12        <p>Cleanliness: Good</p>
13        <p>Supply Levels: Adequate</p>
14      </div>
15    </div>
16  );
17 }
18
19 const handlesignout = async () => {
20   try {
21     await auth.signout();
22     // Redirect or update the user state as needed
23   } catch (error) {
24     // Handle sign-out error
25     console.error('Error signing out', error.message);
26   }
27 }
```

You can now view smart-restroom-dashboard-react in the browser.

Local: <http://localhost:3000>
On Your Network: <http://10.10.104.161:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

Firebase.js

```
1 // src/firebase.js
2 import { initializeApp } from 'firebase/app';
3 import { getAuth } from 'firebase/auth';
4
5 const firebaseConfig = {
6   apiKey: 'YOUR_API_KEY',
7   authDomain: 'YOUR_AUTH_DOMAIN',
8   projectId: 'YOUR_PROJECT_ID',
9   storagebucket: 'YOUR_STORAGE_BUCKET',
10   messagingSenderId: 'YOUR_MESSAGING_SENDER_ID',
11   appId: 'YOUR_APP_ID',
12 };
13
14 const firebaseApp = initializeApp(firebaseConfig);
15 const auth = getAuth(firebaseApp);
16
17 export { auth };
18
```

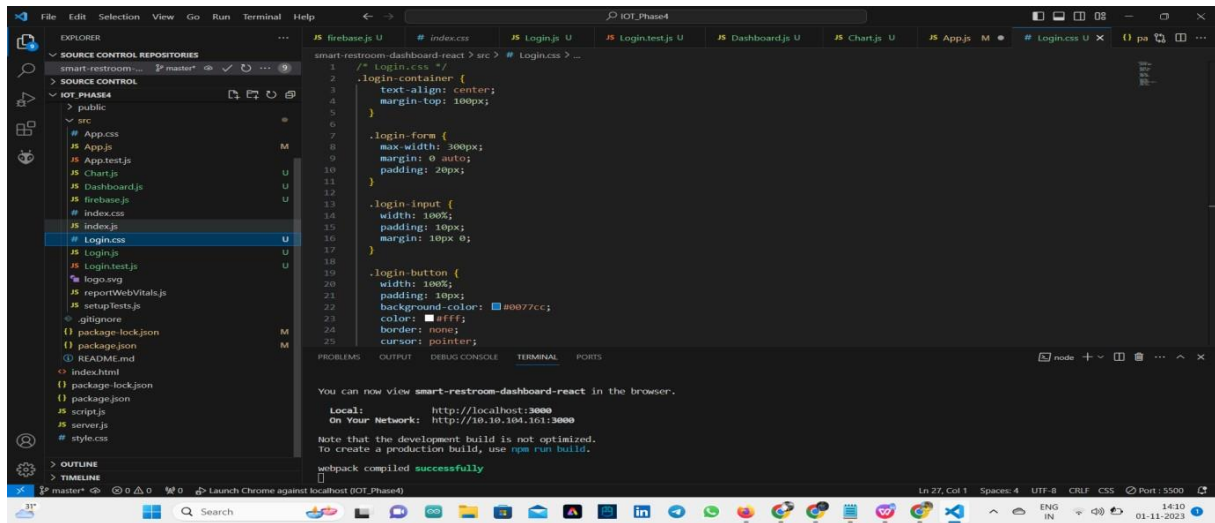
You can now view smart-restroom-dashboard-react in the browser.

Local: <http://localhost:3000>
On Your Network: <http://10.10.104.161:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

Login.css



The screenshot shows the Visual Studio Code editor with the 'Login.css' file open. The file is located in the 'src' directory of a project named 'smart-restroom-dashboard-react'. The code defines styles for a login container, form, input, and button. The terminal shows the development server running on localhost:3000.

```
1 /* Login.css */
2
3 .login-container {
4   text-align: center;
5   margin-top: 100px;
6 }
7
8 .login-form {
9   max-width: 300px;
10  margin: 0 auto;
11  padding: 20px;
12 }
13
14 .login-input {
15   width: 100%;
16   padding: 10px;
17   margin: 10px 0;
18 }
19
20 .login-button {
21   width: 100%;
22   padding: 10px;
23   background-color: #00077cc;
24   color: #fff;
25   border: none;
26   cursor: pointer;
27 }
```

smart-restroom-dashboard-react > src > Login.css > ...

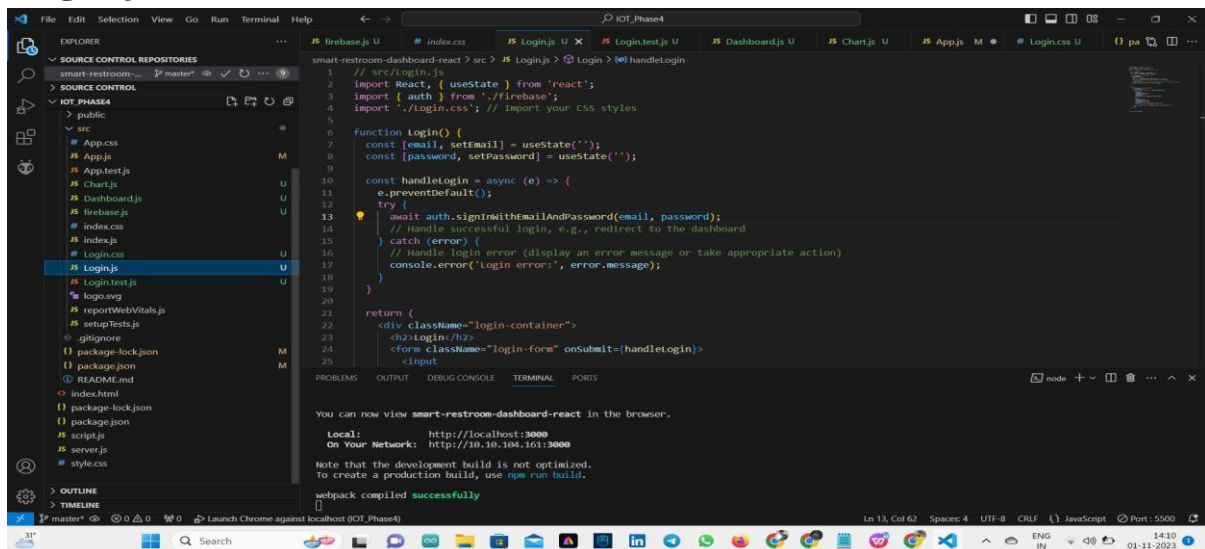
You can now view smart-restroom-dashboard-react in the browser.

Local: http://localhost:3000
On Your Network: http://10.10.104.161:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

Login.js



The screenshot shows the Visual Studio Code editor with the 'Login.js' file open. The file is located in the 'src' directory of a project named 'smart-restroom-dashboard-react'. The code defines a login function that uses Firebase authentication. The terminal shows the development server running on localhost:3000.

```
1 // src/Login.js
2 import React, { useState } from 'react';
3 import { auth } from './firebase';
4 import './Login.css'; // Import your CSS styles
5
6 function Login() {
7   const [email, setEmail] = useState('');
8   const [password, setPassword] = useState('');
9
10  const handleLogin = async (e) => {
11    e.preventDefault();
12    try {
13      await auth.signInWithEmailAndPassword(email, password);
14      // Handle successful login, e.g., redirect to the dashboard
15    } catch (error) {
16      // Handle login error (display an error message or take appropriate action)
17      console.error('Login error:', error.message);
18    }
19  }
20
21  return (
22    <div className="login-container">
23      <h2>Login</h2>
24      <form className="login-form" onSubmit={handleLogin}>
25        <input
```

smart-restroom-dashboard-react > src > Login.js > Login > handleLogin

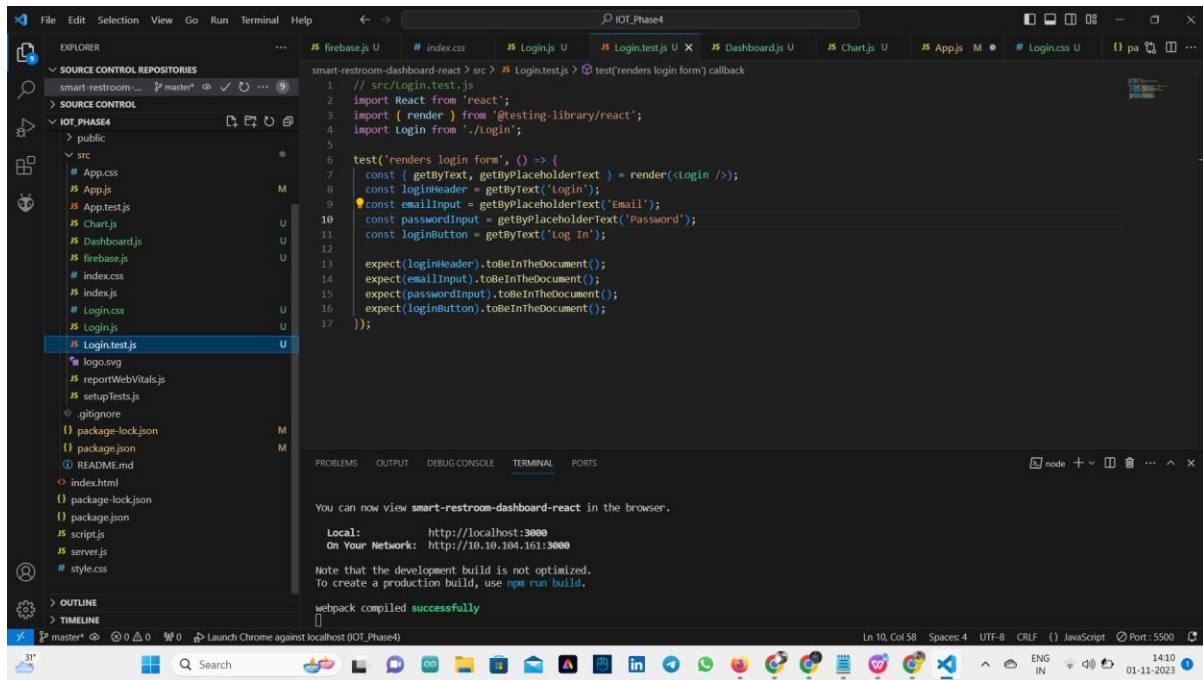
You can now view smart-restroom-dashboard-react in the browser.

Local: http://localhost:3000
On Your Network: http://10.10.104.161:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

Login.test.js



The screenshot shows the VS Code editor with the `Login.test.js` file open. The file contains a Jest test for the login form. The terminal at the bottom shows the command `npm test` being executed, and the output indicates that the test passed successfully.

```
1 // src/Login.test.js
2 import React from 'react';
3 import { render } from '@testing-library/react';
4 import Login from './Login';
5
6 test('renders login form', () => {
7   const { getByText, getByPlaceholderText } = render(<Login />);
8   const loginHeader = getByText('Login');
9   const emailInput = getByPlaceholderText('Email');
10  const passwordInput = getByPlaceholderText('Password');
11  const loginButton = getByText('Log In');
12
13  expect(loginHeader).toBeInTheDocument();
14  expect(emailInput).toBeInTheDocument();
15  expect(passwordInput).toBeInTheDocument();
16  expect(loginButton).toBeInTheDocument();
17 });
```

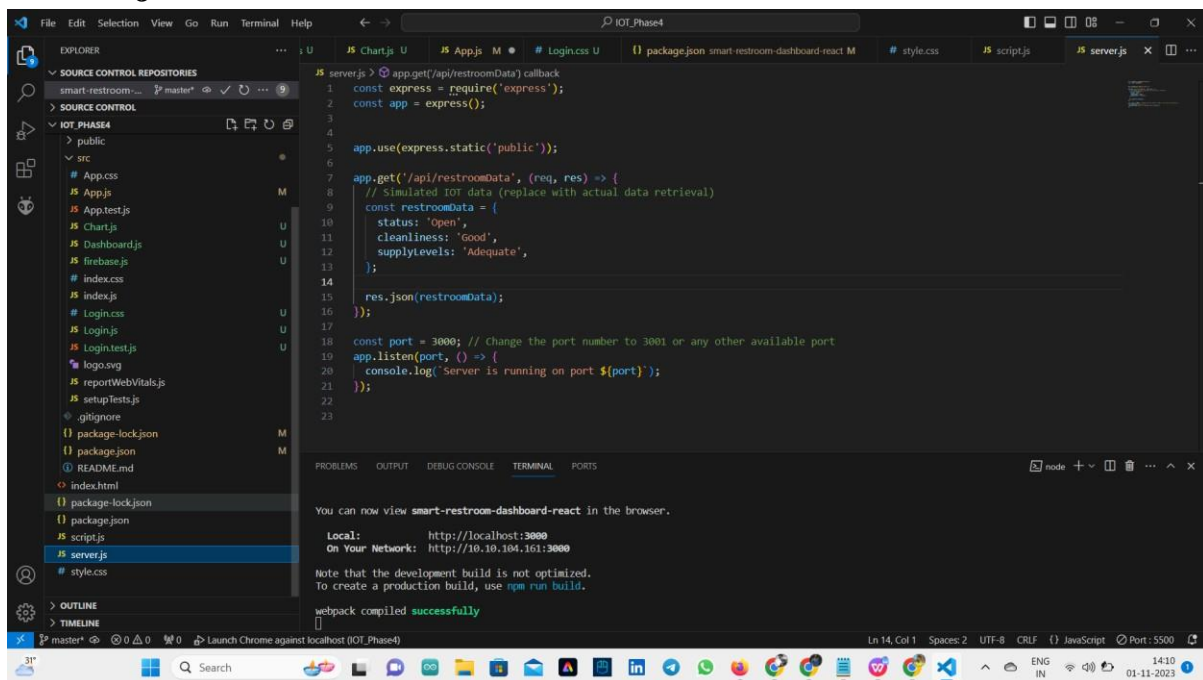
Terminal Output:

```
You can now view smart-restroom-dashboard-react in the browser.
Local: http://localhost:3000
On Your Network: http://10.10.104.161:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Server.js



The screenshot shows the VS Code editor with the `Server.js` file open. The file contains a Node.js server script that uses Express.js to serve static files and handle API requests. The terminal at the bottom shows the command `npm start` being executed, and the output indicates that the server is running successfully on port 3000.

```
1 const express = require('express');
2 const app = express();
3
4 app.use(express.static('public'));
5
6 app.get('/api/restroomData', (req, res) => {
7   // Simulated IOT data (replace with actual data retrieval)
8   const restroomData = {
9     status: 'Open',
10    cleanliness: 'Good',
11    supplyLevels: 'Adequate',
12  };
13
14  res.json(restroomData);
15 });
16
17 const port = 3000; // Change the port number to 3001 or any other available port
18 app.listen(port, () => {
19   console.log(`Server is running on port ${port}`);
20 });
```

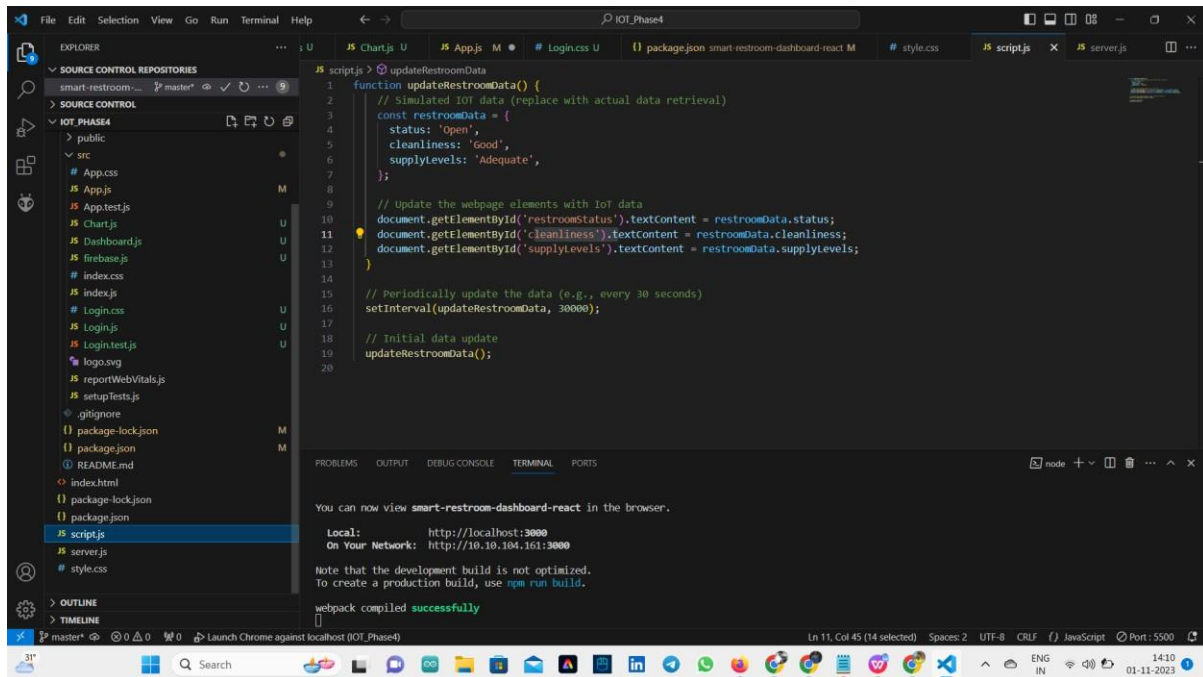
Terminal Output:

```
You can now view smart-restroom-dashboard-react in the browser.
Local: http://localhost:3000
On Your Network: http://10.10.104.161:3000

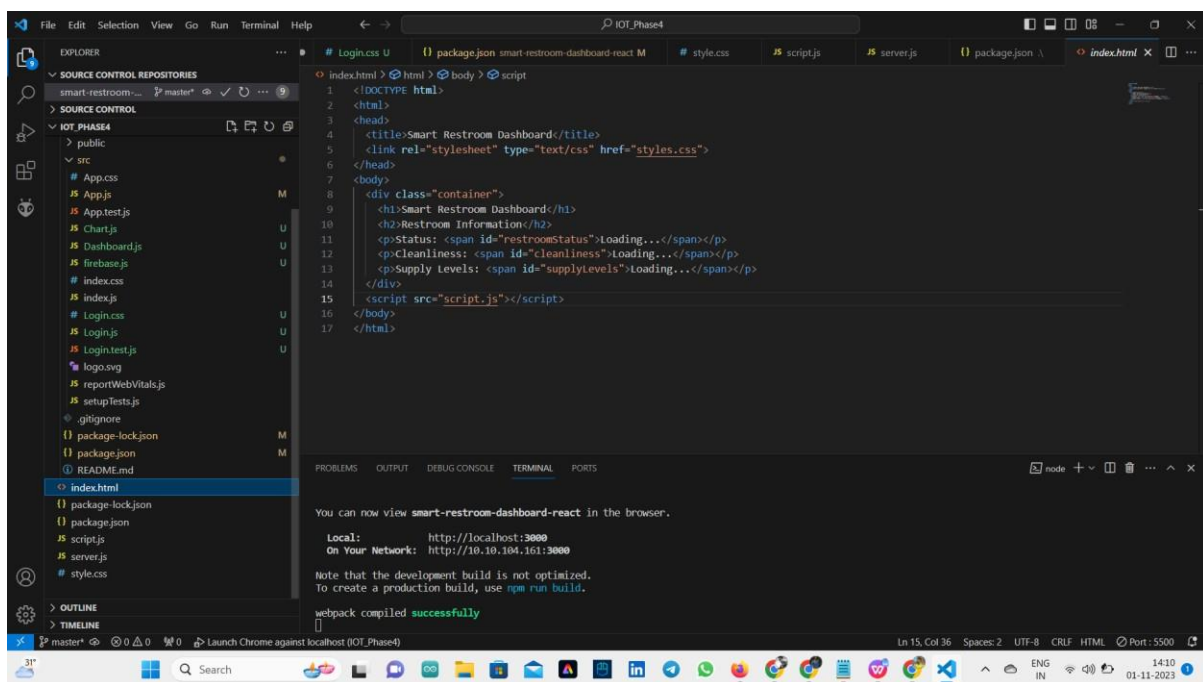
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

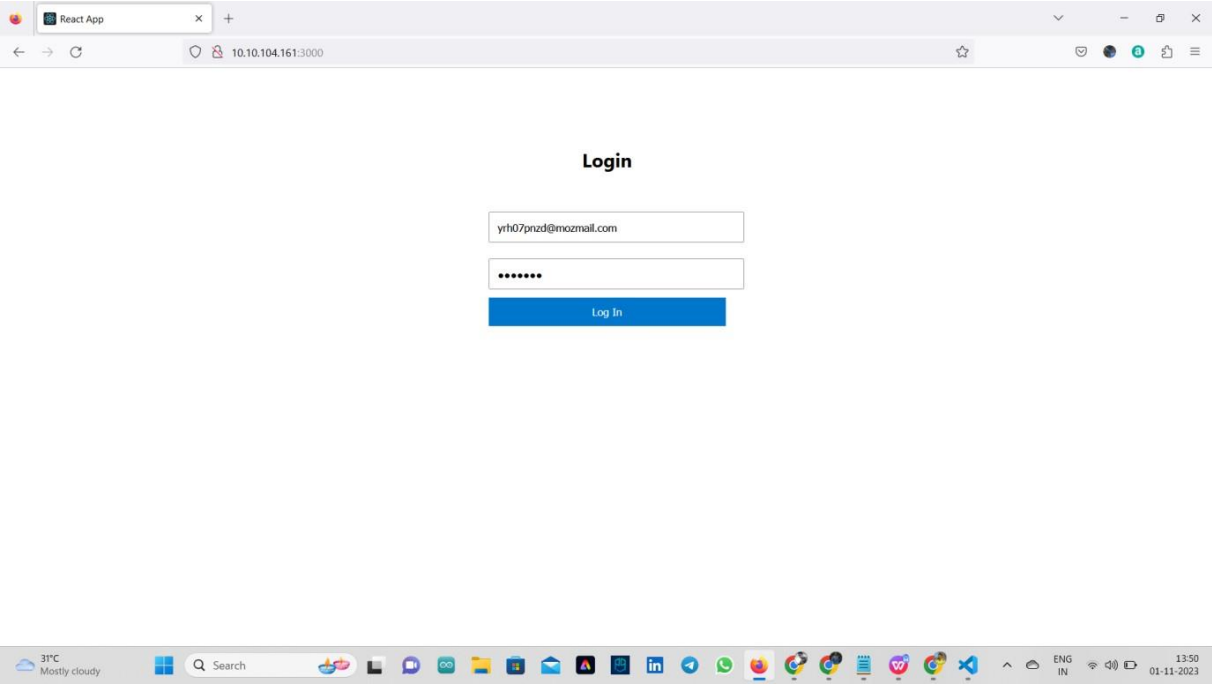
Script.js



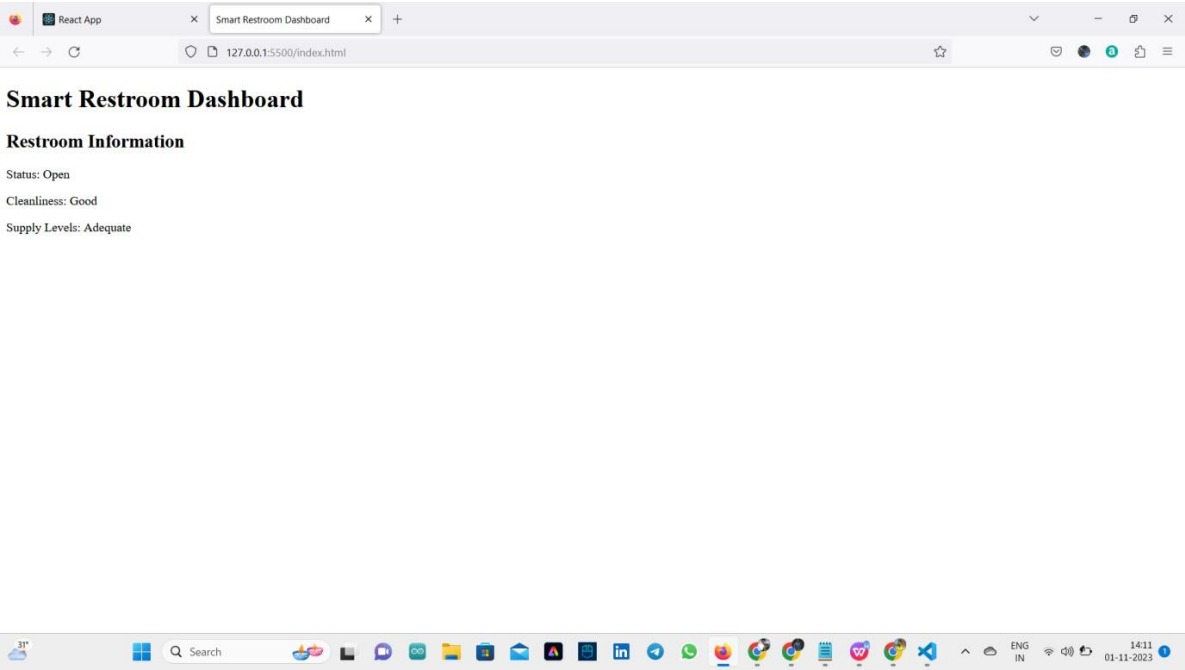
Index.html



Login page



Data Visualization



Note: Designing an app for a smart public restroom using IOT can improve user experience and efficiency, benefiting both restroom users and facility managers. Remember to stay flexible and adaptable, as IOT technology and user needs may evolve over time.