

**KGiSL INSTITUTE OF TECHNOLOGY**

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035**.)**

**DEPARTMENT OF**

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**NAAN MUDHALVAN - INTERNET OF THINGS**

**Smart Public Restroom**

**NAME:** GOWTHAM A

**REG NO:** 711721243304

**NM ID:** au21aia68

**TEAM MENTOR:** Mr**.** Mohankumar M

**TEAM EVALUATOR:** Ms. Akilandeeshwari M

**Phase 4: Development Part 2**

**DESIGN THE PLATFORM TO RECEIVE AND DISPLAY REAL-TIME RESTROOM AVAILABILITY AND CLEANLINESS DATA INVOLVES BOTH HARDWARE AND SOFTWARE COMPONENTS.**

**Problem Statement:**

The Smart Public Restrooms Enhancement project aims torevolutionize public restroom facilities by integratingadvanced technologies to improve user experience, sanitation, and operational efficiency. These smart public restrooms will provide a more convenient, hygienic, and sustainable solution for the community.

**PROCEDURE:**

**Hardware Components:**

**Sensors**: Install sensors in each restroom to collect data. These sensors should detect occupancy, monitor cleanliness, and collect environmental data such as temperature and humidity. Some possible sensor types include occupancy sensors, motion detectors, and air quality sensors.

**Connectivity:** Ensure the sensors are connected to the internet via Wi-Fi, Bluetooth, or other relevant connectivity protocols. You can use IoT (Internet of Things) devices for this purpose.

**Power Supply:** Sensors will require a power source. Depending on the location, you can use batteries, wired power, or energy-efficient solutions like solar panels**.**

**Software Components:**

**Data Collection and Processing:**

Set up a central server to collect data from all sensors.

Implement data processing algorithms to filter and analyze the incoming data.

**Create a database to store historical data and track restroom usage and cleanliness trends over time**.

**User-Facing Application:**

Develop a user-friendly mobile app and/or website where users can access real-time restroom information.

Allow users to search for nearby restrooms and view details such as availability, cleanliness ratings, and wait times.

**Real-time Data Presentation:**

Integrate the data from sensors and present it in real-time on the user interface.

Use intuitive icons or color-coding to convey cleanliness and availability information at a glance.

**User Feedback:**

Enable users to provide feedback on the restroom conditions, adding a social aspect to the platform. This can help in maintaining data accuracy.

Include a rating system for cleanliness and overall restroom experience.

**Notifications:**

Implement push notifications to alert users when a restroom becomes available or when cleanliness levels improve.

**Mapping and Navigation:**

Integrate mapping and navigation features to help users find the nearest available restrooms**.**

**Administrative Dashboard:**

Create a dashboard for administrators and restroom maintenance personnel to monitor and manage the system.Provide tools to configure sensors, set alerts for cleanliness thresholds, and analyze historical data**.**

**Security and Privacy:**

Implement robust security measures to protect the data, user accounts, and the system from unauthorized access.Address privacy concerns by anonymizing user data and explaining the data usage policy to users.

**Business Model:**

Determine your revenue model, whether it's through subscription fees, in-app advertising, or partnerships with venues and facilities.

**Marketing and Adoption:**

Market the platform to potential users, venues, and facilities.Encourage businesses to integrate their restrooms into your platform.

**Maintenance and Updates:**

Continuously monitor and maintain the hardware and software components.Release updates to improve user experience and address any issues that arise.

**Scaling:**

Plan for scalability as your user base grows, and new restrooms are added to the system.Creating a platform that displays real-time restroom availability and cleanliness data requires a combination of hardware, software, and user engagement. The success of your platform will depend on its reliability, accuracy, and user-friendliness.

**CODE:**

restroom-app/

|-- public/

| |-- index.html

| |-- styles.css

| |-- script.js

|-- server.js

|-- package.json

const express = require('express');

const app = express();

const port = 3000;

app.use(express.json());

app.use(express.static('public'));

const sqlite3 = require('sqlite3').verbose();

const db = new sqlite3.Database('restroom.db');

app.get('/api/restroom', (req, res) => {

db.get('SELECT \* FROM restroom\_data ORDER BY timestamp DESC LIMIT 1', (err, row) => {

if (err) {

console.error(err.message);

res.status(500).json({ error: 'Could not fetch data' });

return;

}

res.json(row);

});

});

app.post('/api/restroom', (req, res) => {

const { availability, cleanliness } = req.body;

const timestamp = new Date().toISOString();

const stmt = db.prepare('INSERT INTO restroom\_data (availability, cleanliness, timestamp) VALUES (?, ?, ?)');

stmt.run(availability, cleanliness, timestamp, function(err) {

if (err) {

console.error(err.message);

res.status(500).json({ error: 'Could not insert data' });

return;

}

res.status(201).json({ message: 'Data added successfully' });

});

stmt.finalize();

});

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

db.serialize(() => {

db.run("CREATE TABLE IF NOT EXISTS restroom\_data (availability TEXT, cleanliness TEXT, timestamp TEXT)");

});

<!DOCTYPE html>

<html>

<head>

<title>Restroom App</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<h1>Restroom Availability and Cleanliness</h1>

<div id="restroom-info">

<p>Availability: <span id="availability">Loading...</span></p>

<p>Cleanliness: <span id="cleanliness">Loading...</span></p>

</div>

<form id="update-form">

<label for="availability-input">Availability:</label>

<input type="text" id="availability-input" required>

<label for="cleanliness-input">Cleanliness:</label>

<input type="text" id="cleanliness-input" required>

<button type="submit">Update</button>

</form>

<script src="script.js"></script>

</body>

</html>

body {

text-align: center;

}

#restroom-info {

font-size: 18px;

margin-top: 20px;

}

form {

margin-top: 20px;

}

form label, form input {

display: block;

margin-bottom: 10px;

}

form button {

padding: 10px 20px;

}

document.addEventListener('DOMContentLoaded', function () {

const availabilityElement = document.getElementById('availability');

const cleanlinessElement = document.getElementById('cleanliness');

const updateForm = document.getElementById('update-form');

const availabilityInput = document.getElementById('availability-input');

const cleanlinessInput = document.getElementById('cleanliness-input');

updateForm.addEventListener('submit', (e) => {

e.preventDefault();

const newAvailability = availabilityInput.value;

const newCleanliness = cleanlinessInput.value;

fetch('/api/restroom', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ availability: newAvailability, cleanliness: newCleanliness }),

})

.then((response) => {

if (response.status === 201) {

availabilityInput.value = '';

cleanlinessInput.value = '';

updateRestroomData();

} else {

alert('Failed to update data.');

}

});

});

function updateRestroomData() {

fetch('/api/restroom')

.then((response) => response.json())

.then((data) => {

availabilityElement.textContent = data.availability;

cleanlinessElement.textContent = data.cleanliness;

});

}

updateRestroomData();

});

const express = require('express');

const app = express();

const port = 3000;

app.use(express.json());

app.use(express.static('public'));

const sqlite3 = require('sqlite3').verbose();

const db = new sqlite3.Database('restroom.db');

// Simulated user authentication

const users = [

{ username: 'demo', password: 'password' },

// Add more users here

];

app.post('/api/login', (req, res) => {

const { username, password } = req.body;

const user = users.find((u) => u.username === username && u.password === password);

if (user) {

res.status(200).json({ message: 'Authentication successful' });

} else {

res.status(401).json({ error: 'Authentication failed' });

}

});

app.get('/api/restroom', (req, res) => {

db.get('SELECT \* FROM restroom\_data ORDER BY timestamp DESC LIMIT 1', (err, row) => {

if (err) {

console.error(err.message);

res.status(500).json({ error: 'Could not fetch data' });

return;

}

res.json(row);

});

});

app.post('/api/restroom', (req, res) => {

const { availability, cleanliness } = req.body;

const timestamp = new Date().toISOString();

const stmt = db.prepare('INSERT INTO restroom\_data (availability, cleanliness, timestamp) VALUES (?, ?, ?)');

stmt.run(availability, cleanliness, timestamp, function (err) {

if (err) {

console.error(err.message);

res.status(500).json({ error: 'Could not insert data' });

return;

}

res.status(201).json({ message: 'Data added successfully' });

});

stmt.finalize();

});

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

db.serialize(() => {

db.run('CREATE TABLE IF NOT EXISTS restroom\_data (availability TEXT, cleanliness TEXT, timestamp TEXT)');

});

<!DOCTYPE html>

<html>

<head>

<title>Restroom App</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<h1>Restroom Availability and Cleanliness</h1>

<div id="restroom-info">

<p>Availability: <span id="availability">Loading...</span></p>

<p>Cleanliness: <span id="cleanliness">Loading...</span></p>

</div>

<form id="update-form">

<label for="availability-input">Availability:</label>

<input type="text" id="availability-input" required>

<label for="cleanliness-input">Cleanliness:</label>

<input type="text" id="cleanliness-input" required>

<button type="submit">Update</button>

</form>

<div id="login-form">

<label for="username">Username:</label>

<input type="text" id="username" required>

<label for="password">Password:</label>

<input type="password" id="password" required>

<button id="login">Login</button>

</div>

<button id="logout" style="display: none;">Logout</button>

<script src="script.js"></script>

</body>

</html>

body {

text-align: center;

}

#restroom-info {

font-size: 18px;

margin-top: 20px;

}

form {

margin-top: 20px;

}

form label,

form input {

display: block;

margin-bottom: 10px;

}

form button {

padding: 10px 20px;

}

#login-form,

#logout {

display: none;

}

document.addEventListener('DOMContentLoaded', function () {

const availabilityElement = document.getElementById('availability');

const cleanlinessElement = document.getElementById('cleanliness');

const updateForm = document.getElementById('update-form');

const availabilityInput = document.getElementById('availability-input');

const cleanlinessInput = document.getElementById('cleanliness-input');

const loginForm = document.getElementById('login-form');

const usernameInput = document.getElementById('username');

const passwordInput = document.getElementById('password');

const loginButton = document.getElementById('login');

const logoutButton = document.getElementById('logout');

// User authentication status

let isAuthenticated = false;

loginButton.addEventListener('click', () => {

const username = usernameInput.value;

const password = passwordInput.value;

fetch('/api/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ username, password }),

})

.then((response) => {

if (response.status === 200) {

isAuthenticated = true;

alert('Authentication successful!');

loginForm.style.display = 'none';

logoutButton.style.display = 'block';

updateForm.style.display = 'block';

usernameInput.value = '';

passwordInput.value = '';

} else {

alert('Authentication failed. Try again.');

}

});

});

logoutButton.addEventListener('click', () => {

isAuthenticated = false;

logoutButton.style.display = 'none';

loginForm.style.display = 'block';

updateForm.style.display = 'none';

});

updateForm.addEventListener('submit', (e) => {

e.preventDefault();

const newAvailability = availabilityInput.value;

const newCleanliness = cleanlinessInput.value;

fetch('/api/restroom', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ availability: newAvailability, cleanliness: newCleanliness }),

})

.then((response) => {

if (response.status === 201) {

availabilityInput.value = '';

cleanlinessInput.value = '';

updateRestroomData();

} else {

alert('Failed to update data.');

}

});

});

function updateRestroomData() {

fetch('/api/restroom')

.then((response) => {

if (response.status === 401 && !isAuthenticated) {

alert('Please log in first.');

return;

}

return response.json();

})

.then((data) => {

availabilityElement.textContent = data.availability || 'Not available';

cleanlinessElement.textContent = data.cleanliness || 'Unknown';

});

}

updateRestroomData();

});

**CONCLUSION:**

The provided code example illustrates the fundamental components of a simplified real-time restroom availability and cleanliness application. While it covers key features like data retrieval, user authentication, and basic error handling, real-world implementations require robust user security, integration with physical sensors, and adherence to data privacy regulations. To create a production-ready platform, it's essential to enhance security, scalability, and user experience, alongside developing cross-platform mobile apps for wider accessibility. Continuous user feedback and adherence to privacy and compliance standards are also crucial. The provided example serves as a foundational framework for a more complex and comprehensive restroom information platform..