Smart Guard: An IoT-Based Environmental Monitoring and Alert System for Educational Spaces

Aryan Sahu*, Bhavik Patel†, Hitesh Kumar‡, Jinil Patel§
*22110038, †22110047, ‡22110098, §22110184
Indian Institute of Technology Gandhinagar
Email: {aryan.sahu,bhavik.patel,hitesh.kumar,jinilkumar.patel}@iitgn.ac.in

***Abstract*—In this project, we propose and develop "Smart Guard," a scalable and modular IoT-based solution for environmental monitoring across classrooms and laboratories. The system collects data on temperature, humidity, light intensity, air quality, etc. via sensors connected to ESP-32 microcontrollers. The data is transmitted to a centralized server using lightweight protocols and visualized on a dashboard for real-time monitoring and alerts. This report details the architecture, implementation, results, and future scope of the system.**

***Index Terms*—IoT, Environmental Monitoring, ESP-32, React, Vite.js, PostgreSQL, HTTP**

I. Introduction

The quality of an indoor environment plays a vital role in maintaining concentration, productivity, and overall health. Recognizing this, the Smart Guard System is conceived as a low-cost, scalable, and efficient solution to monitor classroom and laboratory conditions in real time. This project integrates hardware components such as temperature, humidity, light, and air quality sensors with a microcontroller (ESP32) and a robust backend server. The sensor data is transmitted over Wi-Fi using HTTP and stored in a PostgreSQL database, from where it is retrieved by a React-based dashboard for live visualization.

II. Objectives

The main goal of the Smart Guard System is to create a simple and reliable solution to monitor classroom and lab environments in real time. The system should be easy to expand and maintain over time. The specific objectives of this project are:

- To collect environmental data such as temperature, humidity, and light intensity using various sensors.
- To send this data from the microcontroller to a backend server using HTTP requests.
- To store the sensor data in a structured and scalable database.
- To build a frontend dashboard that shows real-time and past data in a user-friendly way.
- To implement an alert notification system that informs users when any environmental parameter crosses a defined safe threshold.
- To keep the system modular, so that new sensors or classrooms can be added easily in the future.

Technology Stack

The Smart Guard System is built using a full-stack approach, combining both frontend and backend technologies along with a robust database and supporting libraries.

- **Frontend**
  - React.js with Vite for fast and modern web development.
  - Chart.js for rendering interactive graphs and past trends.
  - Context API for global state management across the dashboard.
- **Backend**
  - Node.js for building RESTful APIs.
  - Nodemailer for sending alert notifications via email.
- **Database**
  - PostgreSQL used as the primary database for storing time-series sensor data.

III. System Architecture

The system is organized into four primary layers, each responsible for a distinct part of the data flow and user interaction pipeline:

A. Hardware

The hardware layer consists of various sensors interfaced with an ESP32 microcontroller. The ESP32 collects data from the sensors and transmits it to the back-end API for storage and processing. In the current setup, the following sensors are utilized:

- **DHT22** – Measures temperature and humidity.
- **BH1750** – Measures ambient light intensity.

The architecture is designed to be scalable, allowing the integration of additional sensors depending on the I/O capabilities of the ESP32.

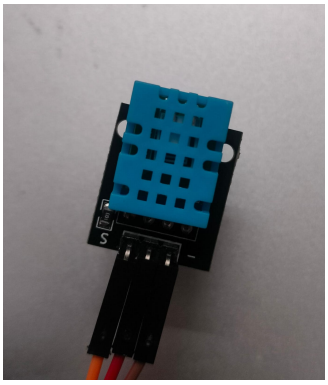

Fig. 1. DHT22 Sensor

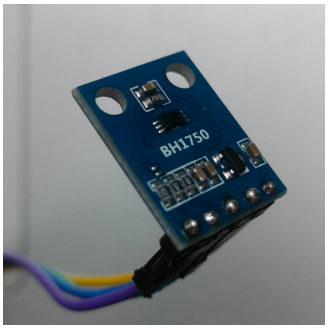

Fig. 2. BH1750 Sensor

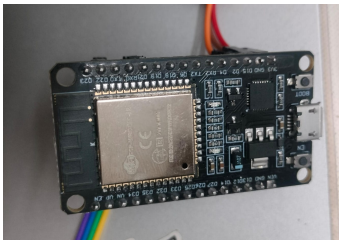

Fig. 3. ESP32 Microcontroller

B. Front-End

A React-based web interface delivers real-time visualization of sensor data through interactive graphs and dynamic tables, allowing users to monitor environmental conditions intuitively and efficiently. The application leverages React's Context API for seamless state management across components and utilizes Chart.js for rendering visually engaging and responsive data visualizations.

C. Back-End

The back-end is powered by a Node.js server that serves as a bridge between the hardware and the PostgreSQL database. It handles incoming sensor data, performs necessary preprocessing, and reliably stores the information for future use. The server also exposes a set of RESTful APIs that enable the front-end to access both real-time and past data for analytics and visualization. Additionally, Nodemailer is integrated to send automated email alerts to administrators based on predefined conditions or anomalies.

D. Database

A PostgreSQL database is used to persist sensor data. It is optimized for time-series data storage and retrieval, supporting efficient querying for both recent and past sensor readings.

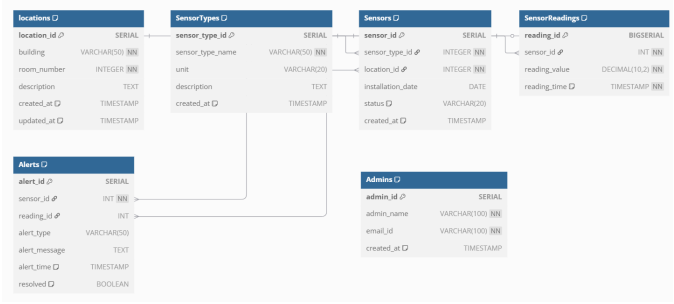

Fig. 4. Schema Diagram of Database

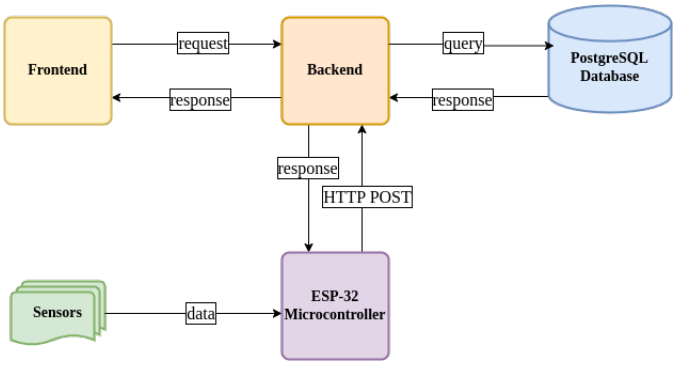

Fig. 5. System Architecture

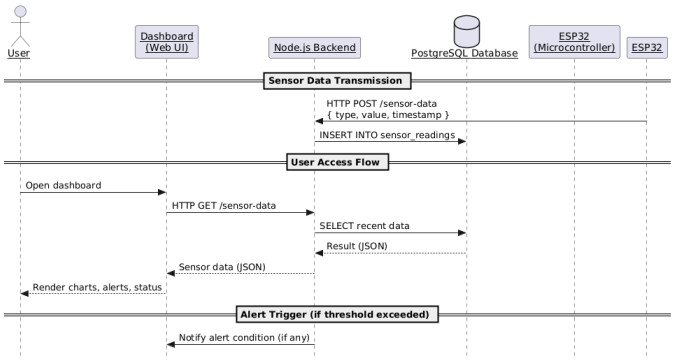

Fig. 6. Complete Data Flow between different layers

IV. Project File Structure

The project is organized into separate directories for the frontend, backend, and simulation scripts. This modular structure makes the codebase easier to maintain and extend.

Folder Structure:

```
Smart-Guard-IOT-Application/
|- backend/
|   \- src/
|       |- controllers/
|       |   |- alertController.ts
```

```
|     |  |- sensorController.ts
|     |  \- ...
|     |- routes/
|     |  |- alertRoutes.ts
|     |  |- sensorRoutes.ts
|     |  \- ...
|     |- database/
|     |- index.ts
|     \- db.ts
|- db/
|- frontend/
|  |- src/
|  |  |- components/
|  |  |  |- AlertsPage.tsx
|  |  |  |- Dashboard.tsx
|  |  |  |- Navbar.tsx
|  |  |  \- ...
|  |  |- context/
|  |  |  \- SelectionContext.tsx
|  |  |- utils/
|  |  |  \- fetch.tsx
|  |  |- App.tsx
|  |  \- main.tsx
|  \- public/
|- sensor_simulator.py
|- Makefile
\- README.md
```

Important Files and Their Roles:

- `backend/src/index.ts`
  The main entry point for the backend server. It initializes the Express app, sets up middleware, and registers API routes.
- `backend/src/db.ts`
  This files connects to the database using the Environment Variables.
- `backend/src/controllers/`
  Contains the backend logic for processing incoming requests. This includes files like `alertController.ts` and `sensorController.ts` which manage alerts and sensor data respectively.
- `backend/src/routes/`
  Defines RESTful API endpoints that route client requests to the appropriate controller functions. For example, `alertRoutes.ts` and `sensorRoutes.ts` map URL paths to backend operations.
- `db/`
  Not a necessary folder but it contains the schema and queries for the database.
- `frontend/src/components/`
  Includes frontend components like `Dashboard.tsx`, `Navbar.tsx`, and `AlertsPage.tsx`. These handle the visual layout and live data rendering on the web dashboard.
- `frontend/src/context/`
  Holds React context providers `SelectionContext.tsx` used for managing global application state (e.g., selected acadblock or room number).
- `frontend/src/utils/fetch.tsx`
  Contains helper functions to handle API calls from the frontend to the backend.
- `frontend/src/App.tsx` and `main.tsx`
  `App.tsx` is the root component of the React application, while `main.tsx` is the entry point that renders the app into the DOM.
- `sensor_simulator.py`
  A Python script that simulates sensor behavior by periodically sending mock data to the backend via HTTP. This was useful during development and testing before real hardware integration.
- `Makefile`
  Provides convenient command shortcuts for starting the backend, running database setup, and building the frontend — helpful for streamlining development tasks.

V. Objectives Achieved

- Modular architecture built with scalability in mind.
- Sensor integration and real-time data transmission completed
- Functional backend and frontend communication established
- Threshold-based alert system operational
- Visualizations for past trends implemented

The system provides a robust and user-friendly interface for administrators and users to interact with real-time and past data. Key features include:

- **Sensor Management:** Administrators can seamlessly register new sensors into the system, specifying the sensor type (e.g., temperature, humidity) and installation location (building and room). This modular setup allows flexible expansion as needed.

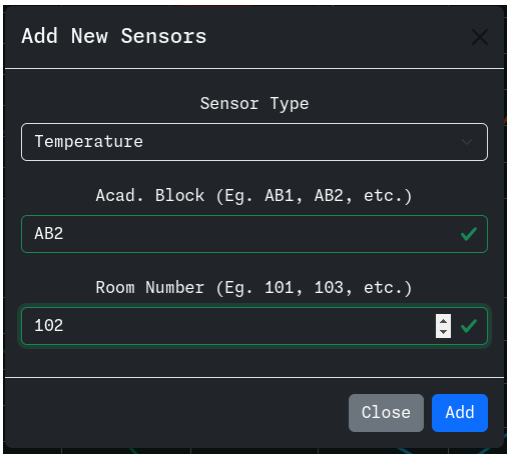

Fig. 7. Add sensor to existing locations.

- **Room-based Data Access:** Users can select specific buildings and rooms (e.g., Academic Block, Room 105) to view current and past readings from all sensors deployed in that room. The interface filters sensor data contextually.

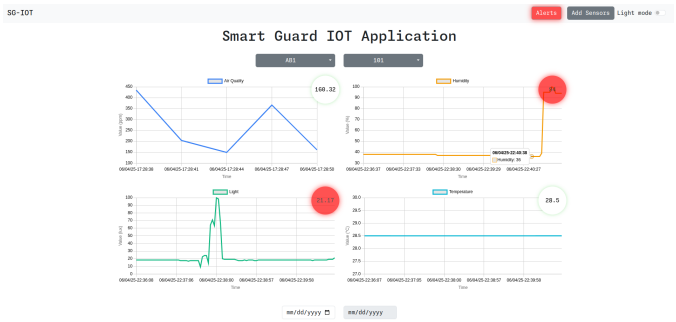

Fig. 8. Dashboard

- **Live Data Visualization:** Real-time sensor readings are displayed through interactive dashboards, including graphs, allowing users to monitor environmental conditions on the fly.
- **past Trend Analysis:** Users can select a date or custom date range to view trends in sensor readings such as temperature or air quality over time.

Fig. 9. Filter data for specific range of dates

- **Alert Log Viewer:** The system maintains a detailed log of alerts triggered by threshold violations (e.g., high temperature or poor air quality).

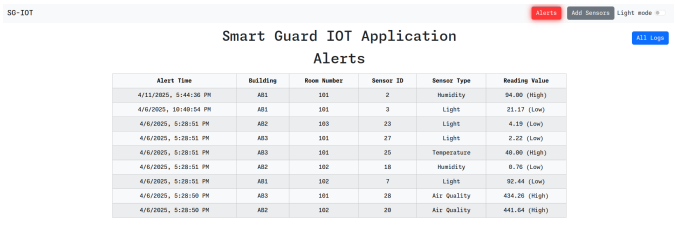

Fig. 10. Current Unresolved Alerts

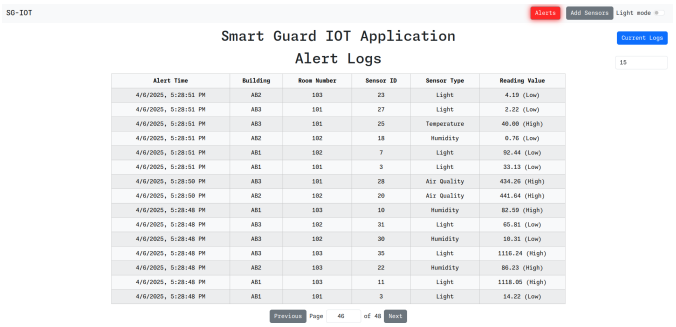

Fig. 11. All alert logs(resolved and unresolved)

- **Email Alert System:** Whenever a sensor reading exceeds a predefined threshold (e.g., temperature rises above 35°C or humidity drops below 30%), the system automatically triggers an email notification to the registered administrator(s). This enables proactive action without the need for manual monitoring.

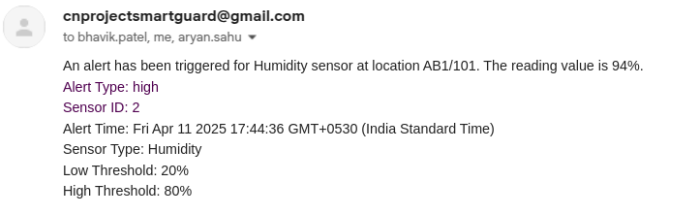

Fig. 12. Email Alert to the User

VI. Learnings

- Experience with IOT devices and microcontroller programming.
- Familiarity with full-stack development and REST API architecture.
- Proficiency in asynchronous communication and state management.
- Gained insights into real-time data handling and alert systems.

VII. Future Prospects

- **Integration with MQTT Protocol:** Implement MQTT (Message Queuing Telemetry Transport) for efficient publish-subscribe communication between IoT devices and the server. This will reduce overhead, improve real-time responsiveness, and scale better for numerous sensors.
- **User Role Management:** Introduce granular access control for different types of users—such as faculty, facility managers, and system admins—with role-based dashboards.
- **Mobile App and Push Notifications:** Build a companion mobile app to give users access to real-time readings and alerts on the go, with push notifications for critical events.

- **Predictive Analytics:** Use past sensor data with machine learning models to predict environmental anomalies or system failures before they happen.

Individual Contributions

Each team member contributed to different parts of the project to ensure even distribution of work. The individual contributions are as follows:

- **Bhavik Patel**
  - Developed the backend using Node.js and Express.js.
  - Created REST API endpoints for inserting and retrieving sensor data.
  - Connected the backend to the PostgreSQL database.
- **Aryan Sahu**
  - Set up the ESP32 microcontroller and interfaced it with environmental sensors.
  - Implemented HTTP-based data transmission from sensors to the backend.
  - Conducted initial hardware testing to verify data accuracy.
- **Hitesh Kumar**
  - Built the frontend dashboard using React.js.
  - Designed UI components to display real-time and past data.
  - Integrated the frontend with the backend APIs using Axios.
- **Jinil Patel**
  - Designed initial database schema.
  - Integrated the hardware, backend, and frontend components
  - Implemented the alert email system for threshold breaches.
  - Carried out system testing and ensured smooth end-to-end data flow.

VIII. Conclusion

Smart Guard successfully demonstrates the feasibility and value of IoT-based environmental monitoring in academic environments. With modularity, real-time feedback, and actionable insights, it provides a foundation for intelligent infrastructure management.

Acknowledgments

We thank the CS331: Computer Networks instructor Sameer G. Kulkarni and the TA Ayushman Singh for their support and technical guidance throughout the development of this project. Source code: Github Link.

References

[1] Node.js Foundation. https://nodejs.org
[2] PostgreSQL Documentation. https://www.postgresql.org
[3] React Docs. https://react.dev
[4] ESP32 Tutorials, "ESP32 Tutorials," *ESP32IO.com*. [Online]. Available: https://esp32io.com/tutorials.