# Final Project Report: IoT Temperature Sensor Control System

## 1. Introduction

To demonstrate advanced knowledge in IoT, I have developed an IoT-based Smart Room Heater Control System. This project seeks to improve smart home automation through the automatic regulation of room temperature, thereby enhancing comfort and promoting energy efficiency. The project utilises the Internet of Things (IoT) to integrate temperature monitoring with automated heater control. The system employs a simulated temperature sensor that transmits data to a MQTT broker, which is then processed and stored in a MongoDB Atlas database. The data is accessible in real-time through a web interface created with Node.js and Express.js, facilitating efficient interaction with the system.

## 2. System Architecture and Components

The Smart Room Heater Control System is composed of several key components that work cohesively to achieve real-time temperature monitoring and control. These components include:

### 2.1 Temperature Sensor Simulation

- A client script (client.js) was developed to generate random temperature values, simulating the behavior of a real temperature sensor. This simulation is crucial for testing the system's response under varying temperature conditions.
- Temperature values are published to an MQTT broker every 5 seconds, providing consistent updates that enable the heater control system to react promptly.

### 2.2 MQTT Integration

- MQTT, a lightweight messaging protocol, was selected to enable efficient data transmission between the simulated temperature sensor and the backend server.
- The data is disseminated to a specific MQTT topic (/smartroom/heater100), serving as a communication channel for further processing by other system components.The MQTT integration provides low latency and reliable communication, rendering it suitable for real-time IoT applications.

### 2.3 Node-RED Flow

- Node-RED was utilised to subscribe to the MQTT topic and process incoming temperature data.

- The implemented logic assesses whether to activate or deactivate the heater according to established temperature thresholds. If the temperature falls below a specified threshold, such as 15°C, the heater is engaged. If the temperature surpasses a predetermined threshold (e.g., 25°C), the heater is turned off.The processed data is subsequently stored in a MongoDB Atlas database for future reference and analysis.

### 2.4 Database Management

- A MongoDB Atlas cluster was established to store temperature readings and heater status data, providing scalability and reliability.
- A well-defined schema was established to ensure consistent data storage, facilitating the retrieval, visualisation, and analysis of sensor information, thereby enabling scalability for larger environments.

### 2.5 Web Interface Development

- A web interface was developed using HTML, CSS, and JavaScript, with backend support from Node.js and Express.js.
- The interface enables users to track temperature trends, check heater status, and operate heater manually via a toggle switch. This flexible feature lets users override automated controls.Users see room conditions and system operations in real time with 5-second updates.

## 3. System Features

### 3.1 Real-Time Data Display

- Real-time temperature data is displayed on the web interface using Chart.js for dynamic visualization, making the information easy to interpret.
- The temperature data is updated every 5 seconds, providing an accurate and live view of the room temperature and heater status. Users can observe trends and understand the system's performance at a glance.

### 3.2 Heater Control Mechanism

- The system offers automatic and manual control options, enabling users to adjust heater settings via the web interface.
- The dashboard displays current heater status, guiding users in manual overrides.

### 3.3 Light/Dark Mode

- Implemented light/dark mode toggle for enhanced user experience, offering visual theme alternatives based on taste.
- This function improves usability by making the interface accessible and comfortable in various lighting circumstances.

## 4. System Scalability

The system was designed with scalability in mind to accommodate increased sensor inputs and potential deployment in larger environments:

### 4.1 Scalable Data Handling

- MQTT enables the system to manage data from scattered sensors, allowing for scalability to meet user demands.
- MongoDB Atlas was utilised to store massive amounts of data without performance concerns, ensuring responsiveness as additional devices are added.

### 4.2 Cloud Deployment

- MongoDB Atlas, offers seamless scalability for growing data volumes. Automatic scaling and maintenance reduce manual involvement in the cloud.
- The workload was distributed across numerous EC2 instances using an AWS Classic Load Balancer. This arrangement maintains system performance amid peak demand, ensuring consistent user experiences.

## 5. Detailed Implementation

### 5.1 Backend Implementation

- The backend server using Node.js and Express.js, providing RESTful APIs that enable seamless interaction between the database and the frontend.
- The server.js script manages incoming requests, interacts with MongoDB Atlas, and returns data to the web interface. This ensures that users receive up-to-date information in real-time.

### 5.2 Frontend Implementation

- The interface features a real-time temperature chart, a table displaying recent temperature records, and a toggle switch for manual heater control.
- The dashboard dynamically updates every 5 seconds using the Fetch API, ensuring users have access to the latest data and control options, enhancing overall system interactivity.

### 5.3 Node-RED and MQTT Integration

- Node-RED was configured to subscribe to the MQTT topic /smartroom/heater100, enabling it to receive temperature data from the simulated sensor.
- The system was designed to take action based on temperature thresholds.

### 5.4 Load Balancing and EC2 Instances

- To ensure reliability and fault tolerance, two EC2 instances were deployed with an AWS Classic Load Balancer distributing incoming requests evenly.
- Enabled cross-zone load balancing for even distribution across availability zones, and connection draining for graceful instance deregistration to minimise user impact.

### 5.5 Data Flow and Storage

- Temperature data is generated by the simulated sensor and sent to the MQTT broker, which then passes it to Node-RED for processing.
- Node-RED processes the data, applies the logic for heater control, and stores the resulting information in MongoDB Atlas.
- The web interface periodically requests stored data, providing real-time insights to users about the temperature and heater status.

## 6. System Functionality

### 6.1 Temperature Monitoring and Heater Control

- The system automates room temperature monitoring, adjusting the heater status automatically to maintain user comfort and energy efficiency.
- Users have the flexibility to manually override the heater control through the web interface, allowing for personalized control of the heating system.

### 6.2 Real-Time Data Flow

- The MQTT broker facilitates the real-time flow of temperature data between sensors, the backend server, and the web interface.
- Node-RED plays a key role in processing and filtering sensor data before it is sent to the backend, ensuring the accuracy and reliability of the information presented to users.

## 7. Challenges and Solutions

### 7.1 Database Connectivity Issues

- Initially, challenges were encountered in connecting to MongoDB Atlas, specifically with IP whitelisting and connection string configurations. These issues were resolved by updating MongoDB security settings to allow connections from authorized IP addresses.
- Additionally, testing under different network conditions helped identify and mitigate connectivity issues, ensuring the system remained robust.

### 7.2 Real-Time Synchronization

- Ensuring real-time synchronization of data between the MQTT broker, database, and web interface was challenging. To address this, a periodic data fetching mechanism was implemented, ensuring that users received up-to-date information without overwhelming the server with requests.
- Future improvements could include implementing WebSocket connections to facilitate more instantaneous data updates, further reducing latency.

## 8. Future Improvements That Can be Made

### 8.1 User Interface Enhancements

- Improve the user interface by adding more advanced charts for historical data visualization and integrating predictive analytics to provide deeper insights into temperature trends.
- Add alerts for abnormal temperature conditions, notifying users promptly to take necessary action.

### 8.2 System Optimization

- Implement caching mechanisms to optimize data retrieval speed and reduce latency when fetching data from MongoDB, improving the system's responsiveness.
- Enhance the heater control logic by integrating predictive algorithms that analyze historical temperature patterns and optimize energy consumption based on user habits.

### 8.3 Deployment and Monitoring

- Deploy the system globally on a cloud platform to extend accessibility and cater to a wider user base, ensuring the solution is available from different geographic locations.

- Set up continuous monitoring and logging to track system performance, detect anomalies, and ensure proactive error handling. Tools like AWS CloudWatch can be used to gain real-time insights into the system's health.

**9. Conclusion**

The Smart Room Heater Control System integrates numerous IoT technologies to produce a scalable and responsive smart home automation system. This project uses MQTT for real-time data transmission, MongoDB Atlas for reliable and scalable data storage, Node-RED for data flow management, and Node.js for backend processing to create a strong and coherent system. This idea demonstrates a profound knowledge of IoT concepts and the potential of networked technology to improve daily life. The system's automated temperature adjustment improves user comfort and energy efficiency, which is crucial to smart home solutions. User-friendly interface and manual override options emphasise end-user freedom and control over their surroundings.

Scalable design, incorporating cloud-hosted MongoDB and AWS load balancing, makes the solution ideal for bigger and more sophisticated smart settings. The lightweight communications protocol MQTT enables real-time performance, essential for maintaining ideal room conditions without delays. Node-RED also makes data flow development and management easy.

Future efforts will focus on caching, predictive analytics, and wider cloud deployment to improve system performance. Advanced capabilities including historical data analysis, abnormal condition alarms, and predictive control will enhance user experience and system customisation.

Finally, IoT technologies like the Smart Room Heater Control System demonstrate their strength and adaptability. It solves real-time temperature monitoring and control problems while scaling for expansion. This study shows how IoT can provide practical, efficient, and user-centric solutions, enabling smart home automation.