Journal of Applied Informatics and Computing (JAIC)
Vol.10, No.4, August 2026, pp. 3471~3484
e-ISSN: 2548-6861

Development of a Configurable IoT-Based Adaptive Thermal Comfort Monitoring Platform Using Scrum and DevOps Practices

Sri Atikah [1]*
* BPS Statistics Indonesia
sri.atikah@bps.go.id [1]

Article Info

Article history:

Received 2026-06-08
Revised 2026-07-25
Accepted 2026-08-08

Keyword:

Internet of Things,
Adaptive thermal comfort,
Scrum,
DevOps,
CI/CD,
Web-based monitoring,
Building management.

ABSTRACT

Effective indoor thermal comfort management in multi-room buildings requires the integration of environmental sensor data and occupant feedback while remaining adaptable to different building configurations. However, many existing occupant-centric thermal comfort systems are tightly coupled to specific building environments, which limits their adaptability to new buildings and different room configurations. To address this limitation, this study develops and evaluates an IoT-based adaptive thermal comfort monitoring platform designed for configurable multi-building deployment. The platform integrates environmental sensor data with occupant feedback collected through room-specific QR code-accessible web forms and supports two complementary recommendation mechanisms: sensor-driven threshold alerts and feedback-driven recommendations based on aggregated occupant responses. The platform was developed using the Scrum methodology over seven one-week sprint cycles, supported by a DevOps CI/CD pipeline that automated application building, testing, artifact management, and deployment. Evaluation included unit testing, integration testing, Selenium-based user interface testing, and Apache JMeter performance testing, with all automated test scenarios passing successfully. Performance testing yielded APDEX scores of 0.997 and 0.960 for the General User and Administrator workflows, respectively, with zero errors across all tested endpoints. Flexible building and room expansion was achieved through an entity-based architecture that models Building, Room, and Sensor as independent entities, enabling the incorporation of new spaces without structural modifications. These results demonstrate that the proposed platform provides a configurable and extensible software solution for adaptive thermal comfort monitoring in multi-room building environments.

I. INTRODUCTION

HVAC (Heating, Ventilation, and Air Conditioning) systems play a vital role in providing thermal comfort for building occupants [1]. Considering that humans spend most of their time indoors [2], thermal comfort levels are a key factor influencing both occupants' well-being and a building's energy consumption [3]. Solano et al. (2021) report that HVAC systems contribute to approximately 40–60% of energy consumption in commercial buildings [1]. Inefficient HVAC systems can lead to counterproductive occupant actions, such as opening windows when a room is too hot or overusing the air conditioner. These conditions will not only reduce the comfort of the occupants but also increase energy waste.

The rapid development of Internet of Things (IoT) technologies has enabled real-time monitoring of indoor environmental conditions through the deployment of interconnected sensing devices. Environmental sensors integrated into modern buildings generate continuous streams of data on key indoor conditions, such as temperature, humidity, air quality, and noise levels. This information serves as a foundation for adaptive decision-making

mechanisms that promote occupant well-being while improving the energy performance of the building [4], [5].

Although modern buildings can leverage sensor technologies for continuous environmental monitoring, not all rooms are equipped with adequate environmental sensors. Consequently, decisions regarding thermal comfort management are often guided by generalized assumptions that may not correspond to the specific comfort requirements and preferences of occupants in each space. To overcome these limitations, occupant feedback has emerged as a valuable complement to environmental sensor data. When integrated with IoT-based sensing technologies, occupant feedback provides a more comprehensive understanding of thermal comfort conditions and occupants' preferences. Consequently, HVAC systems can be adjusted more accurately to improve comfort levels while preserving energy efficiency.

Previous research has demonstrated that integrating occupant feedback with building control systems can improve both thermal comfort and energy efficiency. Pritoni et al. (2017) developed the TherMOOstat application, which integrates occupants' thermal comfort feedback with the HVAC control system and successfully achieved energy savings of 20–30% across the participating buildings [6]. Similarly, Soleimanijavid et al. (2024) demonstrated that the Occupant-Centric Control (OCC) approach is capable of maintaining occupant comfort while ensuring energy efficiency through the integration of environmental sensor data with occupant preferences [2]. Nevertheless, most of these approaches remain tailored to specific building environments, and deployment in new buildings necessitates significant changes to sensor configurations, data structures, and software components.

Besides scalability challenges in physical environments, the development of Internet of Things (IoT)-based systems also presents challenges from a software engineering perspective. The complexity of integrating sensor data, backend services, databases, user interfaces, and deployment processes demands a development methodology that supports iterative development, automated testing, and continuous deployment throughout the system lifecycle. However, few studies have incorporated both system architecture and software engineering practices into the development of IoT-based adaptive thermal comfort monitoring systems.

This limited incorporation of software engineering practices is evident in various studies on IoT-based environmental monitoring systems. Susilawati et al. (2023) developed a web-based temperature and humidity monitoring system using a microcontroller and DHT sensor, while Nugraha et al. (2023) designed an IoT-based monitoring system for hospital medication storage rooms integrating DHT sensors, the ThingSpeak and Blynk platforms, and an LCD display for threshold-based notifications [7], [8]. Although these studies demonstrate the viability of IoT technology for environmental monitoring, the focus of both studies remains on functional implementation and sensor accuracy, with limited discussion of software engineering aspects such as requirements management, iterative development, automated testing, and deployment processes.

From a software engineering perspective, several studies have applied Scrum and DevOps to the development of IoT systems and web-based applications. Kirsan et al. (2023) applied the Scrum framework to develop an IoT-based fire detection system, demonstrating that an iterative development approach can effectively manage the complexity of combined hardware-software projects [9]. Meanwhile, Ekasari et al. (2025) applied DevOps practices to develop a web-based e-commerce application by implementing Continuous Integration and Continuous Deployment (CI/CD) to automate the build, testing, and deployment processes [10]. However, these studies addressed Scrum or DevOps independently, and their combined application to the development of an IoT-based adaptive thermal comfort monitoring system has received limited attention.

A more relevant approach to this integration was proposed by Al Masud et al. (2022), who introduced a hybrid "DevOps-Enabled Agile" methodology to combine Agile and DevOps practices into a single software development process [11]. Their findings suggest that this approach can improve coordination across development, testing, and deployment activities. However, the proposed methodology was demonstrated using generic software projects, leaving limited evidence of its application to IoT-based systems, where sensor data integration, backend services, user interfaces, and deployment infrastructure introduce additional software engineering challenges.

The literature review indicates that previous studies have primarily focused either on IoT-based environmental monitoring systems or on software engineering practices such as Scrum and DevOps, with limited evidence of their combined application to adaptive thermal comfort monitoring platforms. Developing such systems requires the coordinated integration of sensor data processing, backend services, databases, user interfaces, and deployment infrastructure while remaining adaptable to evolving requirements. Furthermore, many existing solutions are tightly coupled to specific building environments, limiting their deployment across multiple buildings or changing spatial configurations.

To address these gaps, this study develops an IoT-based adaptive thermal comfort monitoring platform implemented as a configurable web platform that integrates environmental sensor data with occupant feedback. The platform was developed using Scrum as an iterative development framework and supported by DevOps practices for continuous integration, automated testing, and deployment throughout the software development lifecycle. Furthermore, the study proposes an entity-based architecture that enables flexible expansion to additional buildings and rooms through configuration rather than code modification, and validates the developed platform through multi-layered software testing to verify functional correctness, system integration, user interface functionality, and performance.

II. METHODOLOGY

This study adopted the Software Development Life Cycle (SDLC) to develop an IoT-based adaptive thermal comfort monitoring system. Scrum was employed to support iterative and incremental development in response to evolving project requirements [12], [13], while DevOps practices were integrated to automate code integration, testing, artifact management, and deployment throughout the development lifecycle. Together, Scrum and DevOps provided a structured framework for coordinating the development of multiple interdependent components, including sensor data processing, backend services, databases, user interfaces, and deployment infrastructure. The overall research methodology is illustrated in Figure 1.

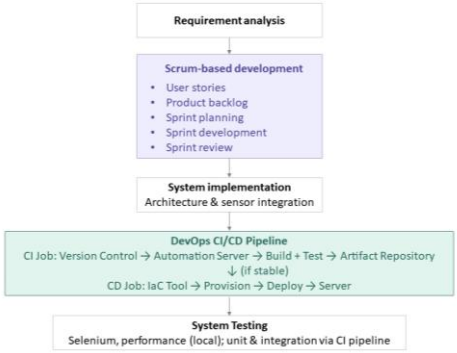

Figure 1. Research Methodology

A. Requirement Analysis

In this study, a requirements analysis was performed during the SDLC to define the requirements for an IoT-based adaptive thermal comfort monitoring system. The analysis results were then classified into three categories: functional, non-functional, and data requirements.

1) *Functional Requirements*: Functional requirements specify the system functionalities needed to support user needs and operational processes. The identified requirements are presented as follows:

- FR-01: The system shall be designed to support the integration of real-time environmental data from LORD MicroStrain sensors through available APIs or supported data integration mechanisms.
- FR-02: The system shall provide a room-specific web-based thermal comfort feedback form accessible via a QR code displayed in the corresponding room.
- FR-03: The system shall analyze environmental sensor data and occupant feedback to provide recommendations for room temperature adjustment.
- FR-04: The system shall provide a centralized interface that allows administrators to monitor and manage system information.
- FR-05: The system shall enable the visualization of sensor data and user feedback through charts and graphs.
- FR-06: The system shall notify specified users whenever room temperature or environmental conditions exceed configured thresholds.
- FR-07: The system shall provide mechanisms for user authentication and authorization.
- FR-08: The system shall facilitate the collection and persistent storage of environmental sensor data and occupant feedback.
- FR-09: The system shall support the dynamic expansion of new buildings and rooms based on the installed sensors.

2) *Non-Functional Requirements*: Non-functional requirements specify the quality attributes necessary to support effective system operation and long-term sustainability, including usability, performance, security, and maintainability. The identified non-functional requirements are summarized as follows:

- NFR-01: The system shall support responsive layouts to maintain usability and consistency across multiple device platforms and screen sizes.
- NFR-02: The system shall ensure that application pages are displayed within a reasonable time under normal operating conditions.
- NFR-03: The system shall implement security mechanisms to protect user data and access to the system.
- NFR-04: The system shall be designed to support future growth through a scalable and easily maintainable architecture.

3) *Data Requirements*: Data requirements define the information required to support environmental monitoring, recommendation generation, and thermal comfort analysis. Because a live API connection to the client's sensing infrastructure was unavailable during this study, historical sensor datasets provided by the client were used to support system design and validate data processing capabilities. The identified data requirements are summarized as follows:

- DR-01: Historical environmental sensor data provided by the client, containing time-stamped measurements such as temperature, CO_2, humidity, indoor air quality, and noise-related variables recorded at 15-minute intervals.
- DR-02: Occupant feedback data collected through room-specific web questionnaires accessed via QR codes displayed in the rooms.
- DR-03: Temperature thresholds and other environmental parameters used as the baseline for system alerts and recommendation generation.
- DR-04: Building, room, and sensor metadata used to support identification and multi-room monitoring within the system.

B. Scrum-Based Development Process

1) *User Requirement Collection:* User requirements were elicited through structured discussions with the project

client, who represented both building management and prospective system users. These discussions focused on identifying operational workflows, existing challenges in monitoring indoor environmental conditions, information needs for different user groups, and the functional requirements of the proposed adaptive thermal comfort monitoring platform.

Based on the elicited requirements, three user roles were identified: General User, Administrator, and Super Administrator, each associated with specific responsibilities and access rights, as summarized in Table I. These user roles served as the basis for formulating user stories and prioritizing Product Backlog Items during the Scrum-based development process.

TABLE I
ROLES AND USER DESCRIPTIONS

Role	Description
General User	Building occupants who use the system to monitor real-time room temperature and submit thermal comfort feedback to support adaptive thermal comfort management.
Administrator	Building managers responsible for managing building, room, and sensor data, monitoring indoor thermal conditions, and reviewing system-generated recommendations for thermal comfort management.
Super Admin	A privileged system administrator responsible for managing user accounts and the overall system configuration.

2) *User Stories:* The identified requirements were translated into user stories to describe the system functionality from the perspective of each user role. The resulting user stories served as the primary input for constructing the Product Backlog and are summarized as follows:

- All Users
  US-01: As a user, I want to log in to the system through a role-based access portal so that I can securely access features relevant to the role defined for me.
- General User
  US-02: As a general user, I want to submit feedback through a simple and user-friendly form so that I can report my thermal comfort experience efficiently.
  US-03: As a general user, I want to access real-time room temperature data so that I can monitor the thermal conditions of my current environment.
- Administrator
  US-04: As an administrator, I want to import and export data in CSV format so that I can support external data analysis and interoperability with other tools.
  US-05: As an administrator, I want the system to automatically provide recommendations for controlling the building's heating or cooling system based on sensor data and occupant feedback, so that thermal comfort can be optimally maintained.
  US-06: As an administrator, I want to be able to add, edit, and delete building and room data so that the system can be easily adjusted when sensors are installed or removed in a new building.
  US-07: As an administrator, I want to view spatial representations of sensor locations and thermal comfort data within a building map so that I can quickly detect areas that require intervention.
- Super Admin
  US-08: As a super administrator, I want to manage administrator accounts (create, update, and delete) so that system access can be centrally controlled.

3) *Product Backlog:* Based on the defined user stories, Product Backlog Items (PBIs) were created to represent the implementable units of work throughout development. Each PBI was assigned a unique identifier and prioritized according to its contribution to the Minimum Viable Product (MVP), providing the basis for backlog refinement and sprint planning. The representative Product Backlog is summarized as follows:

- PBI-01: As an admin user, I want there to be a login function to ensure that only authorized individuals can access the protected information. (MVP)
- PBI-02: As a user/admin, I want the application to include a way to store data. (MVP)
- PBI-03: As an admin user, I want to view detailed sensor data over time through the form of a graph so that I can visually check information for each room. (MVP)
- PBI-04: As an admin user, I want to view recommendations on how to counter issues noted by the users or measured by the sensors. (MVP)
- PBI-05: As a user, I want the application to have a questionnaire form so that I can give feedback as to what the temperature (and other measurables) of a room are like. (MVP)
- PBI-06: As a user/admin user, I want the application to have a navigation bar to allow me to move efficiently through the application. (MVP)
- PBI-07: As an admin, I want the application to be able to handle .csv files so that I can import and export data efficiently (MVP)
- PBI-08: As an admin, I want a help/support page so that I know how to use the application.
- PBI-09: As a user, I want to observe temperature heatmaps of buildings so that I can conveniently monitor the current temperature in each location within a single view.
- PBI-10: As an admin, I want the ability to add/edit buildings & nodes so that the application is reusable and up to date.
- PBI-11: As a user, I want the ability to view the weather forecast alongside the heatmap so that I can compare the external and internal data side by side.
- PBI-12: As an admin/user, I want the application to be reliably accessible and functional in a non-local environment.

4) *Sprint Planning:* Prior to development, a pre-sprint preparation phase was conducted to establish the project foundation, including requirement elicitation, stakeholder meetings, Definition of Done (DoD), initial wireframes, and Product Backlog construction. Development was subsequently organized into seven one-week sprint iterations. At the beginning of each sprint, prioritized PBIs were selected based on MVP objectives, task dependencies, and team capacity. The sprint allocation adopted in this study is summarized as follows:

- Sprint 1: Project setup, authentication foundation, and initial database schema
- Sprint 2: Authentication, authorization, password encryption, and session management
- Sprint 3: Occupant feedback questionnaire, building management, and room management
- Sprint 4: Role-based access control and user interaction flow
- Sprint 5: Interface refinement and initial functional testing
- Sprint 6: Recommendation mechanism and initial DevOps pipeline integration
- Sprint 7: Data visualization, unit testing, system refinement, and client delivery

5) *Sprint Execution and Review:* Throughout each sprint, Daily Scrum meetings were conducted to synchronize development activities, report progress, identify impediments, and coordinate remaining work among team members. As each PBI neared completion, it was evaluated against the team's agreed Definition of Done (DoD) before being integrated into the main branch. Acceptance criteria fulfilment, successful automated test execution, and synchronization with the latest main branch were required before a merge request could be submitted for peer review.

Every merge request underwent a structured peer review using a standardized checklist agreed upon by the development team, covering functional correctness, compliance with acceptance criteria, code readability, maintainability, and documentation adequacy. A merge request was approved only after receiving independent review from at least two other team members, after which the validated implementation was merged into the main branch.

At the end of each sprint, completed PBIs were demonstrated during the Sprint Review, followed by a Sprint Retrospective to identify challenges and improvements for subsequent iterations. Client show-and-tell sessions were additionally conducted after every two sprint iterations to present the implemented functionalities and obtain stakeholder feedback. Feedback collected during these sessions was incorporated into the Product Backlog and considered during subsequent sprint planning whenever refinements or additional requirements were identified.

C. System Architecture

The system was designed using the Model–View–Controller (MVC) architectural pattern to establish a clear separation between the presentation layer, business logic, and data access components. As illustrated in Figure 2, user requests are handled by the Controller, which coordinates application logic through the Model and interacts with the Repository layer to retrieve or persist data using Spring Data JPA. Data Transfer Objects (DTOs) are employed to exchange data between layers, while the View, implemented using Thymeleaf templates, renders the processed information as dynamic HTML pages returned to the client. This layered architecture improves modularity and maintainability by separating presentation, business logic, and data access responsibilities.

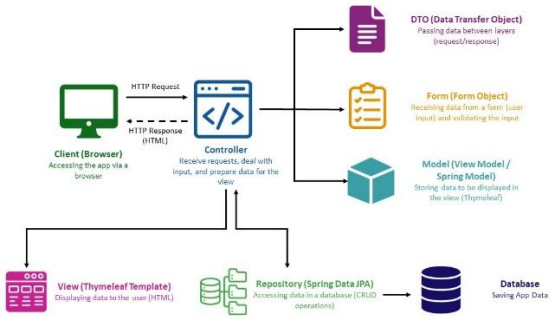

Figure 2. System Architecture

The technology stack was selected based on both the system's technical requirements and the development team's prior expertise. Spring Boot was chosen as the primary backend framework because it provides built-in support for dependency injection, auto-configuration, Spring MVC, Spring Data JPA, and Spring Security [14], while also offering sufficient scalability for IoT-based applications [15].

D. System Design

This study focuses on developing a web application that integrates and processes environmental sensor data to support adaptive thermal comfort monitoring. The design and implementation of the sensor hardware itself fall outside the scope of this study; rather, the system was designed to consume, store, and process data originating from existing sensor infrastructure.

1) *Sensor Data Source and Characteristics:* The sensor data used in this study originated from the client's existing environmental sensing infrastructure. Because a live API connection was unavailable during development, historical datasets provided by the client were used to inform the system's data structure and validate its data processing capabilities.

Two datasets were employed. The first contained temperature (Temp_C) and equivalent CO_2 concentration (CO2Equiv_ppm) measurements collected from devices identified by serial codes (e.g., LORD_1 and LORD_2). The second contained environmental and acoustic measurements,

including temperature, pressure, relative humidity, indoor air quality (IAQ), noise level (noise_dB), and peak noise level (max_bin_dB), collected from devices identified by device name and device EUI (e.g., lord-r1 to lord-r5). Both datasets were recorded at 15-minute intervals.

2) *Use Case Diagram:* The use case diagram illustrates the interactions between the system and its three user roles: General User (Occupant), Administrator, and Super Administrator. General Users can view room conditions and submit thermal comfort feedback through a web-based feedback form accessed via room-specific QR codes. Administrators monitor environmental conditions, manage buildings, rooms, and sensors, handle environmental data, and review alerts and visualizations. Super Administrators additionally manage administrator accounts and system-level settings. This role separation supports role-based access control by restricting each user to the functions relevant to their responsibilities. The complete use case diagram is presented in Figure 3.

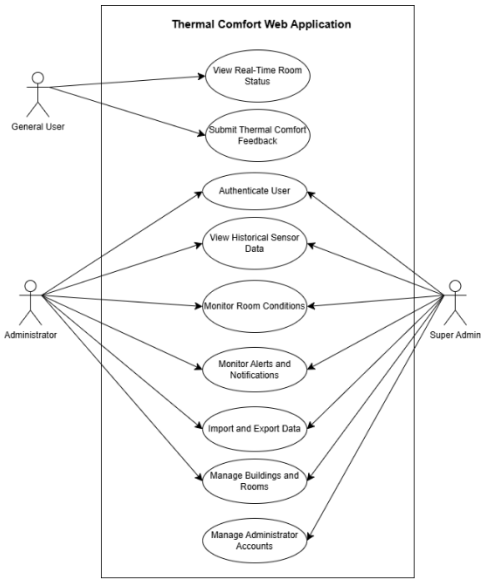

Figure 3. Usecase Diagram

3) *Sequence Diagram:* Sequence diagrams were developed to model the dynamic interactions among system components and verify that the proposed MVC architecture supports the identified functional requirements prior to implementation [16]. Figure 4 illustrates the thermal comfort feedback submission process. After an occupant accesses the room-specific feedback form by scanning a QR code, the submitted data is validated through the FeedbackForm object, processed by the FeedbackController, and stored in the database via the FeedbackRepository. Upon successful storage, the system returns a confirmation response and redirects the occupant to a thank-you page, illustrating the interaction among the user interface, controller, repository, and database within the MVC architecture.

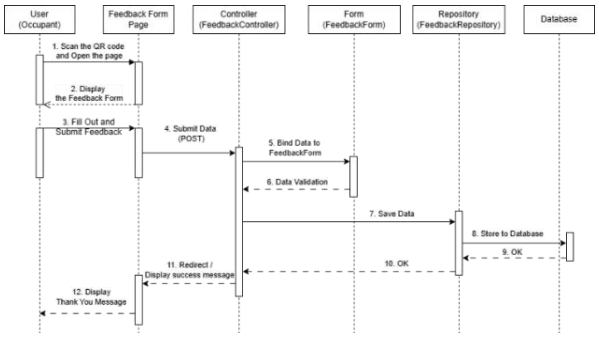

Figure 4. Sequence Diagram

4) *Database Design:* An Entity Relationship Diagram (ERD) was developed during the system design phase to define the database structure prior to implementation. The schema was established collaboratively during the early sprints and designed with reference to the historical sensor datasets provided by the client, ensuring compatibility with the expected sensor data structure should a live API integration become available in future development.

Figure 5 presents the ERD of the proposed system. The Building, Room, and Sensor entities are linked through foreign key relationships, allowing new buildings, rooms, and sensors to be registered without modifying the database schema. Historical environmental measurements are stored in the lord_sensor_data and remote_sensor_data entities, while occupant feedback is maintained in the submissions entity. Sensor-based and feedback-based recommendations are stored separately to simplify maintenance and future extension, whereas configurable environmental thresholds are maintained in the sensor_threshold entity to support recommendation generation without requiring application code changes.

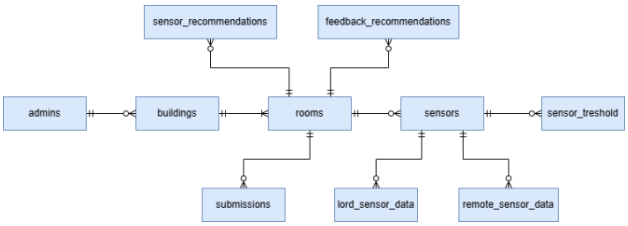

Figure 5. ERD

5) *Recommendation Mechanism:* The recommendation mechanism was designed to support adaptive thermal comfort management by combining two complementary yet independent approaches: a sensor data-based mechanism and an occupant feedback-based mechanism. The sensor data-based mechanism compares measured room temperatures against configurable threshold values stored in the database. Temperature thresholds of 21°C and 25°C were defined according to the client's operational requirements and can be modified through the administrative interface without requiring changes to the application code.

Complementing the objective sensor measurements, the occupant feedback-based mechanism generates

recommendations from aggregated thermal comfort responses collected using a five-point scale (Cold, Chilly, Comfortable, Warm, and Hot). As illustrated in Figure 6, recommendations are generated when at least three responses are received within a 15-minute interval or when the aggregation interval expires, ensuring that recommendations remain available even in low-occupancy environments. The 15-minute aggregation interval was selected to match the sensor measurement frequency, thereby maintaining temporal consistency between objective sensor observations and subjective occupant feedback.

Accordingly, the scope of this study is limited to the design and implementation of the recommendation mechanisms, while their effectiveness in improving occupant thermal comfort is left for future investigation.

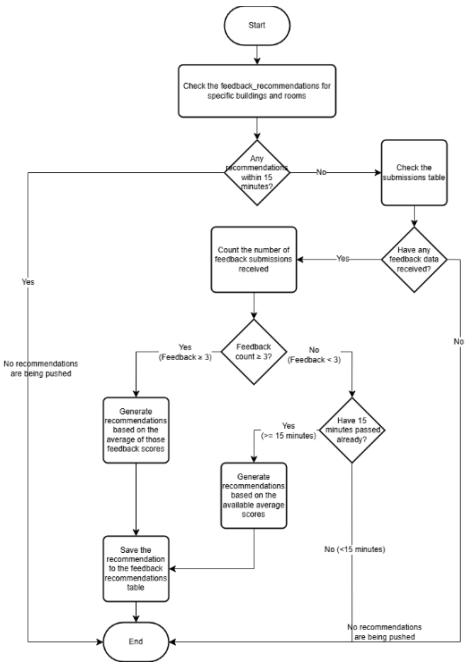

Figure 6. User-feedback-based recommendation algorithm

E. DevOps Pipeline

To complement the iterative Scrum-based development process, a DevOps pipeline was designed to automate source code integration, application building, testing, artifact management, infrastructure provisioning, and deployment throughout the software development lifecycle. The adoption of DevOps practices ensured that every accepted code change underwent a consistent validation process before deployment while reducing manual intervention and improving deployment repeatability [17], [18]. The overall DevOps workflow adopted in this study is illustrated in Figure 7.

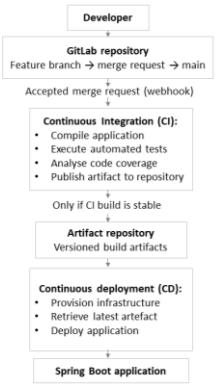

Figure 7. Devops Workflow

The pipeline begins with source code management using GitLab, where development is performed on dedicated feature branches and integrated into the main branch through peer-reviewed merge requests. Accepted merge requests automatically trigger the Continuous Integration (CI) stage, which compiles the application, executes automated tests, performs code coverage analysis, and produces a deployable JAR artifact. Once the CI stage completes successfully, the artifact is published to a centralized repository, providing version traceability and enabling subsequent deployment.

Following successful completion of the CI stage, the Continuous Deployment (CD) stage provisions the target environment using an Infrastructure as Code (IaC) approach and deploys the latest validated artifact. Separating CI and CD ensures that only validated builds are deployed while allowing deployment failures to be retried independently of the build process, thereby supporting a consistent and maintainable software delivery workflow.

F. Testing Strategy

A multi-layered testing strategy was adopted to evaluate the correctness and quality of the developed system throughout the development process. The strategy comprised unit testing, integration testing, Selenium-based automated user interface testing, and performance testing. Unit and integration testing verified the correctness of individual software components and their interactions, while Selenium-based testing validated representative end-to-end user workflows through the web interface. Performance testing assessed system responsiveness under concurrent workloads using the Application Performance Index (APDEX) as the primary evaluation metric [19].

To support continuous quality assurance, automated unit and integration tests were incorporated into the Continuous Integration (CI) pipeline, where code compilation, test execution, and code coverage analysis were performed automatically following code integration. JaCoCo generated code coverage reports as an indicator of test completeness throughout development. Selenium-based user interface testing and performance testing were executed separately because they required browser automation and workload generation beyond the scope of the CI pipeline.

The testing activities focused on validating the developed web application and its sensor data processing capabilities using the historical datasets provided by the client. Security testing, including vulnerability assessment, penetration testing, and authentication bypass testing, was outside the scope of this study. Likewise, because a live API connection to the client's sensing infrastructure was unavailable, the evaluation did not include end-to-end validation using real-time sensor transmission.

III. Result and Discussion

A. Requirement Realization

The identified functional and non-functional requirements were systematically implemented throughout the development process. All non-functional requirements were successfully realized, while eight of the nine functional requirements were fully implemented. FR-01 was the only requirement that could not be fully implemented because access to the client's production sensor API was unavailable during the study. Historical sensor datasets representative of the client's sensing infrastructure were therefore used to validate the implemented data storage, processing, visualization, and recommendation functionalities. Nevertheless, the proposed system architecture and database schema remain fully compatible with future real-time API integration without requiring structural modifications.

B. System Implementation

The following implementation results demonstrate the realization of the identified functional requirements through the developed web interfaces and system functionalities.

1) *Landing Page:* Figure 8 presents the implemented landing page, which provides separate access paths for occupants and administrative users. Occupants can access the thermal comfort questionnaire by selecting a building and room or by scanning a room-specific QR code, while administrators authenticate through the login interface.

Figure 8. Landing Page

2) *Monitoring Dashboard:* Figure 9 presents the monitoring dashboard, which consolidates environmental measurements, room status, visualizations, alerts, and recommendations within a single interface. Because a live sensor API was unavailable, the displayed measurements were populated using the historical datasets provided by the client.

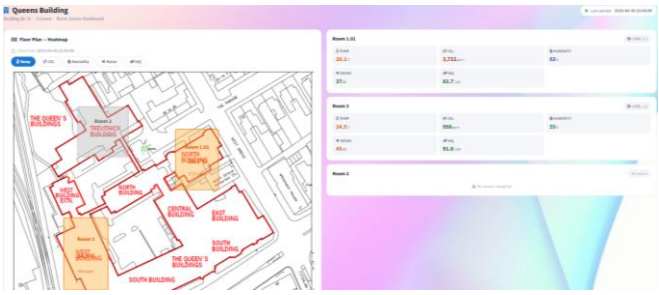

Figure 9. Monitoring Dashboard

3) *QR Code Feedback Interface:* Figure 10 presents the occupant feedback interface, which displays the current room temperature and enables occupants to submit thermal comfort responses using a five-point scale with optional comments.

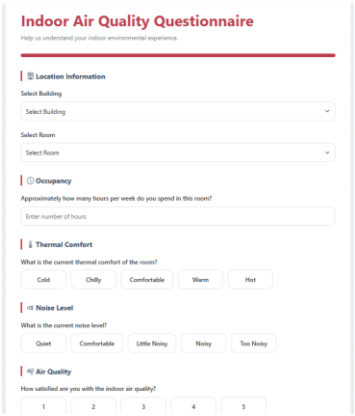

Figure 10. Feedback Page

4) *Recommendation and Alert Interface:* Figure 11 presents the recommendation interface, where administrators can review sensor-based alerts and feedback-based recommendations together with their associated location, timestamp, and urgency status.

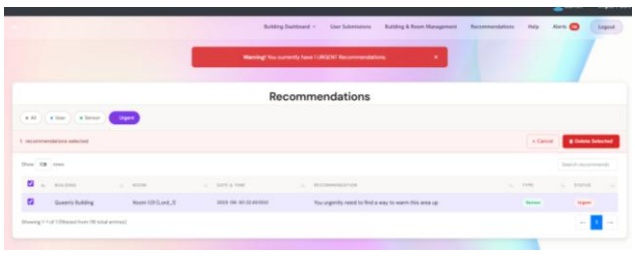

Figure 11. Recommendation and Alert System

5) *Building and Room Management Interface:* Figure 12 presents the building and room management interface, which enables administrators to register buildings, define room layouts, associate rooms with sensors, and manage existing configurations.

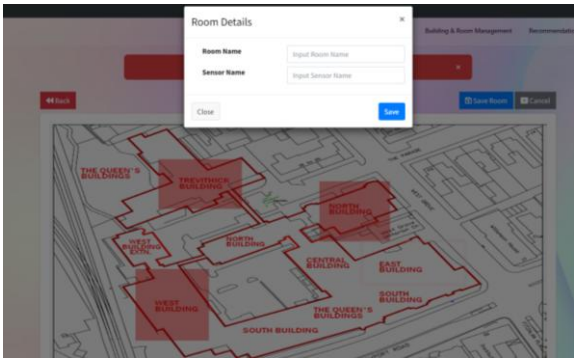

Figure 12. Building and Room Management Page

C. *Validation of Building and Room Extensibility*

One of the primary design objectives of the proposed platform was to support deployment across multiple buildings without requiring modifications to the application architecture or database schema. The newly registered entities were then verified to be immediately available across the occupant feedback, monitoring, and recommendation modules without requiring modifications to the application code or database schema.

Figure 13 presents the implemented Building Management interface, which enables administrators to manage registered buildings and initiate the registration of new buildings through the Add Building function. During registration, administrators provide the required building information, including the building floor plan, which serves as the spatial reference for subsequent room configuration.

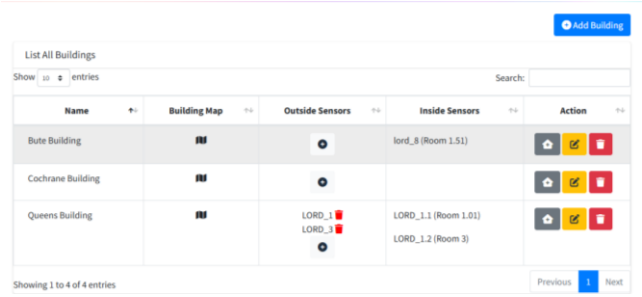

Figure 13. Building Registration Interface

Figure 14 demonstrates the configuration of rooms within a newly registered building. After the building was created, administrators defined room boundaries on the uploaded floor plan and associated each room with its corresponding sensor. This configuration establishes the Building–Room–Sensor relationship defined in the proposed database schema and enables independent management of rooms within each building.

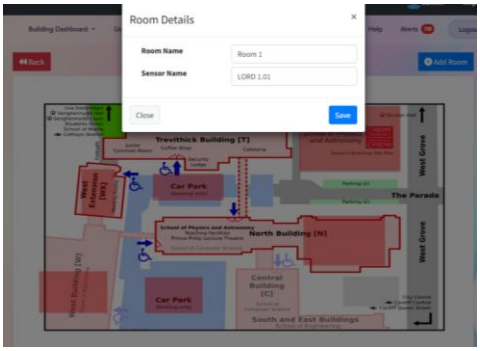

Figure 14. Add Room and Sensor Details

Following registration, the newly created buildings and rooms were automatically integrated into the remaining system components. As illustrated in Figure 15, the newly registered "Abacws Building" immediately became available in the occupant feedback interface without requiring additional configuration, allowing occupants to submit thermal comfort responses for the newly added location. Figure 16 further demonstrates that environmental measurements associated with the configured sensors were successfully incorporated into the monitoring dashboard and recommendation module, confirming that newly configured buildings and rooms were fully integrated into the system.

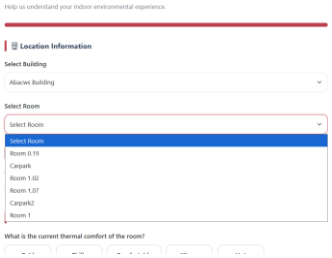

Figure 15. Feedback Questionnaire

Figure 16. Monitoring Dashboard of Abacws Building

D. *Scrum-Based Development Results*

The proposed platform was developed over seven one-week Scrum sprint iterations, resulting in the incremental implementation of system functionality throughout the project. Development began with foundational components, including project setup, authentication, and database configuration, before progressing to building and room management, environmental monitoring, recommendation generation, and supporting administrative features. Figure 17 presents the implemented Product Backlog in GitLab,

illustrating the completed Product Backlog Items (PBIs) tracked throughout the development process.

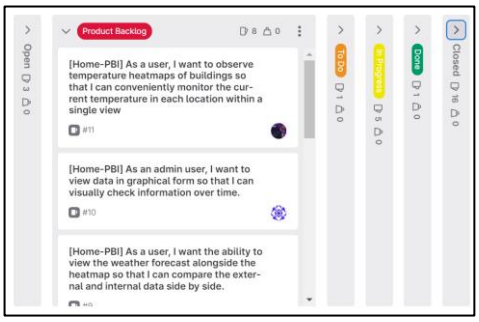

Figure 17. Product Backlog Managed in GitLab

All planned Product Backlog Items were completed through successive sprint iterations and integrated into the final system. Continuous peer review, automated testing, and periodic client demonstration sessions supported incremental refinement of the implemented functionalities and ensured that stakeholder feedback was incorporated throughout development.

E. DevOps Pipeline Implementation

The proposed DevOps pipeline was implemented as a two-stage Jenkins workflow comprising Continuous Integration (CI) and Continuous Deployment (CD). The CI stage automated application building, testing, and artifact publication, while the CD stage provisioned the target OpenStack environment and deployed the latest validated application artifact. Figure 18 presents the implemented two-stage Jenkins workflow.

Figure 18. Jenkins Jobs Implementing the CI/CD Pipeline

Development was managed through GitLab using feature branches and peer-reviewed merge requests. Once a merge request targeting the main branch had been accepted, a GitLab webhook automatically triggered the CI pipeline. Ordinary push events and newly opened merge requests were excluded from the trigger configuration, ensuring that only reviewed and approved code changes initiated the automated build process. The implemented trigger configuration is shown in Figure 19.

Following pipeline execution, the CI stage compiled the application using Gradle, executed the automated test suite, and generated JUnit test reports together with JaCoCo code coverage reports. These reports were integrated directly into Jenkins, providing immediate feedback on build correctness and test coverage after each accepted code integration, as illustrated in Figure 20.

Figure 19. Jenkins Build Trigger Configuration

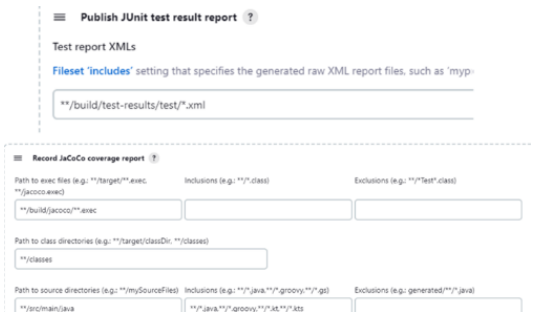

Figure 20. Jenkins Post-build Configuration for JUnit and JaCoCo

Following successful completion of the CI stage, the generated JAR artifact was automatically published to JFrog Artifactory using the Gradle Artifactory plugin. Timestamp-based versioning (TCApp-YYYYMMDD-HHMMSS) was adopted to distinguish successive builds and maintain traceability between software versions. The publication configuration and the resulting versioned artifacts are presented in Figures 21 and 22.

```
// Further details as to where artifacts should be sent (using Maven publications).
artifactory {
    contextUrl = 'http://10.72.102.219:8082'
    publish {
        repository {
            repokey 'example-repo-local'
            maven = true
        }

        defaults {
            publications('mavenJava')
            publishArtifacts = true
        }
        resolve {
            repokey = 'mavenCentral'
        }
    }
}

// Function to create datetime that can be used in naming the artifacts (for future convenience).
def getCustomArtifactId() {
    def dateFormat = new SimpleDateFormat("yyyyMMdd-HHmmss")
    def timestamp = dateFormat.format(new Date())
    return "TCapp-$timestamp"
}

// Creating datetime to use for artifact ID.
def customArtifactId = getCustomArtifactId()
```

Figure 21. Gradle Configuration for Artifact Publication

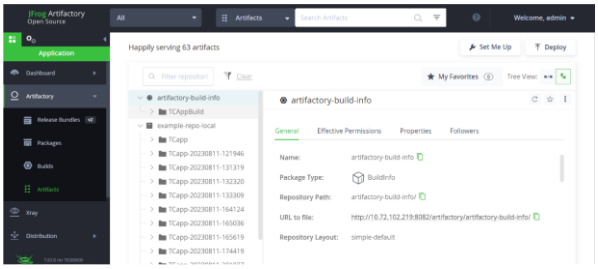

Figure 22. Versioned Artifacts Stored in JFrog Artifactory

Successful completion of the CI stage automatically triggered the downstream CD job, which provisioned the target OpenStack environment and deployed the latest validated artifact. To ensure secure deployment, OpenStack credentials were stored in the Jenkins Credentials Manager and injected into the deployment process through credential bindings rather than embedded directly in deployment scripts. The deployment dependency and credential configuration are illustrated in Figures 23 and 24.

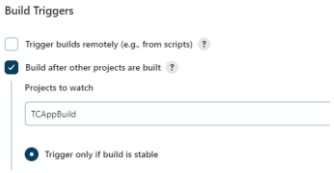

Figure 23. Deployment Job Dependency Configuration

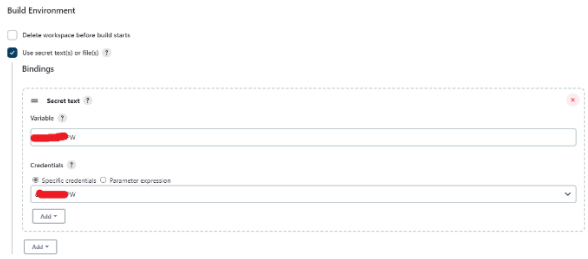

Figure 24. Jenkins Credential Binding for Deployment

F. *Testing Results*

To evaluate the developed platform comprehensively, multiple testing approaches were employed to verify functional correctness, integration behavior, user interface functionality, and system performance. The evaluation comprised unit testing, integration testing, Selenium-based automated user interface testing, and performance testing.

1) *Unit and Integration Testing:* Unit and integration testing were conducted to verify the correctness of individual software components and their interactions across the application layers. A total of 25 automated tests were executed, comprising 9 unit tests, 12 web-layer integration tests, and 4 full application integration tests, covering the authentication, building and room management, recommendation, questionnaire, and application infrastructure modules. As summarized in Table II, all automated tests passed successfully, confirming the correctness of both individual components and their interactions within the complete application.

2) *Selenium-Based User Interface Testing:* Selenium-based automated testing was conducted to validate representative end-to-end user workflows through browser simulation. Rather than exhaustively testing every interface, two representative test scenarios were implemented: LandingPageAndFillQuestionnaire, representing the primary occupant workflow, and UserManagement, representing core administrator account management. The first scenario verified the complete occupant workflow, including landing page navigation, room selection, questionnaire completion, form submission, and successful redirection after submission. The second scenario verified administrator authentication, account creation, account verification, account deletion, and logout through the administrative interface. As summarized in Table III, both scenarios completed successfully without failures, confirming that the implemented user interface behaved correctly for the evaluated representative workflows.

TABLE II
SUMMARY OF AUTOMATED UNIT AND INTEGRATION TEST RESULTS

Testing Level	Components Covered	Number of Tests	Result
Unit testing	DTOs and Form Objects (Building, Login, Recommendation, Questionnaire, and Room)	9	Pass
Web-layer Integration Testing	Building, Login, Recommendation, Questionnaire, and Room Controllers	12	Pass
Full Application Integration Testing	Authentication, Building Controller, HTTP Connectivity, and Application Context	4	Pass
Total		**25**	**Pass**

TABLE III
SUMMARY OF SELENIUM AUTOMATED TESTING RESULTS

Scenario	Functionality Verified	Outcome
LandingPageAndFillQuestionnaire	Navigation, room selection, questionnaire submission, and redirection	Pass
UserManagement	Authentication, account creation, verification, deletion, and logout	Pass

3) *Performance Testing:* Performance testing was conducted using Apache JMeter 5.6.3 in the local development environment to evaluate system responsiveness under simulated concurrent workloads. The local environment was selected to provide a controlled and reproducible testing setup while avoiding interference with shared deployment infrastructure. Two thread groups representing the General User and Administrator workflows

were executed. Performance was evaluated using the Application Performance Index (APDEX), where values closer to 1 indicate higher user satisfaction with application response times [19].

TABLE IV
JMETER THREAD GROUP CONFIGURATION

Workflow	Users	Ramp-up	Loops	Endpoints
General User	50	30 seconds	3	Landing page / login page access, ListAllBuildings, SubmitQuestionnaire, getFeedbackByRoomSelect
Administrator	10	10 seconds	3	LoginPage, LoginPost, ListAllBuildings, getFeedbackByRoomSelect, getAllQuestionnaireData, getThermalComfortGraph, Dashboard, adminHome, Logout

TABLE V
GENERAL USER PERFORMANCE RESULTS

Endpoint	APDEX	Mean (ms)	Median (ms)	P90 (ms)	Max (ms)	Error %
Landing / Login Page Request	0.993	30.25	7.5	26.8	1423	0%
ListAllBuildings	1.000	17.47	10.0	37.0	115	0%
SubmitQuestionnaire	0.990	133.61	94.0	263.2	528	0%
getFeedbackByRoomSelect	1.000	54.10	41.0	100.9	165	0%
Total	**0.997**	**42.09**	**11.0**	**97.0**	**1423**	**0%**

The General User workflow achieved an overall APDEX score of 0.997 across 900 requests with a 0% error rate, indicating that the system maintained a high level of responsiveness under the evaluated testing conditions. The questionnaire submission endpoint recorded the highest average response time, although it remained within acceptable performance limits.

The Administrator workflow achieved an overall APDEX score of 0.960 across 390 requests, also with a 0% error rate. Among the evaluated endpoints, the authentication request recorded the highest response time, potentially reflecting the additional processing involved in session establishment and user authentication, whereas dashboard and monitoring operations remained within acceptable response-time thresholds.

TABLE VI
ADMINISTRATOR PERFORMANCE RESULTS

Endpoint	APDEX	Mean (ms)	Median (ms)	P90 (ms)	Max (ms)	Error %
LoginPage	0.967	82.20	16.0	62.0	1423	0%
LoginPost	0.633	636.03	565.0	1273.1	1713	0%
ListAllBuildings	1.000	27.30	23.0	52.9	74	0%
getFeedbackByRoomSelect	1.000	16.73	17.0	24.7	28	0%
getAllQuestionnaireData	1.000	57.77	47.5	122.6	126	0%
getThermalComfortGraph	1.000	16.57	13.5	32.0	44	0%
Dashboard	1.000	102.00	69.0	378.0	412	0%
adminHome	1.000	159.60	155.0	298.7	315	0%
Logout	1.000	16.87	15.0	24.0	40	0%
Total	**0.960**	**136.71**	**31.0**	**414.0**	**1713**	**0%**

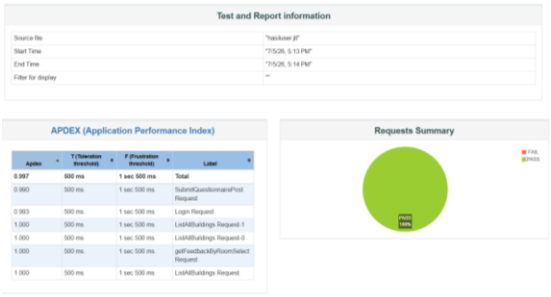

Figure 25. APDEX Report for the General User Workflow

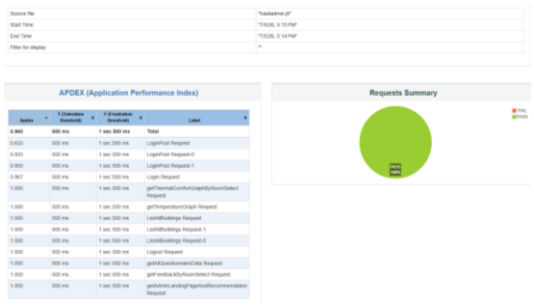

Figure 26. APDEX Report for the Administrator Workflow

G. *Validation Using Historical Sensor Dataset*

Because a live API connection to the client's sensing infrastructure was unavailable during the study, historical

sensor datasets supplied by the client were imported through the platform's web-based import interface to validate its data processing capabilities. The imported data populated the monitoring dashboard, supported historical data visualization, and served as the basis for generating recommendation outputs.

Following the import process, the lord_sensor_data table contained 155,022 records, while the remote_sensor_data table contained 1,885,540 records. In total, the platform successfully stored and managed 2,040,562 historical sensor records across the two dataset formats without requiring modifications to the underlying database schema or application architecture. The successful import and management of these datasets demonstrate the platform's ability to accommodate historical environmental datasets while supporting the implemented monitoring, visualization, and recommendation functionalities. Although this validation relied on historical datasets rather than real-time sensor transmission, the results demonstrate that the proposed data management architecture is compatible with the client's sensing data and can support future integration with a live sensor API.

H. Discussion

The results demonstrate that the proposed platform successfully integrates environmental sensor data and occupant feedback into a configurable web-based monitoring system while supporting incremental software development through Scrum and automated software delivery through DevOps. The developed platform not only satisfies the identified functional and non-functional requirements but also demonstrates how software engineering practices can be systematically incorporated into the development of IoT-based monitoring applications.

In terms of the development process, the proposed approach differs from previous IoT-based environmental monitoring studies, which primarily focused on system implementation and sensor functionality with limited discussion of the supporting software engineering lifecycle [7], [8]. While previous studies have applied Scrum or DevOps independently in different contexts [9]–[11], this study demonstrates their combined application within a single development process, providing a structured workflow from requirements analysis through deployment and system validation. As such, the study contributes empirical evidence on the integration of Scrum and DevOps for the development of an IoT-based adaptive thermal comfort monitoring platform.

Beyond the development process, the proposed system architecture offers a further contribution in terms of deployment flexibility. By separating Building, Room, and Sensor as independent entities, the platform allows new buildings and rooms to be incorporated through configuration alone without requiring modifications to the application code or database schema. The validation results presented in this study demonstrate that this architecture supports deployment across multiple buildings, addressing the reconfiguration challenges identified in prior occupant-centric thermal comfort approaches.

Nevertheless, several limitations should be acknowledged. Although the platform generates recommendations from both environmental sensor measurements and occupant feedback, the effectiveness of these recommendations in improving occupant thermal comfort was not evaluated. Security evaluation, including vulnerability assessment and penetration testing, was likewise outside the scope of this study. The two recommendation mechanisms also operate independently rather than as a unified decision-support model, and system validation relied on historical sensor datasets because a live sensor API was unavailable during development. Future work will therefore focus on evaluating occupant satisfaction, integrating live sensor APIs, conducting security evaluation, and developing a unified recommendation model that combines sensor measurements with occupant feedback.

IV. CONCLUSION

This study developed an adaptive thermal comfort monitoring platform that integrates environmental sensor data and occupant feedback within a configurable web-based system. Scrum supported incremental implementation across seven one-week sprints, while the DevOps pipeline automated application building, testing, artifact management, infrastructure provisioning, and deployment. The entity-based architecture also enabled new buildings and rooms to be added without modifying the application code or database schema.

The platform was evaluated through 25 automated unit and integration tests, two Selenium-based end-to-end scenarios, and Apache JMeter performance testing, with all automated tests passing successfully. Performance testing demonstrated high responsiveness, with APDEX scores of 0.997 and 0.960 for the General User and Administrator workflows, respectively, and zero errors across all tested endpoints. The platform also successfully validated its sensor data processing, storage, visualization, and recommendation capabilities using historical datasets supplied by the client.

Nevertheless, validation was limited to historical sensor datasets because access to the client's live sensor API was unavailable during the study. Future work should therefore integrate the client's sensing infrastructure, evaluate occupant satisfaction with the generated recommendations, conduct comprehensive security and production-environment performance evaluations, and develop a unified recommendation model that integrates environmental sensor data with occupant feedback.

REFERENCES

[1] J. C. Solano, E. Caamaño-Martín, L. Olivieri and D. Almeida-Galárraga, "HVAC systems and thermal comfort in buildings

climate control: An experimental case study," *Energy Reports*, vol. 7, pp. 269-277, 2021.

[2] A. Soleimanijavid, I. Konstantzos and X. Liu, "Challenges and opportunities of occupant-centric building controls in real-world implementation: A critical review," *Energy and Buildings*, vol. 308, p. 113958, 2024.

[3] V. Sharma and V. Mistry, "Human-centric HVAC control: Balancing comfort and energy efficiency," *European Journal of Advances in Engineering and Technology*, vol. 10, no. 10, pp. 42-48, 2023.

[4] G. Marques, J. Saini, M. Dutta, P. K. Singh and W.-C. Hong, "Indoor Air Quality Monitoring Systems for Enhanced Living Environments: A Review toward Sustainable Smart Cities," *Sustainability*, vol. 12, no. 10, 2020.

[5] I. Lavrinovica, J. Judvaitis, D. Laksis, M. Skromule and K. Ozols, "A Comprehensive Review of Sensor-Based Smart Building Monitoring and Data Gathering Techniques," *Applied Sciences*, vol. 14, no. 21, p. 10057, 2024.

[6] M. Pritoni, K. Salmon, A. Sanguinetti, J. Morejohn and M. Modera, "Occupant thermal feedback for improved efficiency in university buildings," *Energy and Buildings*, vol. 144, pp. 241-250, 2017.

[7] H. Susilawati, A. N. Andiyani and S. Nurpadillah, "Rancang Bangun Sistem Monitoring dan Kendali Suhu Ruangan Berbasis Internet of Things," *CYCLOTRON : Jurnal Teknik Elektro*, vol. 6, no. 1, pp. 55-60, 2023.

[8] P. D. Nugraha, R. Soekarta and I. Amri, "Rancang Bangun Alat Monitoring Suhu Dan Kelembaban Berbasis Internet Of Things (IOT) Pada Gudang Obat Rumah Sakit Aryoko Sorong," *Framework : Jurnal Ilmu Komputer dan Informatika*, vol. 2, no. 1, pp. 21-31, 2023.

[9] A. S. Kirsan, A. H. D. Putra and V. F. Insanittaqwa, "Rancang Bangun Alat Pendeteksi Kebakaran Berbasis Internet Of Things," in *SEMIOTIKA*, Balikpapan, 2023.

[10] M. H. Ekasari, D. Diana and Y. I. Chandra, "Rancang Bangun Aplikasi E-Commerce Menggunakan Model Model DevOps Berbasis Web," *Jurnal Esensi Infokom*, vol. 9, no. 1, pp. 41-49, 2025.

[11] S. M. R. Al Masud, M. Masnun, M. A. Sultana, A. Sultana, F. Ahmed and N. Begum, "DevOps Enabled Agile: Combining Agile and DevOps Methodologies for Software Development," *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 13, no. 11, pp. 278-283, 2022.

[12] A. N. Dugbartey and O. Kehinde, "Optimizing project delivery through agile methodologies: Balancing speed, collaboration and stakeholder engagement," *World Journal of Advanced Research and Reviews*, vol. 25, no. 1, pp. 1237-1257, 2025.

[13] C. Verwijs and D. Russo, "A Theory of Scrum Team Effectiveness," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1-51, 2023.

[14] S. Modugu and H. Farhat, "Implementation of the Internet of Things Application Based on Spring Boot Microservices and REST Architecture," in *Proceedings of the Computational Methods in Systems and Software*, Online, 2020.

[15] F. Teng and Q. Wu, "Design and Implementation of the Information System of Retired Veteran Cadres Bureau Based on SpringBoot Framework," in *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, Guangzhou, 2021.

[16] P. Mani and M. Prasanna, "Test case generation for embedded system software using UML interaction diagram," *Journal of Engineering Science and Technology*, vol. 12, no. 4, pp. 860-874, 2017.

[17] N. Ruseno, Muhidin and S. Kurniawan, "Penerapan Devops (Development Operations) Dalam Pengembangan Aplikasi Untuk Meningkatkan Kecepatan Delivery Software," *Jurnal PASTI: Jurnal Publikasi Artikel Sistem Teknologi Informasi*, vol. 1, no. 1, pp. 1-10, 2025.

[18] S. Yang, "The Impact of Continuous Integration and Continuous Delivery on Software Development Efficiency," *Journal of Computer, Signal, and System Research*, vol. 2, no. 3, pp. 59-68, 2025.

[19] R. Dhuny and F. Ying, "Enhancing Education Accessibility: Portable Microservers for Computer-Based Testing in Resource-Constrained Environments," in *2025 22nd International Learning and Technology Conference (L&T)*, Jeddah, 2025.