



REPUBLIC OF TUNISIA  
THE MINISTRY OF HIGHER EDUCATION  
AND SCIENTIFIC RESEARCH  
THE UNIVERSITY OF CARTHAGE  
THE HIGHER INSTITUTE OF INFORMATION AND  
COMMUNICATION TECHNOLOGIES



## END-OF-STUDY PROJECT REPORT

Submitted in Partial Fulfillment of the Requirements for the BACHELOR  
DEGREE IN COMPUTER SCIENCE

Field of Study : Software Engineering and Information Systems

---

# SARA : Smart Analysis and Research Assistant

---

*By*

MELIK HAJLAWI  
YOSR DERBELI

Carried out through the joint efforts of ISTIC and Aizu University



Publicly defended on May 31, 2025 in front of the jury members:

|                          |   |
|--------------------------|---|
| Chairman:                | Mr. Mohamed Wahb OURTANI, President, CONECT INTECH  |
| Reporter:                | Mrs. Amira BELHEDI, Teacher, ISTIC                  |
| Examiner:                | Mr. Moez ATTIA, Teacher, ISTIC                      |
| Professional Supervisor: | Mr. Rentaro YOSHIOKA, Professor, University of AIZU |
| Academic Supervisor:     | Mrs. Asma NAJJAR, Teacher, ISTIC                    |

Academic Year: 2024-2025



REPUBLIC OF TUNISIA  
THE MINISTRY OF HIGHER EDUCATION  
AND SCIENTIFIC RESEARCH  
THE UNIVERSITY OF CARTHAGE  
THE HIGHER INSTITUTE OF INFORMATION AND  
COMMUNICATION TECHNOLOGIES



## END-OF-STUDY PROJECT REPORT

Submitted in Partial Fulfillment of the Requirements for the BACHELOR

DEGREE IN COMPUTER SCIENCE

Field of Study : Software Engineering and Information Systems

---

# SARA : Smart Analysis and Research Assistant

---

*By*

MELIK HAJLAWI

YOSR DERBELI

Carried out through the joint efforts of ISTIC and Aizu University



### AUTHORIZATION OF GRADUATION PROJECT REPORT SUBMISSION:

Professional Supervisor:

Academic Supervisor:

Issued on :

Issued on :

Signature:

Signature:

# Acknowledgements

We would like to express our sincere gratitude to our thesis supervisor, Ms. Asma NAJJAR, and our co-supervisor, Mr. Rentaro YOSHIOKA, for their insightful guidance, constructive feedback, and unwavering encouragement throughout the course of this project. Their expertise and support were invaluable at every stage of our work.

We are also deeply grateful to our families for their constant patience, understanding, and moral support. Their belief in us provided the motivation and resilience needed to complete this graduation project.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Project Context</b>  | <b>3</b>  |
| 1.1      | Introduction . . . . .  | 3         |
| 1.2      | Presentation of the Host Organization . . . . .                                       | 3         |
| 1.2.1    | The Higher Institute of Information and Communications Technologies (ISTIC) . . . . . | 3         |
| 1.2.2    | The University of Aizu . . . . .  | 4         |
| 1.3      | Project Overview and Objectives . . . . .   | 4         |
| 1.3.1    | Motivation and Problem Statement . . . . .  | 4         |
| 1.3.2    | Project Goals . . . . .   | 5         |
| 1.4      | Existing Systems . . . . .  | 5         |
| 1.4.1    | The OSEMN Pipeline Framework . . . . .  | 5         |
| 1.4.2    | Overview of Prominent Existing Systems . . . . .                                      | 6         |
| 1.5      | Critical Assessment of Existing Systems . . . . .                                     | 8         |
| 1.5.1    | Definition of Assessment Criteria . . . . .   | 8         |
| 1.5.2    | Assessment of Existing systems . . . . .  | 9         |
| 1.6      | Proposed Solution . . . . .   | 10        |
| 1.7      | UML Language . . . . .  | 13        |
| 1.8      | Conclusion . . . . .  | 13        |
| <b>2</b> | <b>Specification of Requirements</b>  | <b>14</b> |
| 2.1      | Introduction . . . . .  | 14        |
| 2.2      | Functional Requirements . . . . .   | 14        |
| 2.2.1    | Obtain . . . . .  | 14        |
| 2.2.2    | Scrub . . . . .   | 14        |
| 2.2.3    | Explore . . . . .   | 15        |
| 2.2.4    | Model . . . . .   | 15        |
| 2.2.5    | iNterpret . . . . .   | 15        |
| 2.3      | Non-Functional Requirements . . . . .   | 15        |
| 2.4      | Actors and their Roles . . . . .  | 15        |
| 2.5      | Use Case Diagram . . . . .  | 16        |
| 2.5.1    | External api Use Case: Provide Data . . . . .   | 20        |
| 2.5.2    | Administrator Use Case: Manage Reference Data . . . . .                               | 21        |
| 2.5.3    | Administrator Use Case: Manage Reference Data . . . . .                               | 21        |
| 2.6      | Conclusion . . . . .  | 21        |
| <b>3</b> | <b>Conceptual Study</b>   | <b>22</b> |
| 3.1      | Introduction . . . . .  | 22        |
| 3.2      | Component Overview . . . . .  | 22        |

|          |  |           |
|----------|--|-----------|
| 3.3      | Authentication Flow: Sign-In Use Case . . . . .                        | 25        |
| 3.3.1    | Sequence Diagram of "Sign-In" use case . . . . .                       | 25        |
| 3.3.2    | Sign-In Process Description . . . . .                                  | 25        |
| 3.4      | Visualize Historical Data: Conceptual Design . . . . .                 | 27        |
| 3.4.1    | Sequence Diagram of "Visualize Historical Data" use case . . . . .     | 27        |
| 3.4.2    | Visualize Dataset Data . . . . .                                       | 27        |
| 3.4.3    | Visualize Sensor Data . . . . .  | 29        |
| 3.5      | Real-Time Monitoring: Conceptual Design . . . . .                      | 35        |
| 3.5.1    | Event-Driven Architecture . . . . .                                    | 35        |
| 3.5.2    | Sequence Diagram of The "Visualize Real-Time Data" use case. . . . .   | 35        |
| 3.6      | Manage Custom Datasets: Conceptual Design . . . . .                    | 38        |
| 3.6.1    | Sequence Diagram of the "Create Dataset" use case . . . . .            | 38        |
| 3.6.2    | Sequence Diagram of the "Update Dataset Visibility" Use Case . . . . . | 38        |
| 3.7      | Manage Reference Data: Conceptual Design . . . . .                     | 41        |
| 3.7.1    | Overview of Reference Data . . . . .                                   | 41        |
| 3.7.2    | Create Site . . . . .  | 41        |
| 3.7.3    | Update Site . . . . .  | 43        |
| 3.7.4    | Delete Site . . . . .  | 44        |
| 3.8      | Global Class Diagram . . . . .   | 45        |
| 3.9      | Conclusion . . . . .   | 46        |
| <b>4</b> | <b>Implementation</b>  | <b>47</b> |
| 4.1      | Introduction . . . . .   | 47        |
| 4.2      | Software Environment . . . . .   | 47        |
| 4.3      | Frameworks and Technologies Used . . . . .                             | 47        |
| 4.4      | User Authentication and Sign-Up Implementation . . . . .               | 47        |
| 4.4.1    | Login Interface . . . . .  | 50        |
| 4.4.2    | Sign-Up Interface . . . . .  | 50        |
| 4.5      | Real-Time Monitoring Implementation . . . . .                          | 51        |
| 4.5.1    | Interface Overview . . . . .   | 51        |
| 4.5.2    | User Controls . . . . .  | 52        |
| 4.5.3    | Visualization Example: BLE and Radio Streams . . . . .                 | 54        |
| 4.6      | Historical Data Visualization Implementation . . . . .                 | 54        |
| 4.6.1    | Data Sources Panel . . . . .   | 55        |
| 4.6.2    | Configuration Panel . . . . .  | 55        |
| 4.6.3    | Visualization Card Layout . . . . .                                    | 57        |
| 4.6.4    | Example: Thermal Map Snapshot at Aizu Cafeteria . . . . .              | 57        |
| 4.6.5    | Same-Type Comparison . . . . .   | 57        |
| 4.6.6    | Cross-Type Comparison . . . . .  | 59        |
| 4.6.7    | Data Export Feature . . . . .  | 60        |
| 4.7      | Custom Dataset Management Interface . . . . .                          | 60        |
| 4.7.1    | Dataset Creation Wizard . . . . .                                      | 60        |
| 4.7.2    | Dataset Management Dashboard . . . . .                                 | 62        |
| 4.8      | Admin Reference Data Management . . . . .                              | 62        |
| 4.8.1    | Administrative Dashboard . . . . .                                     | 65        |
| 4.8.2    | Location Management Interface . . . . .                                | 65        |
| 4.9      | Conclusion . . . . .   | 67        |
|          | <b>General Conclusion</b>  | <b>68</b> |

|              |    |
|--------------|----|
| Netography   | 70 |
| Bibliography | 73 |

# List of Figures

|      |   |    |
|------|---|----|
| 1.1  | The OSEMN Pipeline: A framework for data science workflows, including Obtain, Scrub, Explore, Model, and iNterpret. . . . .   | 6  |
| 1.2  | Gartner Magic Quadrant for Analytics and Business Intelligence Platforms, 2024 [42]. . . . .  | 7  |
| 1.3  | One of the four sensor modules installed at ISTIC: Classroom A006 . . . . .   | 11 |
| 1.4  | Sensor Modules Placement Illustration in Classroom M5 . . . . .   | 11 |
| 2.1  | Use Case Diagram of the SARA Platform . . . . .   | 17 |
| 3.1  | Component Diagram of the SARA platform . . . . .  | 23 |
| 3.2  | Sequence Diagram of the Sign In use case . . . . .  | 26 |
| 3.3  | Historical Data Visualization Sequence Diagram . . . . .  | 28 |
| 3.4  | Processing Microservice Activity Diagram . . . . .  | 31 |
| 3.5  | Sequence diagram of the Real-Time Monitoring workflow. . . . .  | 36 |
| 3.6  | Sequence diagram for creating a custom dataset. . . . .   | 39 |
| 3.7  | Sequence diagram for updating dataset visibility. . . . .   | 40 |
| 3.8  | Sequence diagram for the Create Site workflow. . . . .  | 42 |
| 3.9  | Sequence diagram for the Update Site workflow. . . . .  | 43 |
| 3.10 | Sequence diagram for the Delete Site workflow. . . . .  | 44 |
| 3.11 | Global Class Diagram of the SARA Platform . . . . .   | 45 |
| 4.1  | Login interface prompting for email and password. . . . .   | 50 |
| 4.2  | Sign-Up interface for new user registration. . . . .  | 51 |
| 4.3  | Real-Time Monitoring: “Occupancy” tab with live-updating cards. . . . .   | 52 |
| 4.4  | Real-Time Monitoring: “Environmental” tab with luminance chart expanded and humidity paused. . . . .  | 53 |
| 4.5  | Real-Time Monitoring: “Thermal” tab with thermalmap and temperature charts both expanded. . . . .   | 53 |
| 4.6  | Control panel for an individual sensor stream. . . . .  | 54 |
| 4.7  | Radio Device Count (top) and BLE Signal Strength Detection (bottom) in Aizu classroom M5—raw counts and events per 60 s interval, aligned by time and device IMEI. . . . .  | 55 |
| 4.8  | Annotated overview of the General Trends interface: (1) Data Sources panel, (2) Configuration panel, (3) Visualization card area showing minute microphone readings in classroom A006 between 2,3 May 2025, (4) Controls bar. . . . . | 56 |
| 4.9  | Annotated overview of the General Trends interface in custom-datasets mode. . . . .   | 56 |
| 4.10 | Thermal map of Aizu University cafeteria on October 12, 2023 at 06:00 AM (temperature range: 20–26 °C). . . . .   | 58 |
| 4.11 | Hourly average luminance in ISTIC A006 for 1,3 May 2025. . . . .  | 58 |

|      |  |    |
|------|--|----|
| 4.12 | Same-Type Comparison: temperature streams from classroom A006 and classroom M5 over 5 days span. . . . . | 59 |
| 4.13 | Cross-Type Comparison: individual sub-charts for humidity, motion, and temperature. . . . .              | 59 |
| 4.14 | Dataset Creation Wizard: (1) Data input, (2) Schema definition, (3) Preview & Create. . . . .            | 60 |
| 4.15 | Dataset Creation Wizard, Schema step: schema input and <i>show errors</i> button                         | 61 |
| 4.16 | Dataset Creation Wizard, Preview step: table view and <i>Create</i> button. . . .                        | 62 |
| 4.17 | Dataset Management Dashboard: cards for each dataset with metadata. . . .                                | 63 |
| 4.18 | Dataset Detail view-2: metadata display and management actions. . . . .                                  | 63 |
| 4.19 | Dataset Detail view-2: metadata display and management actions. . . . .                                  | 64 |
| 4.20 | Dataset Detail view-3: deleting a dataset . . . . .  | 64 |
| 4.21 | Admin dashboard showing summary statistics and location-centric hierarchy.                               | 65 |
| 4.22 | Location Management table showing editable entries and action controls. . .                              | 66 |
| 4.23 | Location Management Interface: Update Modal Window . . . . .   | 66 |
| 4.24 | Add Location modal for defining new location metadata. . . . .   | 67 |



# List of Tables

|     |   |    |
|-----|---|----|
| 1.1 | Critical Assessment of Data Analytics Platforms . . . . .           | 9  |
| 1.2 | Description of sensor types within each sensor module . . . . .     | 11 |
| 2.1 | Actors and Roles in the Smart Research Assistant Platform . . . . . | 16 |
| 4.1 | Software Environment Tools . . . . .                                | 48 |
| 4.2 | Key Technologies and Role in SARA development . . . . .             | 49 |

# List of Acronyms

|                   |  |
|-------------------|--|
| <b>API</b>        | Application Programming Interface                              |
| <b>AI</b>         | Artificial Intelligence  |
| <b>IoT</b>        | Internet of Things   |
| <b>ML</b>         | Machine Learning   |
| <b>ISTIC</b>      | Higher Institute of Information and Communications Technologie |
| <b>SARA</b>       | Smart Analysis and Research Assistant                          |
| <b>UoA</b>        | University of Aizu   |
| <b>UML</b>        | Unified Modeling Language                                      |
| <b>OSEMN</b>      | Obtain, Scrub, Explore, Model, and eNterpret                   |
| <b>LMD</b>        | Bachelor, Master, Doctorate                                    |
| <b>ICT</b>        | Information and Communication Technology                       |
| <b>NEC</b>        | Nippon Electric Company  |
| <b>BI</b>         | Business Intelligence  |
| <b>VizQL</b>      | Visual Query Language  |
| <b>YOLO</b>       | You Only Look Once   |
| <b>BLE</b>        | Bluetooth Low Energy   |
| <b>MAC</b>        | Media Access Control   |
| <b>RSSI</b>       | Received Signal Strength Indicator                             |
| <b>JSON</b>       | JavaScript Object Notation                                     |
| <b>CSV</b>        | Comma-Separated Values   |
| <b>SDLC</b>       | Software Development Life Cycle                                |
| <b>NFR</b>        | Non-functional requirements                                    |
| <b>JWT</b>        | JSON Web Token   |
| <b>SSE</b>        | Server-Sent Events   |
| <b>CRUD</b>       | Create, Read, Update, Delete                                   |
| <b>UI</b>         | User Interface   |
| <b>OWASP</b>      | Open Worldwide Application Security Project                    |
| <b>HTTPS</b>      | HyperText Transfer Protocol Secure                             |
| <b>TLS</b>        | Transport Layer Security                                       |
| <b>BCrypt</b>     | Blowfish-based password hashing function                       |
| <b>TTL</b>        | Time To Live   |
| <b>EDA</b>        | Event-Driven Architecture                                      |
| <b>Git</b>        | Global Information Tracker                                     |
| <b>PostgreSQL</b> | Postgre Structured Query Language                              |
| <b>SQL</b>        | Structured Query Language                                      |
| <b>IDE</b>        | Integrated Development Environment                             |

# General Introduction

In today’s research landscape, leveraging data to interpret, understand, and predict human interactions within physical spaces and their environments is essential. Whether analyzing visitor engagement in the Smart Museum at the University of Aizu (UoA) or monitoring classroom dynamics on the Higher Institute of Information and Communications Technologie (ISTIC) campus, researchers require flexible tools that unify real-time streams, historical records, and custom datasets. To meet this need, the *Thakia Daigaku* collaboration between ISTIC and Aizu University has launched the development of the Smart Analysis and Research Assistant (SARA), a modular framework designed to support data-driven investigations across geographically distributed sites.

Despite the proliferation of commercial analytics platforms, existing solutions often prioritize generic business intelligence use cases and rely on monolithic architectures. They fail to provide the configurability, extensibility, and low-latency performance required by academic researchers studying dynamic, sensor-driven environments. Moreover, current workflows remain fragmented: live sensor streams are disconnected from historical trend analysis, and custom datasets cannot be easily integrated into the same analytical pipeline. This fragmentation hinders reproducibility, collaboration, and the rapid incorporation of new research scenarios.

The primary objective of this project is to design and implement SARA, a domain-agnostic, microservice-based platform that:

1. *Unifies heterogeneous data sources*, including live sensor feeds and researcher-provided datasets, through a common Application Programming Interface (API) layer.
2. *Supports real-time and historical workflows* with low-latency responsiveness for live dashboards and scalable aggregation for time-series analysis.
3. *Enables extensibility and modularity*, allowing new analytical modules or data sources to be plugged in without modifying existing components.
4. *Provides user-centric interfaces* for authentication, live monitoring, historical visualization, dataset management, and administrative configuration.
5. *Ensures security, scalability, and maintainability* by leveraging modern frameworks (FastAPI, Spring Boot, Angular) and in-memory caching (Redis).

To achieve these objectives, the report is organized as follows:

- **Chapter 1: Project Context.** Introduces the collaborative partnership, presents the smart campus and museum use cases, surveys existing analytics platforms, and justifies the need for a modular research-centric solution.

- **Chapter 2: Specification of Requirements.** Formalizes functional requirements using the Obtain, Scrub, Explore, Model, and eNterpret (OSEMN) framework and non-functional requirements for quality attributes; identifies system actors and presents the overall use case diagram.
- **Chapter 3: Conceptual Study.** Details the platform’s component architecture, interfaces, and sequence diagrams for authentication, data visualization, custom dataset workflows, and administrative operations.
- **Chapter 4: Implementation.** Describes the software environment, core technologies, and the concrete realization of each user-facing interface and microservice, demonstrating how conceptual models translate into a working system.

By following this structure, the report provides a coherent narrative from problem identification through design and implementation, culminating in a flexible framework that empowers academic researchers to harness both real-time and historical data for smarter, data-driven insights.

# Chapter 1

## Project Context

### 1.1 Introduction

This chapter outlines the context, objectives, and methodology of SARA platform. Section 1.2 introduces the host institutions, ISTIC and the University of Aizu, and their collaborative vision for smart campus research. Section 1.3 defines the project's goals. Sections 1.4 and 1.5 evaluate prominent analytics platforms against research needs and critiques their limitations in scalability and real-time analysis. Section 1.6 proposes SARA's modular architecture. Finally, Section 1.7 details the technical approach, including Unified Modeling Language (UML) diagrams, to guide the system's design.

### 1.2 Presentation of the Host Organization

This project is carried out through the collaboration known as **Thakia Daigaku** between The Higher Institute of Information and Communications Technologies (ISTIC) and the University of Aizu (UoA) in Japan, with the aim of advancing smart campus technologies.

#### 1.2.1 The Higher Institute of Information and Communications Technologies (ISTIC)

The **Higher Institute of Information and Communications Technologies (ISTIC)** [20] is a leading Tunisian public institution established in 2011 under the administrative supervision of the University of Carthage. ISTIC offers high-quality training through the Bachelor, Master, Doctorate (LMD) system, with a capacity to accommodate over 800 students across its programs.

ISTIC differentiates itself through a multidisciplinary curriculum that bridges theoretical knowledge and practical skills in computer science, electronics, telecommunications, and automation. Its academic structure is organized into three departments:

- **Industrial Information Technology**
- **Computer Science and Mathematics**
- **Telecommunication**

ISTIC's research activities are closely tied to its academic mission, focusing on areas like automation, cognitive science, web technologies, and mobile systems. The institute collaborates

with national and international partners, including **Huawei, University of AIZU, and ARM University**, to promote knowledge exchange and technological innovation.

Through initiatives and partnerships with industry leaders, ISTIC cultivates an ecosystem that bridges academia and the private sector, driving advancements in Tunisia’s Information and Communication Technology (ICT) landscape [20].

### 1.2.2 The University of Aizu

The University of Aizu, established in 1993 in Fukushima Prefecture, Japan, is a globally renowned institution dedicated exclusively to cutting-edge education and research in computer science, engineering, and advanced technologies. As Japan’s first university focused solely on computer science, it has carved a niche in fostering innovation through its motto, “Advancing Knowledge for Humanity” [40]. The university is structured around interdisciplinary collaboration, offering undergraduate, master’s, and doctoral programs that emphasize practical problem-solving, creativity, and global competitiveness.

The University of Aizu’s curriculum and research are organized around three core pillars:

- **Computer Science and Engineering:** Specializing in Artificial Intelligence (AI), robotics, cybersecurity, cloud computing, and human-computer interaction.
- **Information Technology Applications:** Bridging theory and practice in areas like Internet of Things (IoT), big data analytics, and embedded systems.
- **Advanced Systems Design:** Innovating in robotics, autonomous systems, and smart infrastructure.

The university is distinguished by its world-class laboratories, including the Robotics Laboratory, Cybersecurity Research Center, and AI Cloud Research Center, which facilitate groundbreaking research in collaboration with industry leaders like Nippon Electric Company (NEC), Hitachi, and Fujitsu.

## 1.3 Project Overview and Objectives

Building on this partnership, the project aims to enhance data-driven research in both institutions.

### 1.3.1 Motivation and Problem Statement

Modern academic institutions increasingly deploy sensor-driven systems to advance empirical research across diverse environments.

At the **University of Aizu**, this is exemplified by two complementary initiatives:

- **Smart Campus:** An ecosystem instrumenting classrooms and the cafeteria to study human-environment interactions in academic spaces.
- **Smart Museum [43]:** A research initiative at the Fukushima Museum, aiming to objectively measure visitor behavior and improve learning experiences through IoT systems. Its mission focuses on:
  - Automated measurement of exhibit engagement (e.g., dwell time, movement patterns)

- Development of non-intrusive tools to support curators in exhibit design and evaluation
- Computational analysis of visitor data to correlate behavior with knowledge acquisition

Similarly, **ISTIC**'s pilot *Smart Classroom* employs embedded sensors to analyze micro-level environmental impacts on student behavior and classroom dynamics.

Although these projects vary in scope researchers share similar challenges:

- **Disjointed Workflows:** No unified platform integrates *real-time monitoring* with *historical trend analysis*.
- **Lack of Research-Centric Customization:** Commercial platforms prioritize business intelligence, lacking configurability for niche academic use cases.

### 1.3.2 Project Goals

This project addresses these gaps by developing a **domain-agnostic** and **modular** smart research assistant **framework** tailored to:

- Unify heterogeneous data streams (APIs, static files) into a single analytical workflow
- Provide customizable modules for cross-disciplinary studies, such as:
  - Time-series analysis of visitor (Museum) and student (Campus) behaviors.
  - Real-time classroom monitoring.
- Enable reproducible experiments through integrated visualization tools.
- Enable academic institutions to avoid vendor-locked solutions.

## 1.4 Existing Systems

In recent years, the proliferation of sensor technology has led to the emergence of numerous **Unified data platforms** designed to process, visualize, and extract insights from large volumes of data. Such systems follow a common structure known as the OSEMN pipeline (Obtain, Scrub, Explore, Model, iNterpret), which breaks down the main steps involved in working with data. This section begins by outlining the stages of the OSEMN pipeline, followed by an overview of existing systems most aligned with our goals.

### 1.4.1 The OSEMN Pipeline Framework

For our purposes, we define Data Analytics platforms as systems that integrate various tools to deliver comprehensive, contextually analyzed data, covering all stages of the **OSEMN pipeline**:



Figure 1.1: The OSEMN Pipeline: A framework for data science workflows, including Obtain, Scrub, Explore, Model, and iNterpret.

As shown in Figure 1.1, the OSEMN pipeline provides a structured approach to data science workflows and includes the following steps [27]:

- **Obtaining Data:**  
Identify all available data sources and extract the data into a usable format. This step involves gathering raw data from databases, APIs, files, or other repositories, ensuring it is accessible for further processing.
- **Scrubbing / Cleaning Data:**  
Data cleaning is highly application-dependent and should be customizable to address specific needs. This stage involves transforming data into a clean, standardized format suitable for analysis.
- **Exploring / Visualizing Data:**  
Explore the data to uncover patterns, trends, and relationships. Use statistical summaries and visualization tools (e.g., histograms, scatter plots, heatmaps) to gain insights and inform subsequent modeling decisions.
- **Modeling the Data:**  
Develop data analysis scripts and build predictive models using machine learning or statistical techniques.
- **Interpreting Data:**  
Analyze the results of the models and translate them into actionable insights. Then, communicate the findings effectively to stakeholders, ensuring that the conclusions are clear, meaningful and aligned with the business objectives.

### 1.4.2 Overview of Prominent Existing Systems

There are numerous data analytics platforms available, but in this section we focus exclusively on the most prominent and feature-rich solutions. Our selection aligns with the industry benchmark “*Gartner Magic Quadrant for Analytics and Business Intelligence Platforms 2024*” [42].





Figure 1.2: Gartner Magic Quadrant for Analytics and Business Intelligence Platforms, 2024 [42].

As illustrated in the quadrant in Figure 1.2, the leading ("Leaders") platforms include:

- **Microsoft Power BI** [34]

Microsoft Power BI has established itself as a dominant force in the business intelligence sector, driven by three core strengths: seamless integration within the Microsoft ecosystem, competitive pricing, and continuous innovation. Over the past four years, adoption has accelerated significantly, with Gartner [42] highlighting Microsoft's strong combination of *ecosystem cohesion*, *functionality*, and *price* as key factors behind its leadership in the analytics market.

- **Salesforce (Tableau)** [37]

Tableau excels in intuitive visual analytics and cross-platform flexibility, supporting over 100 data connectors for hybrid environments. Its drag-and-drop interface and Visual Query Language (VizQL) engine enable rapid creation of complex visualizations without coding, making it ideal for the *Explore* and *iNterpret* stages of the OSEMN pipeline as illustrated in Figure 1.1.

- **Qlik [36]**

Qlik distinguishes itself through its associative analytics engine, enabling non-linear exploration of data relationships. Its in-memory processing and robust data integration tools (e.g., Qlik Data Integration) streamline the *Obtain* and *Scrub* stages of the OSEMN pipeline, as shown in Figure 1.1, while its self-service dashboards empower users in the *Explore* and *iNterpret* phases.

- **Oracle, ThoughtSpot, and Google Looker**

Oracle Analytics [35], ThoughtSpot [38], and Google Looker [33], all cater to distinct enterprise needs. Oracle Analytics excels in enterprise-grade AI/Machine Learning (ML) integration and hybrid cloud support, streamlining the *Model* and *iNterpret* stages of the OSEMN pipeline through automated insights. ThoughtSpot prioritizes search-driven analytics, enabling natural language queries for rapid *Explore* workflows. Google Looker (now part of Google Cloud) emphasizes embedded analytics and unified semantic modeling (LookML), enhancing *Obtain* and *Scrub* efficiency via BigQuery integration. **While Oracle and Google target large-scale enterprises with end-to-end ecosystems, ThoughtSpot appeals to organizations prioritizing agility and user-driven discovery.**

## 1.5 Critical Assessment of Existing Systems

Existing analytics platforms, while powerful, prioritize generic Business Intelligence (BI) use cases, creating gaps for **specialized domains like smart campus monitoring**. This section provides a critical assessment of most prominent solutions.

### 1.5.1 Definition of Assessment Criteria

The selected criteria are drawn from industry benchmarks such as Gartner’s Magic Quadrant for Analytics and Business. They are defined as follows:

- **1. Real-Time & Historical Data Analysis Support:**

This criterion measures the platform’s ability to process, analyze, and present data both in real time and over historical periods. Real-time responsiveness is assessed based on the latency between data ingestion and the update of visualizations. Historical data analysis is evaluated by the platform’s capacity to handle large volumes of archived data and support trend analysis over extended time frames.

- **2. Custom Analytics & ML Flexibility:**

This criterion examines the extent to which a platform allows users to integrate custom analytics and machine learning models into the data processing pipeline. It assesses whether the platform can be tailored to perform advanced, domain-specific data transformations, and whether it supports the development and deployment of custom predictive models.

- **3. Cost & Scalability:**

This criterion evaluates both the economic viability and the technical scalability of the platform. It considers the total cost of ownership, including licensing fees and operational expenses, as well as the ability of the system to scale efficiently with increasing data volumes and user demand. In addition, it assesses the risk of vendor lock-in, which can limit long-term adaptability.

### 1.5.2 Assessment of Existing systems

To better understand the strengths and limitations of current analytics solutions, Table 1.1 provides a comparative assessment of several leading platforms based on the defined evaluation criteria.

Table 1.1: Critical Assessment of Data Analytics Platforms

| Solution                  | Real-Time & Historical Data Analysis Support   | Custom Analytics & ML Flexibility  | Cost & Scalability  |
|---------------------------|--|--|---|
| Oracle [8]                | Traditional batch processing with strong historical support; real-time responsiveness is limited to several minutes.                           | Proprietary tools restrict customization; integrating advanced ML requires significant adaptation.                       | Enterprise-grade scalability with strong vendor lock-in; Professional plan starts at \$16/user/month [24].                      |
| ThoughtSpot [11]          | Provides real-time search-driven analytics with 100–300 ms latency; historical support is adequate.  | Moderate flexibility; well-suited for predefined queries but less ideal for bespoke ML pipelines.                        | Scalable architecture with some dependency on vendor solutions; Essentials plan starts at \$1 250 / month [41].                 |
| Google Looker [6]         | Leverages cloud infrastructure for both real-time and historical analysis, though responsiveness depends on the underlying data warehouse.     | Highly flexible through LookML, facilitating custom data modeling and integration with AI services.                      | Strong scalability tied to the Google ecosystem; Standard plan starts at \$5 000/month [3].                                     |
| Qlik [9]                  | Provides data exploration with in-memory engine; updates typically within 1 minute; historical analysis requires additional aggregation steps. | Good support for self-service analytics; however, deeper custom ML integration demands technical expertise.              | Robust performance for large datasets; Qlik Sense Business starts at \$30/user/month [29].                                      |
| Microsoft Power BI [7]    | Scheduled real-time updates with strong historical data support within the Microsoft ecosystem.  | Limited flexibility for advanced, bespoke ML analytics without relying on additional Azure services.                     | Good scalability within its integrated suite; Pro license starts at \$10/user/month [21].                                       |
| Salesforce (Tableau) [10] | High-quality visualization and historical data analysis; real-time dashboard updates typically within 5 seconds.                               | Excellent for interactive data exploration, yet custom ML functionalities are less accessible without extra integration. | Considerable risk of vendor lock-in, although the interface is highly polished; Creator license starts at \$75/user/month [32]. |

In summary, while the platforms assessed above excel in general BI use cases, none are specifically tailored to the dual needs of researchers, **combining collaborative historical analysis with real-time monitoring across different sites**. This gap motivates our proposed SARA framework, detailed in the next section, which unifies these capabilities in a single, researcher-centric solution.

## 1.6 Proposed Solution

The proposed framework employs a modular architecture to address the varied research requirements of ISTIC and the University of Aizu. At its core, SARA unifies the following capabilities:

### 1. Historical Data Visualization:

Historical analysis enables researchers to identify long-term trends by correlating data across time, locations, and sensor types. This capability is powered by two complementary data sources:

#### (a) **Sensor Network:**

Researchers have access to data from IoT sensor networks deployed across three facilities:

- **University of AIZU:** Classrooms, and cafeteria.
- **ISTIC:** Smart Classroom A006.
- **Fukushima Museum:** Exhibit halls and visitor areas.

### **Sensor Configuration**

Each facility uses standardized sensor modules—such as the one depicted in Figure 1.3—configured as follows:

- Four modules per room, each identified by an `agent_serial`, positioned at corners as illustrated in Figure 1.4.
- Each module collects several types of information, as detailed in Table 1.2:
  - Environmental sensors: Temperature (°C), humidity (%), luminance (lux), noise (dB).
  - Motion detectors and thermographs (24×32 grid thermal maps).
  - Cameras with You Only Look Once (YOLO) v8-based person detection.

Data is recorded at regular intervals and structured in monthly tables.

### **API Integration**

SARA interacts with data through an external API provided by UoA. To consume the API, the client must first obtain an access token from the authentication endpoint. This token is valid for 5 minutes. Once acquired, data is fetched from the data endpoint by specifying the following path parameters:

- **facility:** one of `istic`, `aizu`, `museum`

Table 1.2: Description of sensor types within each sensor module

| Sensor      | Data Description  |
|-------------|---|
| Temperature | Ambient temperature (°C)  |
| Humidity    | Air humidity (%)  |
| Luminance   | Light intensity (lux)   |
| Noise       | Sound pressure (dB)   |
| Presence    | Bounding box coordinates, person count (YOLOv8)   |
| Motion      | Occupancy probability   |
| Thermalmap  | 24 × 32 temperature grid  |
| Radio       | <ul style="list-style-type: none"> <li>• <i>Device count</i>: Number of detected Bluetooth Low Energy (BLE) devices</li> <li>• <i>Per-device records</i>: Temporal Media Access Control (MAC) address (rotated for privacy), Received Signal Strength Indicator (RSSI) (dBm)</li> </ul> |



Figure 1.3: One of the four sensor modules installed at ISTIC: Classroom A006

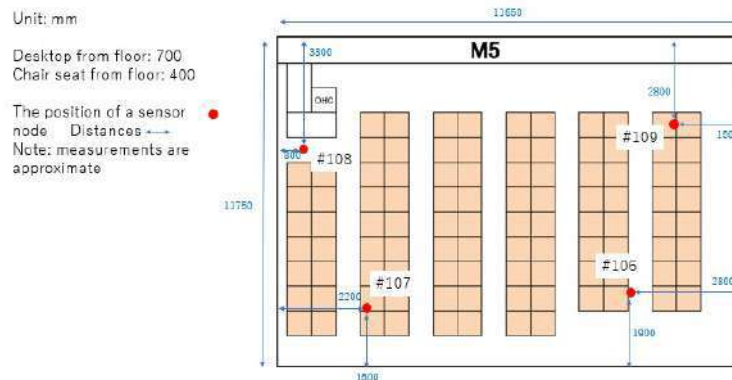


Figure 1.4: Sensor Modules Placement Illustration in Classroom M5

- **sensor\_type**: sensor type as defined in Table 1.2
- **agent\_serial**: unique identifier of the sensor module
- **start\_date, end\_date**: date range in YY-MM-DD format
- **page\_size**: number of records per page
- **page**: page number to retrieve

The full request path is: `/api/{facility}/{sensor_type}/range/{agent_serial}/{start_date}/{end_date}/{page_size}/{page}` .

A valid access token must be included in the **Authorization** header of each request.

### (b) Custom Datasets

In addition to pre-collected sensor data, researchers can upload their own datasets for collaborative analysis. Through the SARA application, users submit data files in JavaScript Object Notation (JSON) or Comma-Separated Values (CSV) format and an accompanying JSON Schema. SARA validates each upload against the provided schema, then prompts the user to define visibility:

- **Private**: visible only to the uploader.
- **Public**: visible to all SARA users.
- **Restricted**: visible only to a specified list of collaborators (entered as email addresses).

Once validated and saved in the central database, any user with access will see their custom dataset listed alongside historical sensor data in the “Visualize Historical Data” interfaces, and may explore and visualize it just like the sensor data feeds.

After defining the two data origins—pre-collected sensor feeds and researcher-provided datasets—SARA’s Visualize Historical Data module provides rich exploration features. These features enable researchers to analyze temporal and spatial patterns through:

- **Temporal Hierarchies**: Drill down or roll up data across resolutions:
  - *Time*: Minute → Hourly → Daily → Monthly → Yearly
  - *Location*: Facility (e.g, ISTIC) → Location (e.g, Classroom A006) → Sensor Box → Sensor Type (e.g., temperature, humidity)
- **Multi-Sensor Correlation**: Analyze and compare trends across sensors of the same type or different types. For example, correlate classroom noise levels with occupancy rates or with the thermograph readings.
- **Metric Flexibility**: Apply statistical operators (mean, max,min, skewness, median) and aggregate data dynamically (e.g., hourly averages, monthly peaks).

## 2. Real-Time Monitoring

Building on the historical and custom dataset views, SARA also delivers live dashboards that stream sensor data from all facilities as it is generated. Key features include:

- **Live Sensor Streams**: Monitor parameters such as temperature, noise, occupancy, and luminance in real time.

- **Dynamic Controls:** Adjust the update frequency (default 1 minute), apply statistical metrics on the fly (mean, sum, max, min, median), or pause and resume individual streams.
- **Multi-Site Support:** Seamlessly switch between facilities—University of AIZU, ISTIC, Fukushima Museum—and focus on any sensor type or module.

## 1.7 UML Language

Unified Modeling Language (UML) provides a standardized set of diagrams to model different aspects of software systems. Some of the most common UML diagrams include:

**Use Case Diagram, Class Diagram, Component Diagram, Sequence Diagram, Activity Diagram, State Diagram**

For this project, we have chosen to use the following diagrams:

- **Use Case Diagram:** To clearly capture and communicate the functional requirements and interactions between the system and its users.
- **Class Diagram:** To define the system’s data structures, business logic, and relationships between key entities by detailing their attributes, methods, and the relationships between them.
- **Component Diagram:** To represent the modular architecture of the proposed platform, emphasizing how microservices and different components interact within our solution.
- **Sequence Diagram:** To illustrate the dynamic behavior of the system by modeling the sequence of interactions between objects or services in response to specific use cases.
- **Activity Diagram:** To model the internal workflows and business processes, depicting the flow of control between activities and decisions within the system.

## 1.8 Conclusion

This chapter contextualizes SARA’s development within ISTIC and UoA’s partnership (Section 1.2), identifies gaps in commercial analytics tools (Sections 1.4–1.5), and proposes a modular framework to unify real-time and historical data workflows (Section 1.6). By leveraging UML for system design (Section 1.7), the chapter establishes SARA as a scalable, research-focused solution. These foundations set the stage for Chapter 2, which will formalize SARA’s functional and non-functional requirements, aligning them with researchers needs.

# Chapter 2

## Specification of Requirements

### 2.1 Introduction

In this chapter, we translate the project goals and use-case narratives into precise system requirements. Section 2.2 articulates the *functional requirements*—what the platform must do—using the OSEMN framework (Obtain, Scrub, Explore, Model, iNterpret) to mirror the data lifecycle. Section 2.3 defines the *non-functional requirements*, specifying the quality attributes needed for a robust and user-friendly system. We then identify the key *actors* and their roles in Section 2.4, clarifying the boundaries of interaction, and present the overall system interactions in a *use case diagram* (Section 2.5). Together, these specifications provide the foundation for the conceptual designs developed in Chapter 3.

### 2.2 Functional Requirements

Functional requirements define the **specific operations, capabilities, and behaviors** a system must exhibit to fulfill its intended purpose. They articulate *what the system must do* in terms of inputs, processes, outputs, and user interactions [13]. In our case these requirements are derived from researchers’ needs, use cases, and domain-agnostic objectives. We choose to categorize functional requirements using the **OSEMN framework** (Obtain, Scrub, Explore, Model, iNterpret) to ensure alignment with the system’s data lifecycle.

The system shall satisfy the following functional requirements:

#### 2.2.1 Obtain

- The system shall consume real-time and historical data from an external API.
- The system shall obtain data from researcher provided datasets.

#### 2.2.2 Scrub

- The system shall convert obtained data into a format optimized for analysis.



### 2.2.3 Explore

- The system shall allow researchers to monitor real-time data streams.
- The system shall enable researchers to visualize historical data.
- The system should allow offline and real-time statistical analysis of data.
- The system shall allow the user to save and comment on their experiments.

### 2.2.4 Model

- The system shall allow the researcher to analyze the impact of environmental factors on student behavior.

### 2.2.5 iNterpret

- The system shall allow researchers to take snapshots of, annotate, and export analysis results.

## 2.3 Non-Functional Requirements

Non-functional requirements (NFRs) define **how well the system performs its functions**, focusing on quality attributes, constraints, and operational capabilities. These requirements ensure the system meets researcher expectations for efficiency, safety, and user satisfaction [13].

The system shall satisfy the following non functional requirements:

- **Scalability:** The application shall be designed to support an increasing number of sensors and data sources.
- **Performance:** The system shall process and visualize large datasets in real time with minimal latency.
- **Security:** The application shall implement authentication and authorization mechanisms to protect user data.
- **Usability:** The web interfaces shall be user-friendly, providing an intuitive experience for researchers.

## 2.4 Actors and their Roles

System actors are entities (human or external systems) that interact with the platform to achieve specific goals. By distinguishing actors we define the **boundaries of the system** and clarify responsibilities, ensuring alignment between user needs and technical implementation [12].

The system interacts with the following key actors:

Table 2.1: Actors and Roles in the Smart Research Assistant Platform

| Actor                | Role   |
|----------------------|--|
| Researcher           | Visualizes historical sensor-network data and custom datasets; uploads, validates, and manages their own datasets; monitors real-time data streams; comments on and saves experimental analyses. |
| System Administrator | Manages reference data.  |
| External API         | Supplies real-time and historical sensor-network data streams to the platform.   |

## 2.5 Use Case Diagram

Figure 2.1 depicts the high-level use case diagram for the SARA platform, illustrating the interactions between primary actors (Researchers, System Administrator, External API) and the core system functionalities.

In the following sub-sections, each use case is explained using a structured template that includes: the actors involved, preconditions, the main (normal) flow, postconditions, and any alternative or exception flows.

### a. Use Case: Sign Up

**Actor:** Researcher

**Preconditions:**

1. The researcher has no existing SARA account.
2. The system is available to accept new registrations.

**Main Flow:**

1. The researcher clicks “Sign Up”.
2. The system presents a registration form requesting username, email, role (Researcher or Admin), password, and password confirmation.
3. The researcher completes and submits the form.
4. The system validates:
  - Email uniqueness.
  - Password length (more than 6 characters) and confirmation match.

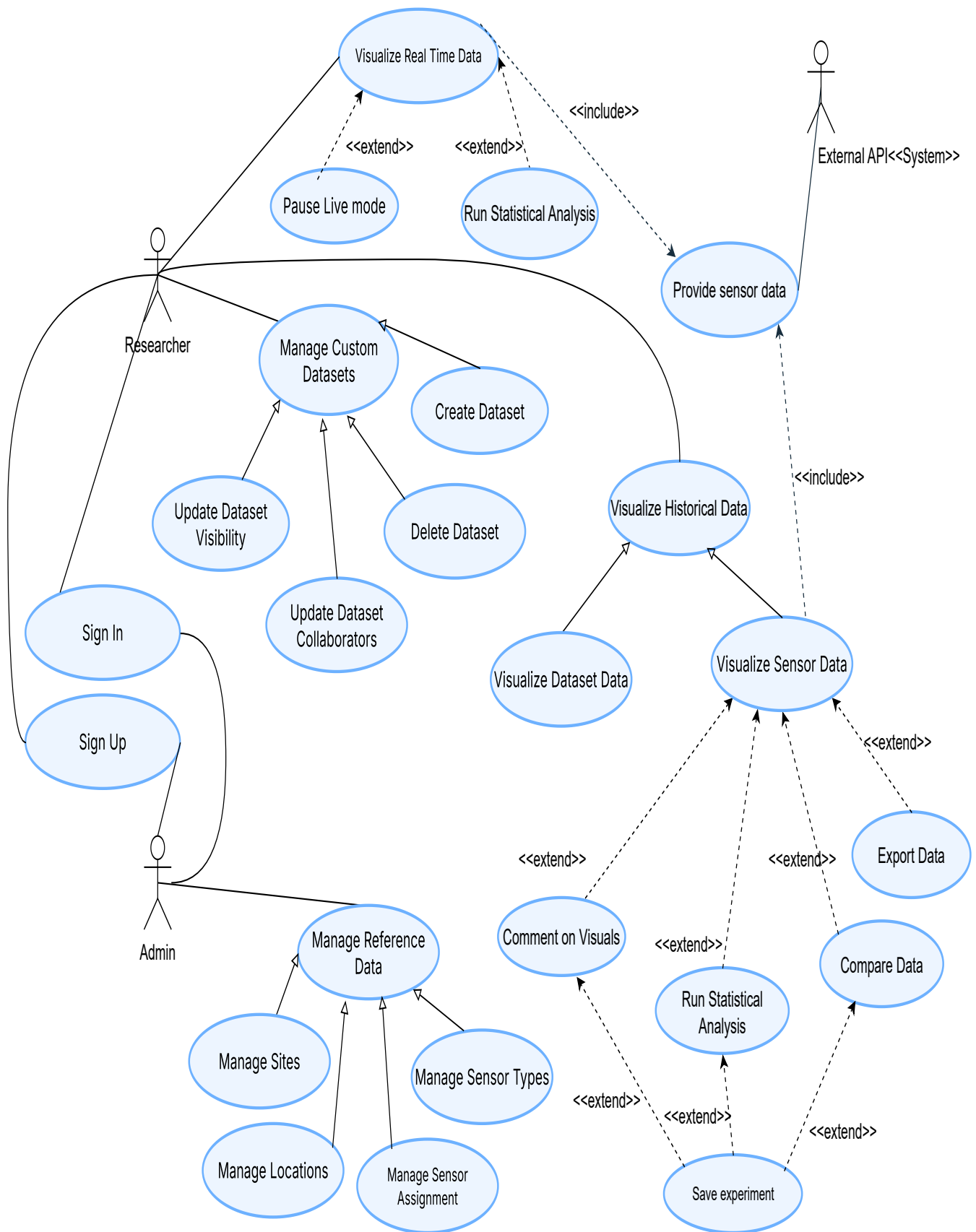


Figure 2.1: Use Case Diagram of the SARA Platform

5. On success, the system creates a new **SaraUser** record.
6. The researcher is redirected to the SignIn tab.

**Postconditions:** A new user account exists in the database.

**Alternative Flows:**

1. *Email already in use:* The system displays an error and prompts for a different email.
2. *Password mismatch or too short:* An error indicates password requirements; the form remains for correction.

**b. Use Case: Sign In**

**Actor:** Researcher

**Preconditions:**

1. The researcher has a registered account.
2. The system's authentication service is operational.

**Main Flow:**

1. The researcher enters email and password on the login screen.
2. The system verifies credentials against the **SaraUser** table.
3. If valid, the system issues a JSON Web Token (JWT) valid for 15 minutes, with a refresh token in place to maintain session continuity.
4. Both tokens are returned to the frontend for storage.
5. The researcher is redirected to:
  - The Admin Dashboard, if role = Admin.
  - The Real-Time Monitoring interface, if role = Researcher.

**Postconditions:**

1. The researcher has valid tokens stored client-side.
2. The session is established with appropriate role-based access.

**Alternative Flows:** *Invalid credentials:* An error is displayed; the researcher may retry.

**c. Use Case: Visualize Real-Time Data**

**Actor:** Researcher

**Preconditions:**

1. Researcher is signed in and authorized (has the appropriate role to access sensor data).
2. Sensors emitting data every minute.
3. At least one sensor stream is active.
4. SSE endpoint of the Real-Time Monitoring Service is available.

**Main Flow:**

1. The researcher navigates to the Real-Time Monitoring dashboard.
2. The frontend subscribes to SSE channels for all sensors.
3. New data events arrive via SSE and update the corresponding visual cards.
4. The researcher may adjust controls: update frequency, metric, pause/resume, and chart type.

**Postconditions:**

1. Live sensor data is continuously displayed.
2. Researcher's control preferences are applied to the stream.

**Alternative Flows:** *No active streams or no data available:* Display placeholder indicating no data.

**d. Use Case: Visualize Historical Data**

**Actor:** Researcher

**Preconditions:**

1. Researcher is authenticated.
2. Historical Processing Service and cache are available.
3. Requested data range and resolution are valid.
4. Data available within the requested range.
5. Researcher owns or has collaborative access to the requested dataset.

**Main Flow:**

1. The researcher selects the Historical Visualization view.
2. User drags data source elements onto Y-axis and defines X-axis parameters.
3. The system retrieves aggregates from Redis or computes them via the microservice.

4. The chart is rendered, supporting trend analysis, comparisons, and annotations.

**Postconditions:**

1. Historical data is displayed at the chosen resolution.
2. Computed aggregates are cached for future requests.

**Alternative Flows:**

1. *Invalid date range:* Display validation error and request correction.
2. *Data unavailable within the requested range and for the requested sensor in the requested location:* Display alert indicating no data and prompt the researcher to change the selected parameters.

**e. Use Case: Create Dataset**

**Actor:** Researcher

**Preconditions:**

1. Researcher is authenticated.
2. Dataset wizard and database are accessible.

**Main Flow :**

1. The researcher launches the Dataset Creation Wizard.
2. Steps:
  - Upload or paste data; select visibility and collaborators.
  - Define and validate JSON schema against the data.
  - Preview data; click *Create*.
3. The system stores the new **Dataset** and associated **DatasetCollaborator** records.

**Postconditions:** The new dataset appears in the Historical Visualization source list.

**Alternative Flows:** *Schema mismatch:* Display errors; prevent progression until resolved.

*Note: The "update dataset visibility", "update dataset collaborators", and "delete dataset" use cases, follow a similar flow.*

### 2.5.1 External api Use Case: Provide Data

This use case describes how an external API interacts with the system to authenticate and provide sensor data to authorized users.

**Actor:** External API

**Main Flow:**

1. Validates researcher's credentials via Keycloak.
2. Issues or renews access tokens.
3. Responds to data requests with sensor readings according to path parameters.

### 2.5.2 Administrator Use Case: Manage Reference Data

This use case describes how the administrator manages and maintains reference data to ensure accurate configuration and organization of system entities.

**Actor:** System Administrator

**Preconditions:** Admin is authenticated and authorized.

**Main Flow:**

1. Admin selects a reference entity (Site, Location, SensorBox, SensorType).
2. Performs Create, Read, Update, Delete (CRUD) operations via the admin interface.
3. System updates database.

**Postconditions:** Updated reference data is immediately available to all dependent services.

### 2.5.3 Administrator Use Case: Manage Reference Data

This use case describes how the administrator manages and maintains reference data to ensure accurate configuration and organization of system entities.

**Actor:** System Administrator

**Preconditions:** Admin is authenticated and authorized.

**Main Flow:**

1. Admin selects a reference entity (Site, Location, SensorBox, SensorType).
2. Performs CRUD operations via the admin interface.
3. System updates database.

**Postconditions:** Updated reference data is immediately available to all dependent services.

## 2.6 Conclusion

This chapter has established a clear and comprehensive specification of the platform's requirements. By defining functional requirements through the OSEMN framework, we ensured that all data acquisition, processing, and visualization needs are systematically addressed. The non-functional requirements articulate the essential quality attributes—scalability, performance, security, usability, and availability—necessary for a reliable research tool. Identifying actors and mapping their interactions in the use case diagram completes the requirements analysis, setting the stage for the conceptual designs presented in Chapter 3.

# Chapter 3

## Conceptual Study

### 3.1 Introduction

This chapter presents the conceptual design of the SARA platform, focusing on the interactions between its core components and the logic behind its key functionalities. The goal is to clarify how the system supports real-time monitoring, historical data analysis, custom dataset integration, and administrative management through a modular architecture. Section 3.1 introduces the system’s components and their interactions (see Figure 3.1), while subsequent sections detail the workflows and sequence diagrams that govern user authentication (Section 3.2), data visualization (Sections 3.3 and 3.4), dataset management (Section 3.5), and administrative operations (Section 3.6). Each section aims to illustrate not only the internal mechanics of the platform but also its extensibility and scalability.

### 3.2 Component Overview

The component diagram in Figure 3.1 illustrates SARA’s **modular architecture**, emphasizing its ability to unify real-time and historical data workflows while **maintaining extensibility for future use cases**. The system is decomposed into seven collaborating components—*SARA User Interface (UI)*, *The SpringBoot gateway*, *the real-time monitoring and the historical processing microservices*, *the Redis cache*, and *the SARA database*—, each fulfilling a distinct responsibility and interacting via well-defined interfaces.

#### a. SARA UI

The SARA UI is the only user-facing component of the system. It issues requests to the Spring Boot Gateway, streams real-time data from the Real-Time Monitoring Service, and renders various visualizations. It presents three historical data visualizations: one for general trends and two for comparing sensor data. In addition, it provides dashboards for real-time monitoring, interfaces for managing custom datasets, and dedicated administrative interfaces for managing reference data.

#### b. Spring Boot Gateway

Serving as the single entry point, the Spring Boot Gateway performs the following roles:

1. **Authentication and Authorization:** Validates incoming requests and enforces role-based access control (e.g., only administrators may modify reference data).



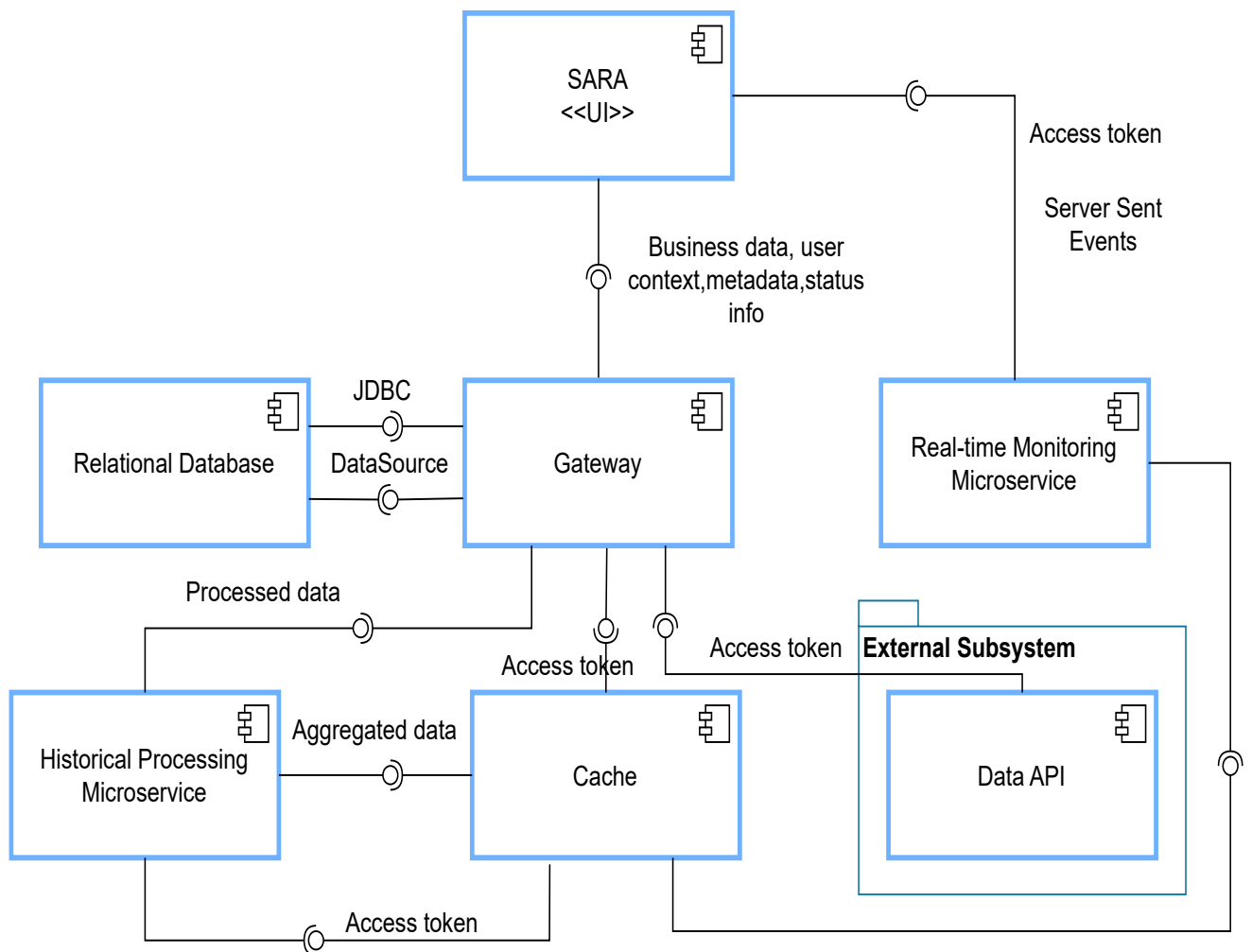


Figure 3.1: Component Diagram of the SARA platform

2. **Token Management:** Periodically (every 5 minutes) fetches an access token from the External API and stores it in the Redis Cache for downstream microservices to reuse to consume the External API.
3. **Request Routing:** Forwards historical queries to the Historical Processing Service and routes CRUD operations to the Application Database.

**c. Historical Processing Service**

This self-contained microservice:

1. Fetches raw historical records from the External SARA.
2. Computes aggregate metrics (e.g. daily success rates).
3. Stores computed aggregates back into Redis for low-latency retrieval.

**d. Real-Time Monitoring Service**

Also fully autonomous, this service:

1. Obtains the same API token from the Cache.
2. Polls the External API for live data streams.
3. Pushes updates directly to the real-time dashboards via Server-Sent Events (SSE).

**e. Redis Cache**

The in-memory Redis Cache serves two central purposes:

1. *Token Store:* Holds the API access token for use by both microservices.
2. *Aggregate Cache:* Stores computed historical aggregates to accelerate repeated queries.

**f. Application Database**

Accessible only through the Spring Boot Gateway, the Application Database persists:

1. User accounts and authentication data
2. Role definitions
3. Reference data and user-defined datasets

**g. External API**

Treated as an external subsystem, the External API provides both historical and live data feeds.

The system's modular architecture is enabled by the strict separation of concerns and the use of explicit interfaces for each component. This design allows new use cases to be integrated without modifying existing services. A new microservice can be added by simply:

- a. Implementing its business logic with its own provided and required interfaces.
- b. Registering its routes in the Spring Boot Gateway.
- c. Configuring any necessary cache or database dependencies.

## 3.3 Authentication Flow: Sign-In Use Case

The sign-in use case is a critical component of the system's authentication flow, enabling users to securely access protected resources. This implementation leverages JWT-based stateless authentication to ensure security, scalability, and compliance with modern security standards such as Open Worldwide Application Security Project (OWASP) Top 10 [25].

### 3.3.1 Sequence Diagram of "Sign-In" use case

The following sequence diagram illustrated in Figure 3.2 outlines the step-by-step flow of the sign-in process.

### 3.3.2 Sign-In Process Description

The authentication workflow follows six principal phases, as illustrated in Figure 3.2.

#### Phase 1: Credential Submission

Users initiate the process by submitting their credentials through a secure web form. The frontend application performs initial validation of input format before transmission.

#### Phase 2: Secure Communication

The Angular client transmits credentials via HyperText Transfer Protocol Secure (HTTPS) POST request to the controller (Spring backend), ensuring encrypted communication through TLS 1.3 protocols.

#### Phase 3: Backend Verification

The Spring security layer coordinates three critical operations:

- Database lookup for user records
- BCrypt hash comparison for password validation
- JWT generation with system-configured expiration

#### Phase 4: Token Management

Upon successful authentication, the system generates paired tokens:

- Short-lived access token (15-minute validity)
- Long-lived refresh token (7-day validity) stored in database

The refresh token is stored in a dedicated database table—separate from the user table—to support multiple simultaneous logins from different devices. When the access token nears expiration, the frontend sends a refresh request with the stored refresh token. The Spring Boot backend then verifies the refresh token's validity against the database. If the refresh token is valid, a new access token is generated; if not, the token is deleted (either upon expiry or user logout).

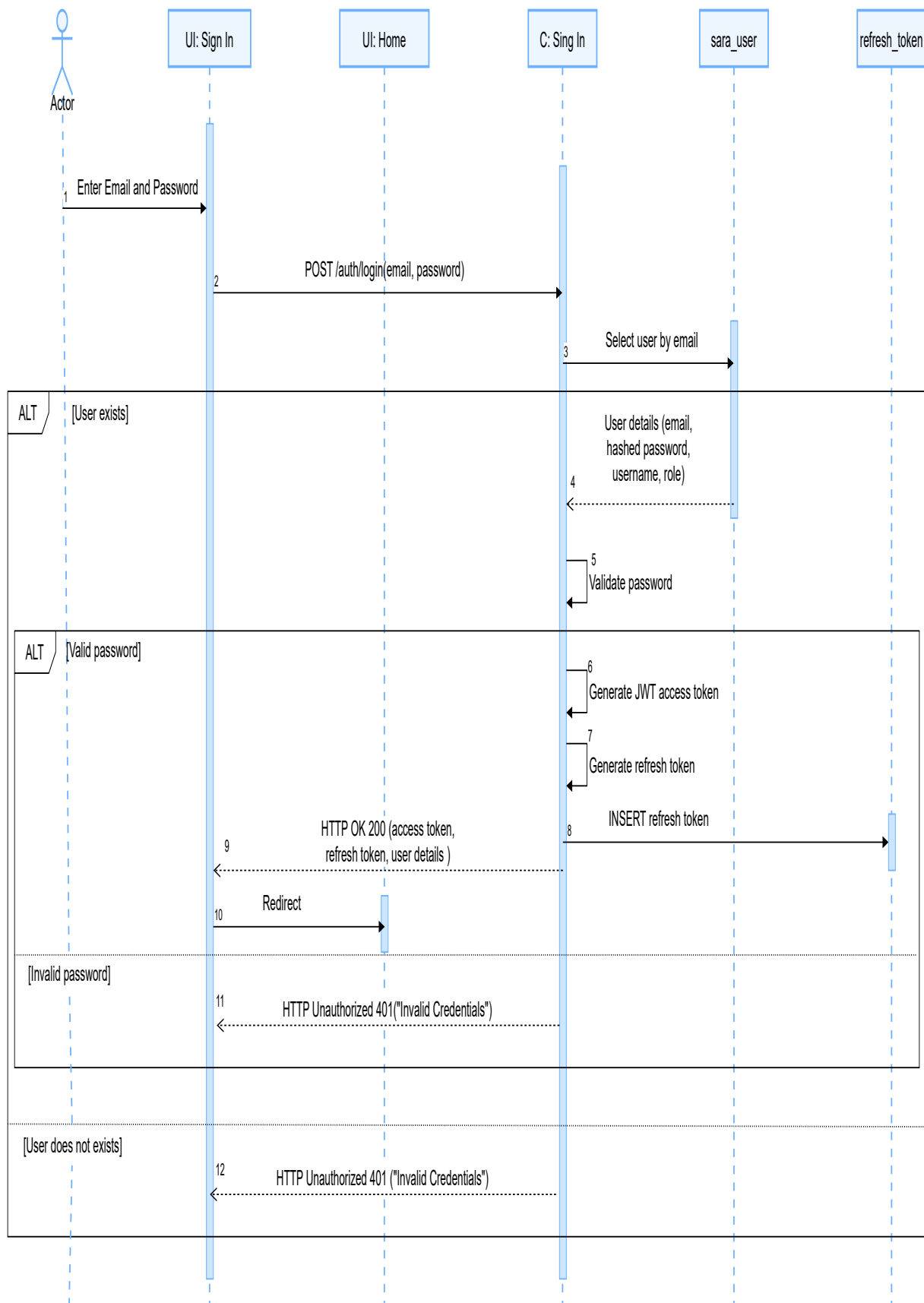


Figure 3.2: Sequence Diagram of the Sign In use case

### Phase 5: Client-Side Storage

The frontend application persists authentication artifacts using browser storage mechanisms specifically:

- The access and refresh tokens are stored in `localStorage`, allowing the application to maintain authentication state across page reloads.

### Phase 6: User Feedback

The system concludes the workflow with either:

- Success: Dashboard (home page) redirection with session initialization
- Failure: Error messaging with credential reset

## 3.4 Visualize Historical Data: Conceptual Design

This section presents the architecture for visualizing historical data from two sources: sensor networks deployed across facilities and user-provided datasets. We first define the data sources, then systematically explain how requests are processed, emphasizing design choices that ensure scalability and interactivity.

Researchers can analyze historical data from:

1. **Custom Datasets:** User-uploaded datasets—conforming to predefined schemas—are stored in the platform’s central database for collaborative analysis.
2. **Sensor Networks:** A network of pre-installed IoT devices at **ISTIC University**, **University of AIZU**, and the **Fukushima Museum** continuously generating environmental and behavioral metrics like temperature and presence. The visualization of data from this origin, requires a dedicated processing microservice, which will be detailed in Section 3.4.1.

### 3.4.1 Sequence Diagram of "Visualize Historical Data" use case

Figure 3.3 presents the unified sequence diagram for the “Visualize Historical Data” use case, showing how the system coordinates the Researcher, Spring Boot Gateway, Processing Microservice, Cache, and data sources. At its core, this use case generalizes into two alternative workflows—*Visualize Custom Dataset* and *Visualize Sensor Data*—each corresponding to a different data origin. In this subsection, we introduce the complete, generalized sequence diagram that encompasses both branches. In the two following subsections, we will examine each specialization in turn: first the “Visualize Custom Dataset” flow (Sub-section 3.4.2), and then the “Visualize Sensor Data” flow (Sub-section 3.4.3), detailing the unique steps and interactions that arise once the user’s data source choice drives the system down one of these two paths.

### 3.4.2 Visualize Dataset Data

When the researcher elects to visualize a custom dataset (the first alternative in Figure 3.3), the frontend issues an HTTP request to the Processing Microservice that includes the list of *attributes* (columns) the user wishes to plot. Upon receiving this request, the controller performs the following steps:

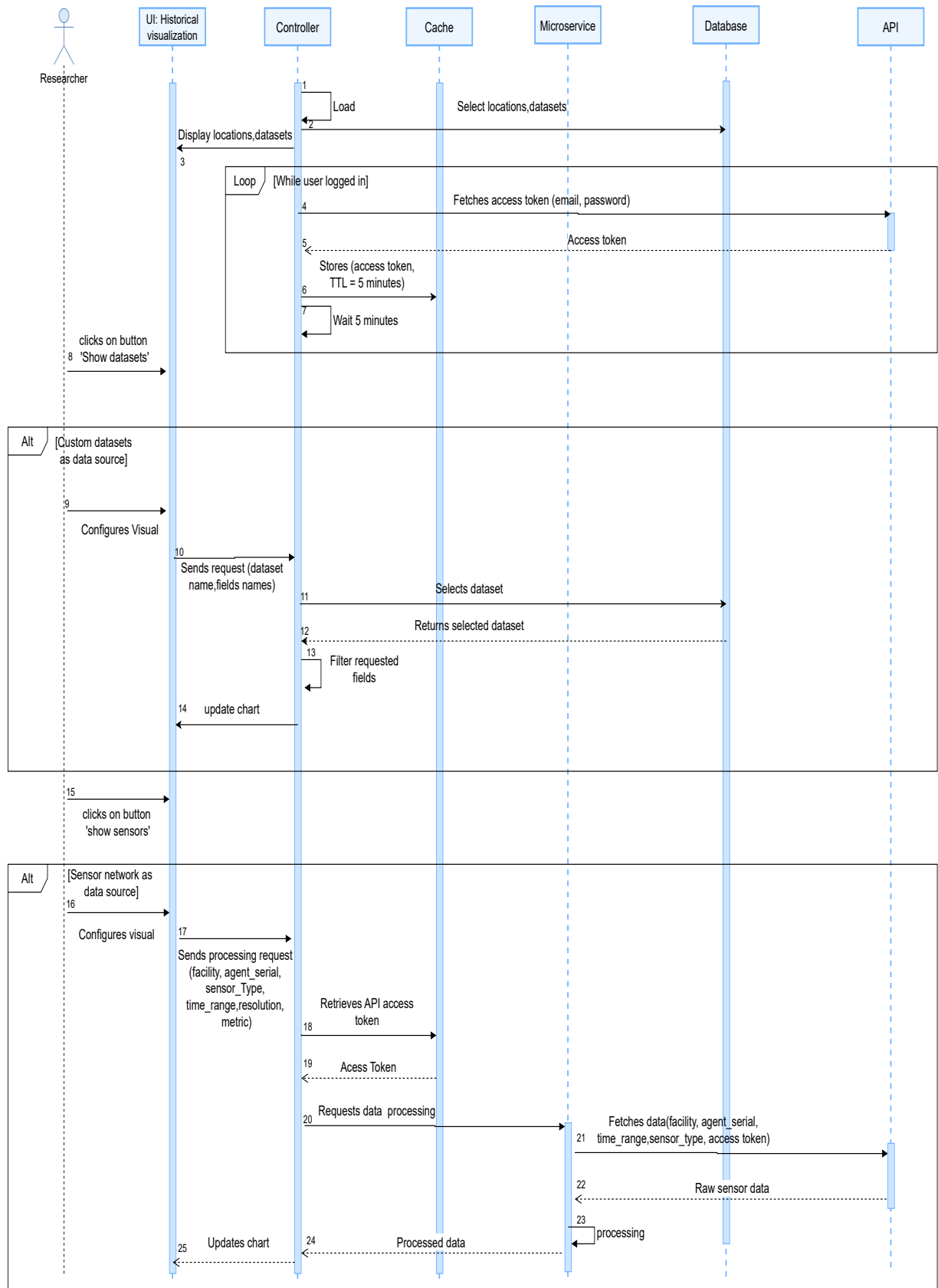


Figure 3.3: Historical Data Visualization Sequence Diagram

1. It queries the Application Database for the stored payload of the dataset.
2. It filters the returned records to include only the requested attributes.
3. It formats the filtered data as a JSON time-series and returns it to the Spring Boot Gateway.
4. The gateway forwards the JSON payload to the UI component, where the visualization will be rendered.

### 3.4.3 Visualize Sensor Data

When the researcher elects to visualize data from the sensor network, the system follows the second alternative in Figure 3.3). In this scenario, the frontend assembles a request containing seven parameters that identify the exact sensor stream to plot:

1. *Facility*: Geographic facility (e.g., Aizu University, ISTIC)
2. *Location*: Sub-area within the facility (e.g., Classroom A006 at ISTIC)
3. *Sensor Box*: Hardware unit of sensors identified by a unique (`agent_serial`, e.g., `dca63282ba53`, see Figure 1.3)
4. *Sensor Type*: Measurable parameter such as humidity, luminance, motion, temperature, thermography, or microphone, as listed in Table 1.2
5. *Time Range*: Start/end dates of the interval data is fetched (e.g., from 10 am-11 am, 1–31, July 2023)
6. *Resolution*: Aggregation granularity (minute/hourly/daily/monthly/yearly)
7. *Metric*: Statistical operators—mean, median, max, min, skewness—are applied to compute aggregated trends

Once these parameters are submitted, the controller retrieves the cached API access token, forwards it alongside the user’s sensor parameters to the Processing Microservice, and then the microservice orchestrates the multi-stage pipeline—detailed in part **a**—to *retrieve*, *clean*, *filter*, *aggregate*, and *cache* the sensor readings. Finally, the resulting time series is returned to the UI for rendering, completing the sensor-data visualization path.

First, in part **a** we establish the necessity of specialized strategies—hierarchical temporal aggregation and adaptive caching—to handle high-volume sensor inputs at scale. Then, in part **b**, with reference to the activity diagram in Figure 3.4 and Algorithm 1, we detail the end-to-end workflow by which the Historical-Processing Microservice converts raw readings into the aggregated time series displayed on the dashboard.

#### **a Historical-Processing Microservice: Addressing Scalability Challenges**

The high-frequency nature of sensor data (525,600 points/year) necessitates two key innovations:

### Temporal Aggregation Hierarchy

The very nature of historical visualization is **inherently tied to time**, specifically to a *hierarchical time axis*. While sensors generate **data every minute** (approx. 525,600 data points per year), the real value lies in transforming this high-frequency data into a **user-friendly, trend-explorable** format.

To make trends perceptible, the system aggregates data into tiers:

**Minute → Hourly → Daily → Monthly → Yearly**

This hierarchy enables three critical operations:

- **Roll-Up:** Identify macro-trends (e.g., hourly → daily averages)
- **Drill-Down:** Inspect granular patterns (e.g., monthly → daily peaks)
- **Metric Comparison:** Apply statistical operators across resolutions

### Adaptive Caching Strategy

To maintain reasonable response times when visualizing large volumes of historical data, SARA employs an adaptive, hierarchical caching strategy. The core idea is to cache precomputed *linear* aggregates at each temporal resolution in Redis Cache, allowing rapid composition of higher-level summaries without reprocessing raw readings. Non-linear metrics, by contrast, are computed on demand.

- **Linear Metrics** (Cacheable: sum / max / min / mean):

Linear metrics—such as **sum**, **max**, **min**, and **mean**—are algebraically composable, meaning that aggregates at one level can be derived from aggregates at a finer level. For example, the daily maximum can be obtained by taking the maximum of 24 hourly maxima. Accordingly, SARA precomputes and caches each of these metrics at all temporal tiers (minute, hourly, daily, etc.).

- **Non-Linear Metrics** (median/skewness):

Metrics such as **median** and **skewness** cannot be composed from partial aggregates. As a result, SARA does not cache these values.

## b Historical-Processing Microservice: Workflow

This section details how the historical-visualization microservice ingests raw sensor readings and produces an aggregated time-series suitable for interactive display. The pipeline comprises six stages formalized in Algorithm 1) and follows the activity diagram in Figure 3.4. At each stage, the system chooses between *cache lookup*, *hierarchical roll-up*, or *raw-data aggregation* to optimize latency and accuracy.



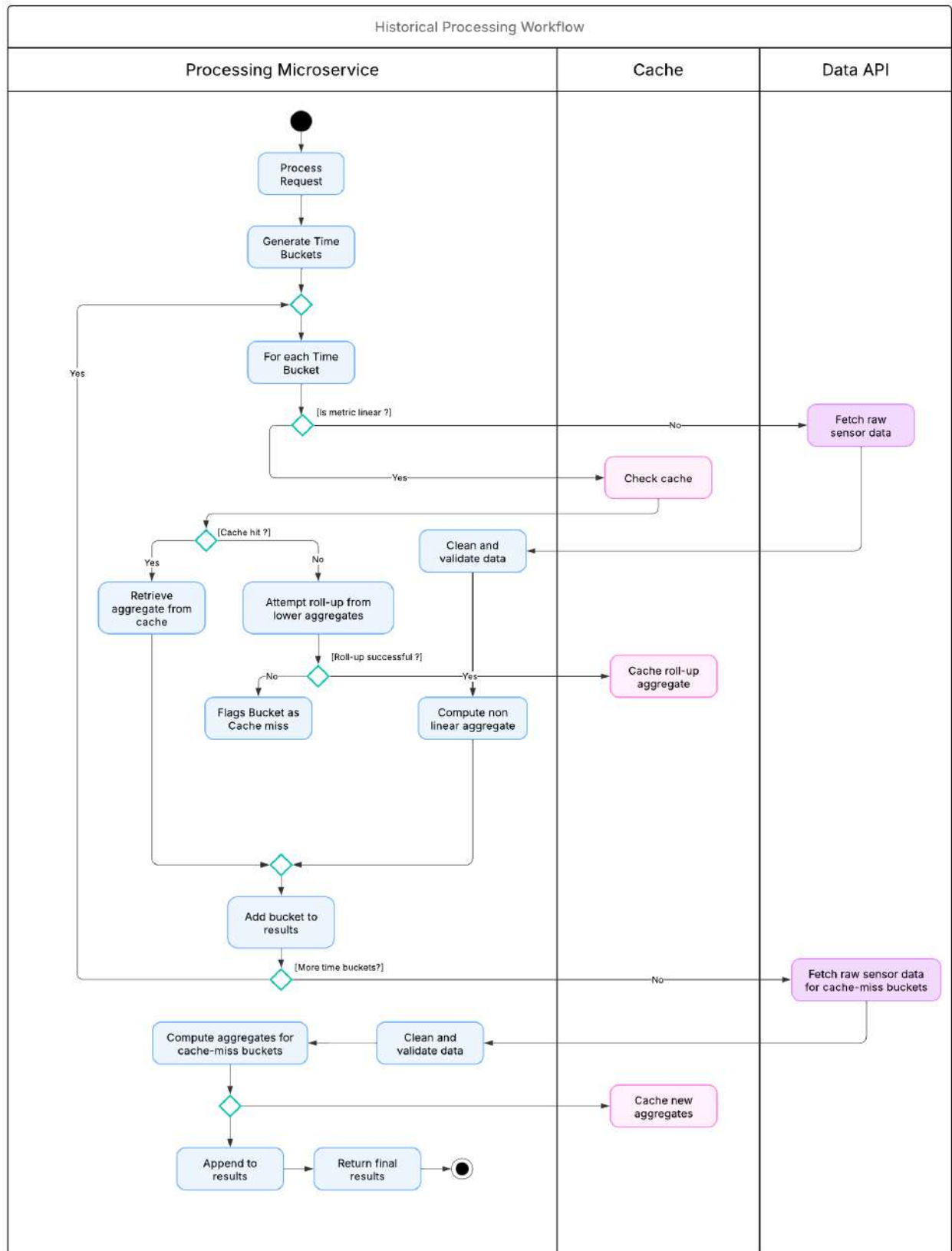


Figure 3.4: Processing Microservice Activity Diagram

**Notation.**

$[t_1, t_2]$  User-requested time interval.

$L$  Aggregation level, one of {minute, hourly, daily, monthly, yearly}.

$\Delta_L$  Duration of one bucket at level  $L$ , defined as

$$\Delta_L = \begin{cases} 1 \text{ minute,} & L = \text{minute,} \\ 1 \text{ hour,} & L = \text{hourly,} \\ 24 \text{ hours,} & L = \text{daily,} \\ 1 \text{ month,} & L = \text{monthly,} \\ 1 \text{ year,} & L = \text{yearly.} \end{cases} \quad (3.1)$$

$m$  Metric operator, either

- *Linear*: sum, min, max, mean
- *Non-Linear*: median, skewness

$B$  Set of buckets covering  $[t_1, t_2]$ , i.e.  $B = \{B_1, \dots, B_N\}$ .

$B_k$  Single bucket  $[t_k^{\text{start}}, t_k^{\text{end}}]$  with  $t_k^{\text{end}} = t_k^{\text{start}} + \Delta_L$ .

**Global Example Setup.**

To make the abstract workflow concrete, we illustrate each stage using a running example. Specifically, consider a researcher who requests the *daily maximum* temperature in Classroom M5 over a five-day period. Formally, we set:

$$[t_1, t_2] = [2023-07-01, 2023-07-05], \quad L = \text{daily}, \quad m = \text{max.}$$

Under these parameters, the system will partition the interval into daily buckets and compute the maximum temperature within each bucket, as detailed in the subsequent workflow stages.

**Workflow Stages**

Having defined our notation and example, we decompose the Historical-Processing pipeline into six sequential stages outlined in Figure 3.4), each responsible for a distinct transformation from raw readings to final aggregates:

(a) **Request Initiation.**

The Processing Microservice parses the user's parameters into:

$$[t_1, t_2], \quad L, \quad m.$$

*Example:*  $[2023-07-01, 2023-07-05]$ , daily, max.

(b) **Time Buckets Generation.**

Partition  $[t_1, t_2]$  into  $N$  buckets of length  $\Delta_L$ , where  $\Delta_L$  follows Eq 3.1 :

$$B_k = [t_k^{\text{start}}, t_k^{\text{start}} + \Delta_L], \quad k = 1, \dots, N.$$

*Example:* with  $\Delta_{\text{daily}} = 24 \text{ h}$ . This yields four daily buckets:

$$B_1 = [07/01, 07/02], \quad B_2 = [07/02, 07/03], \quad B_3 = [07/03, 07/04], \quad B_4 = [07/04, 07/05].$$

(c) **Metric Classification.**

Determine if  $m$  is *linear* or *non-linear* (see **a**):

- *Linear*: sum, min, max, mean
- *Non-Linear*: median, skewness

*Example*: max is linear.

(d) **Per-Bucket Processing.**

For each bucket  $B_k$ , the microservice applies exactly one of three methods, in order of preference, to compute  $\mu_k$ :

- **Cache Lookup.**

If a precomputed  $\mu_k$  exists at level  $L$ , fetch it directly.

- **Hierarchical Roll-Up.**

Only for *linear* metrics: if no cache entry at  $L$ , collect cached aggregates from the next finer level  $L^-$  (e.g. daily  $\rightarrow$  hourly) and recombine:

$$\mu_k = \max_i \mu_{k,i} \quad (\text{for max, analogously for other operators}).$$

If this succeeds, cache the result at level  $L$ .

- **Raw-Data Fallback Flagging.** If neither cache lookup nor roll-up succeeds (or if  $m$  is non-linear), mark  $B_k$  for fallback processing (Stage e).

(e) **Raw-Data Fallback.**

For each  $B_k$  marked above:

- Data API returns minute-level sensor readings.
- Compute  $\mu_k$ , then cache (only if linear) with Time To Live (TTL).
- Append  $\mu_k$  to results.

(f) **Results Compilation.**

Assembling the per-bucket aggregates yields the final series (for our global example) as illustrated in Eq 3.2

$$[\mu_1, \mu_2, \mu_3, \mu_4] = [30\text{ C}, 28\text{ C}, 27\text{ C}, 29\text{ C}]. \quad (3.2)$$

## Cache Key Design and TTL Management

To avoid collisions, cache keys encode: `aggregate:{sensor_type}:{agent_serial}:{aggregation_level}:{metric}:{start_iso}:{end_iso}`

*Example*:

`aggregate:temperature:A002:daily:max:2023-07-02T00:00:00:2023-07-02T23:59:59`

## TTL(Time To Live) Rationale.

TTLs define how long a computed metric remains in the Cache

- *Historical buckets* (completed periods) use long TTLs following Eq 3.3:

$$\text{TTL}(L) = \begin{cases} 7d & L = \text{minute} \\ 30d & L = \text{hourly} \\ 90d & L = \text{daily} \\ 1y & L = \text{monthly} \\ 2y & L = \text{yearly} \end{cases} \quad (3.3)$$

- *Ongoing buckets* (current interval) use short TTLs (e.g. monthly aggregates expire every new day) to reflect real-time updates (e.g., the current [01-05-2025,31-05-2025] monthly bucket at 11-05-2025 is still receiving new data).

---

**Algorithm 1** Hierarchical Temporal Aggregation

---

```
1: Input:
   • Buckets  $B = \{B_k\}_{k=1}^N$ 
   • Metric operator  $m$ 
2: Output: Aggregated values  $\{\mu_k\}_{k=1}^N$ 
3: Initialize  $aggregatedList \leftarrow \emptyset$ 
4: Initialize  $rawList \leftarrow \emptyset$ 
5: for each  $B_k$  in  $B$  do
6:   if cache.contains( $B_k$ ) then
7:     Retrieve  $\mu_k$  from cache
8:     Append  $\mu_k$  to  $aggregatedList$ 
9:   else
10:    if  $m$  is linear then
11:      if rollUp( $B_k$ ) succeeds then
12:        Compute  $\mu_k \leftarrow \text{rollUp}(B_k)$ 
13:        Cache  $\mu_k$ 
14:        Append  $\mu_k$  to  $aggregatedList$ 
15:      else
16:        Add  $B_k$  to  $rawList$ 
17:      end if
18:    else
19:      Compute  $\mu_k \leftarrow \text{computeRaw}(B_k)$ 
20:      Append  $\mu_k$  to  $aggregatedList$ 
21:    end if
22:  end if
23: end for
24: if  $rawList \neq \emptyset$  then
25:   for each  $B_k$  in  $rawList$  do
26:     Compute  $\mu_k \leftarrow \text{computeRaw}(B_k)$ 
27:     Cache  $\mu_k$ 
28:     Append  $\mu_k$  to  $aggregatedList$ 
29:   end for
30: end if
31: return  $aggregatedList$ 
```

---

In summary, this six-stage workflow provides a robust solution to the challenges posed by high-volume sensor data. By combining hierarchical temporal aggregation—reducing query latency—with adaptive caching—leveraging structured keys and well-tuned TTL policies—the system ensures both responsiveness and data freshness during historical visualization.

## 3.5 Real-Time Monitoring: Conceptual Design

Building on the component decomposition in Section 3.2, the Real-Time Monitoring module provides researchers with live insights into environmental and occupancy metrics. It captures high-frequency sensor data streams from geographically distributed sites (classrooms, cafeteria, museum halls) and delivers continuous updates via a fault-tolerant, scalable event-driven architecture (Event-Driven Architecture (EDA)). At its core, SARA cleanly decouples *data producers*—the External API and underlying sensors—from *data consumers*—the dashboard components—enabling users to switch seamlessly between sensor types and locations without performance degradation.

### 3.5.1 Event-Driven Architecture

An *event-driven architecture* (EDA) is a software design paradigm in which decoupled components communicate by producing and consuming discrete events. An event is a record of a noteworthy change of state or an occurrence, which in our case is **a new sensor reading**. EDAs typically comprise three roles:

1. **Event Producers:** Components that detect or fetch raw data and publish it as events.
2. **Event Router (Broker):** Middleware that routes published events to interested subscribers.
3. **Event Consumers:** Components that subscribe to events, process and render results.

### 3.5.2 Sequence Diagram of The "Visualize Real-Time Data" use case.

Figure 3.5 presents the sequence diagram that illustrates how the components of SARA’s event-driven architecture (EDA) collaborate during real-time data visualization. This diagram provides a concrete runtime view of how sensor readings are propagated from their source to the dashboard. As introduced in Section 3.5.1, the Real-Time Monitoring pipeline is structured around three core EDA roles—event producers, router, and consumers—each fulfilled by dedicated components. Figure 3.5 captures how these components interact in sequence to deliver a low-latency, responsive monitoring experience.

#### a. Event Producers

These components generate and publish new sensor readings as events:

1. **Data API:** Fetches raw sensor data from the repository where sensor readings are stored.
2. **Scheduler:** Orchestrates continuous data flow by issuing sensor-data requests to the *Data API* at a configurable interval (default: one minute).

Together, these producers feed EventBus with up-to-date readings.

#### b. Event Router (Broker): EventBus

Serving as an in-memory broker, the EventBus decouples producers from consumers:

1. *Role:* Central publish/subscribe middleware for sensor events.

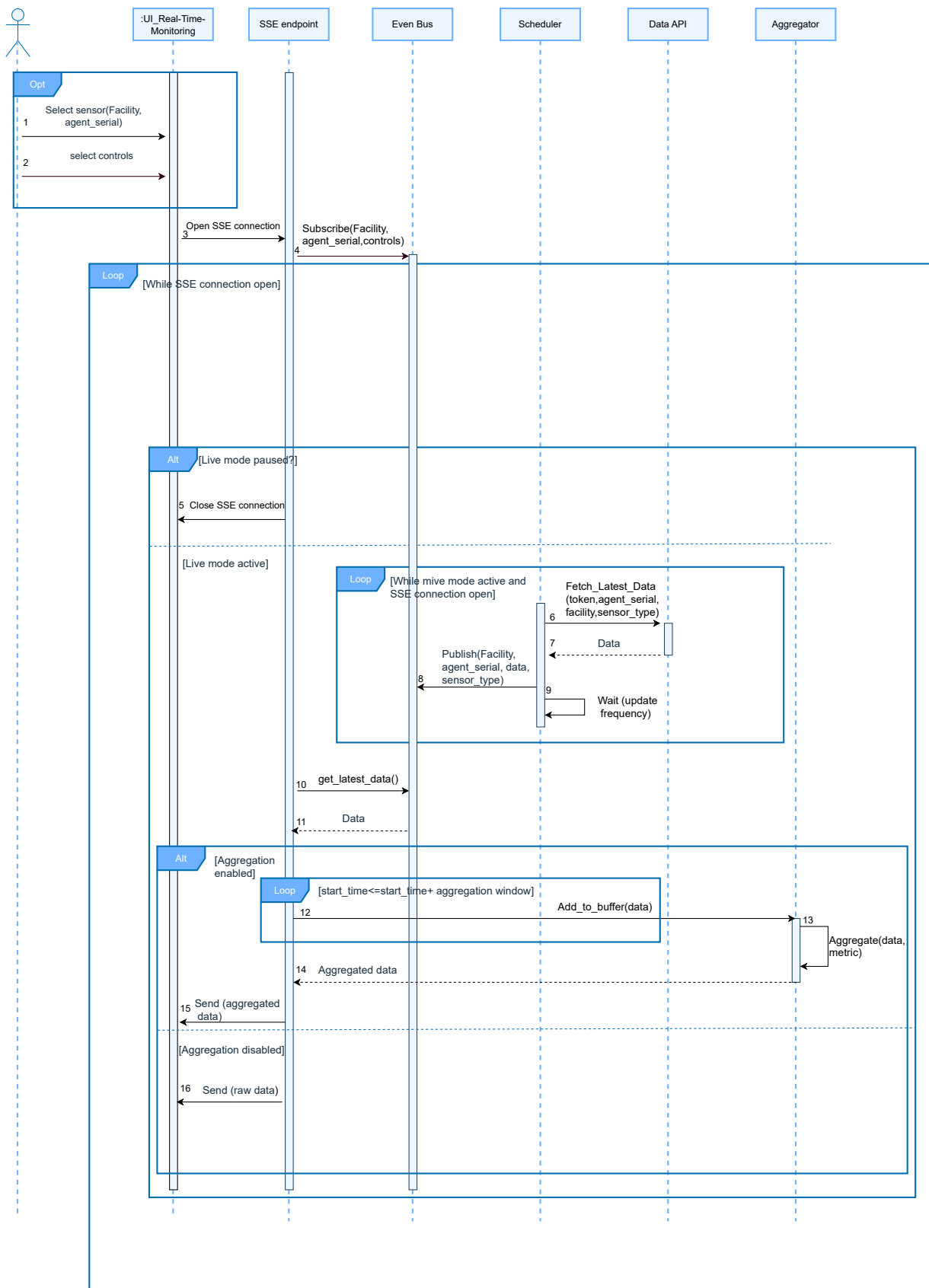


Figure 3.5: Sequence diagram of the Real-Time Monitoring workflow.

## 2. *Design:*

- The EventBus's design rests on three guiding principles to ensure robust, low-latency distribution.
  - First, it is implemented as a *singleton instance*, guaranteeing that only one router handles all subscriptions and thereby preventing duplicate or conflicting registrations.
  - Second, it employs a *topic-based subscription model*, by which clients explicitly register the sensor streams they wish to follow (e.g. `facility:istic / agent_serial:dca63282b9da / sensor: temperature`).
  - Finally, it incorporates an *event storage* mechanism that retains the most recent event for each topic, allowing new subscribers to immediately receive the latest reading upon connection.

## 3. *Workflow:*

- *Publish:* Producers push events to the EventBus.
- *Subscribe:* SSE endpoints subscribe to relevant topics.
- *Notify:* The EventBus forwards each event to all registered subscribers.

## c. Event Consumers

These components receive routed events, transform them if necessary, and update the UI:

### 1. SSE Endpoint (FastAPI)

- *Role:* Exposes Server-Sent Events channels for front-end clients.
- *Workflow:*
  - Opens and closes SSE connections for subscribed topics.
  - Applies user-configured modifiers (update frequency, aggregation window and metric, pause).
  - Streams JSON-encoded events to the front-end dashboards.

### 2. Frontend Dashboards

- Renders charts and heatmaps in real time.
- Provides controls for selecting topics (facility, agent serial, sensor type), chart types, and metrics.

### 3. Aggregator

- *Role:* Buffers recent events in time windows to compute metrics.
- *Workflow:*
  - Computes metrics (mean, sum, max, min, count) over user-defined time windows.
  - Supplies aggregated values to the dashboard for display.

On the frontend, users influence the event pipeline via intuitive controls:

- **Subscription Parameters:** Users can select facility, sensor box (`agent_serial`), and sensor type to tailor SSE topics.

- **Stream Behavior:**

- *Frequency*: Set update interval (default 60 s).
- *Aggregation*: Choose sliding-window metric.
- *Pause/Resume*: Temporarily suspend or restart event delivery.

These controls dynamically adjust the SSE configuration, ensuring that researchers maintain full command over their live monitoring experience.

## 3.6 Manage Custom Datasets: Conceptual Design

The ability to integrate custom datasets into the SARA platform empowers researchers to conduct collaborative, cross-disciplinary analyses by combining pre-collected sensor data with domain-specific datasets. This functionality addresses the need for flexibility in academic research, where heterogeneous data formats and access controls are critical. Custom datasets are subsequently visualized alongside historical sensor data, enabling researchers to correlate institutional metrics with their own observations.

In this section, we detail the "*Manage Custom Datasets*" use case, focusing specifically on dataset creation and visibility update. These processes ensure that datasets adhere to SARA's validation rules and access policies, which directly govern their **availability** in the Historical visualization workflows.

### 3.6.1 Sequence Diagram of the "Create Dataset" use case

The dataset creation workflow consists of three main steps: submission, validation, and persistence. Figure 3.6 illustrates the sequence of interactions among the researcher, frontend, backend, and database, covering these three steps.

1. **Dataset Submission:** Researchers enter the dataset's name, description, payload (JSON or CSV), visibility setting (public, private, or restricted), a list of collaborators, if any, and they upload a JSON Schema.
2. **Validation:** The backend verifies the payload against the provided schema, and validation errors are reported immediately, and the user can correct the schema or payload and revalidate.
3. **Persistence:** After successful validation, the user saves the dataset. The system records the metadata (name, description, visibility, schema) and payload in the `datasets` table. If visibility is restricted, collaborator entries are added to `dataset_collaborators`. A confirmation of successful creation is returned.

### 3.6.2 Sequence Diagram of the "Update Dataset Visibility" Use Case

Researchers can modify a dataset's visibility and collaborator list at any time. Figure 3.7 shows the update sequence.



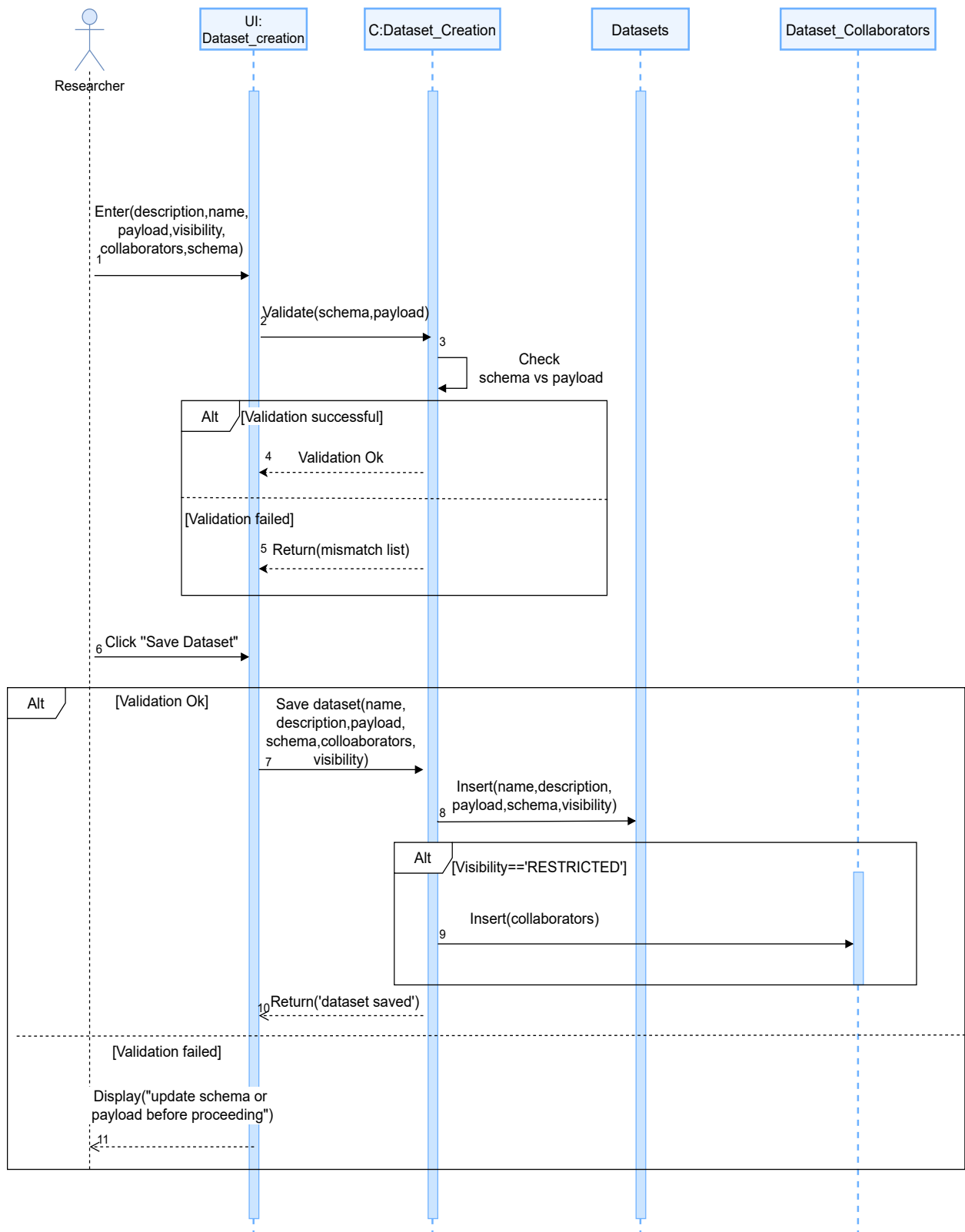


Figure 3.6: Sequence diagram for creating a custom dataset.

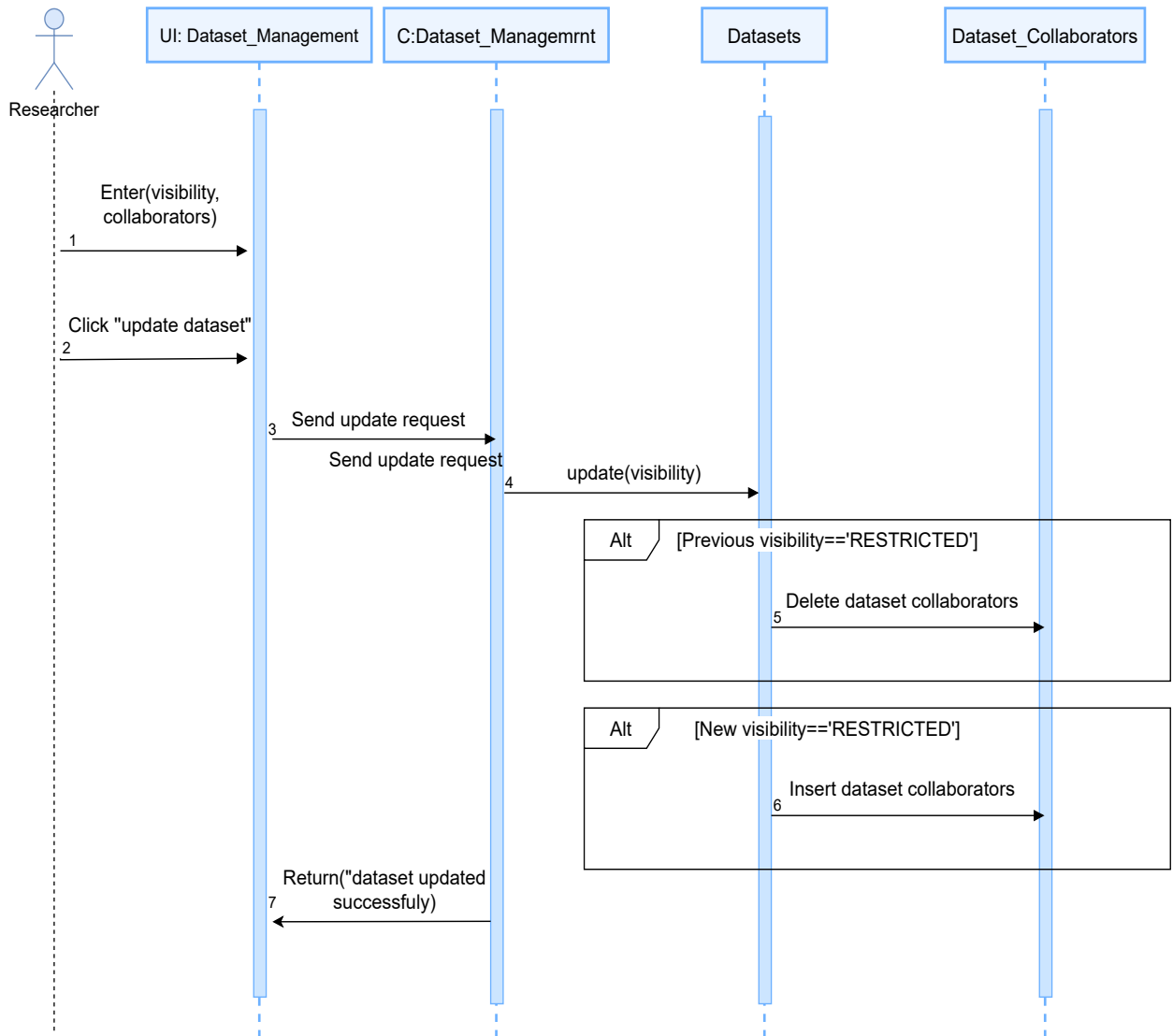


Figure 3.7: Sequence diagram for updating dataset visibility.

When a dataset's visibility setting is changed, SARA enforces a deterministic update of its collaborator records. Specifically, switching from *Restricted* to *Public* or *Private* automatically removes all existing collaborators, since broader visibility has been granted or revoked. Conversely, assigning a dataset to *Restricted* mode requires the owner to specify at least one collaborator, whose entries are then atomically inserted into the `dataset_collaborators` table.

Access at runtime is governed by three straightforward rules:

- **Owners** always see all datasets they have created.
- **Collaborators** see only those datasets for which they have been explicitly granted access.
- **Public** datasets are visible to every authenticated user.

The “Visualize Historical Data” interface dynamically filters available datasets based on these rules, ensuring compliance with SARA’s sharing policies.

## 3.7 Manage Reference Data: Conceptual Design

Reference data management is a foundational use case that ensures all sensor-driven interfaces—namely, the historical data visualization and real-time monitoring—operate with accurate and up-to-date contextual information. By empowering administrators to add, update, or remove deployment entities, the system maintains extensibility and long-term maintainability without disrupting researcher workflows.

### 3.7.1 Overview of Reference Data

Reference data refers to the core structural entities that define the sensor deployment context. These include:

- The list of **facilities (Sites)** (e.g., *ISTIC*, *The Aizu University*),
- The list of **locations** within each facility (e.g., *A006*, *Classroom M5*),
- The list of **sensor boxes** (i.e., hardware units identified by agent serials),
- The list of **sensor types** (e.g., humidity, temperature, motion), and
- The assignment between **sensor types and sensor boxes**, representing the physical deployment of sensors.

### 3.7.2 Create Site

While the following subsections focus on the *Site* entity, the same pattern applies to all other reference data classes. To introduce a new facility into the system, the administrator invokes the **Create Site** workflow as illustrated in Figure 3.8.

1. *Form Initialization*: The admin selects “Add Site,” and the UI renders a blank input form.
2. *Data Entry*: The admin provides the site name, display color, and (optionally) a description.
3. *Validation*: The system checks for uniqueness of the site name to prevent duplicates.
4. *Persistence*: On successful validation, a new **Site** record is created in the database.
5. *Propagation*: The UI confirms creation and updates the site table in the database, making the new site immediately available to all dependent interfaces.

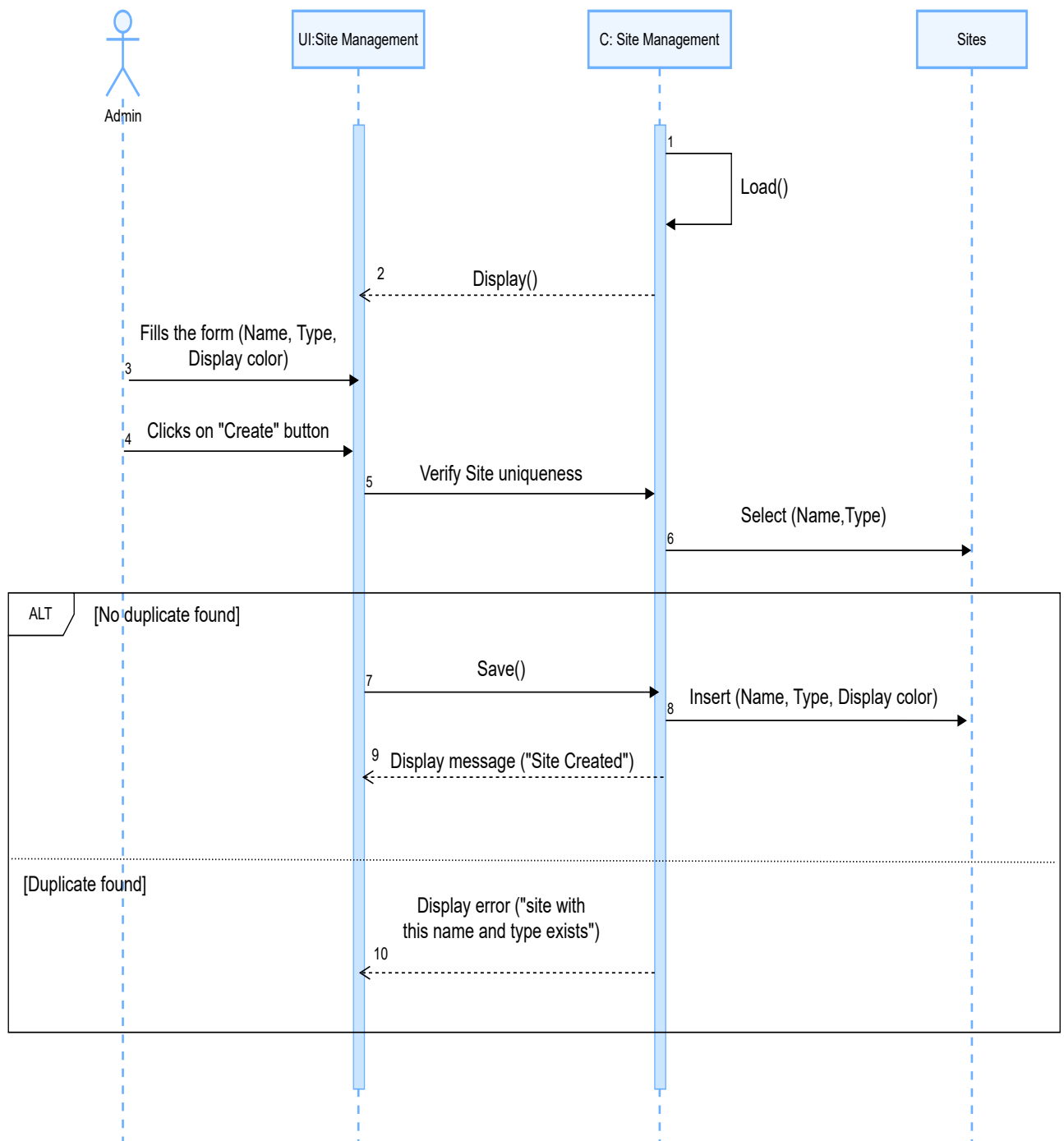


Figure 3.8: Sequence diagram for the Create Site workflow.

### 3.7.3 Update Site

The **Update Site** operation (Figure 3.9) mirrors the creation flow with key differences in the validation and persistence steps:

1. *Selection*: The admin chooses an existing site from the reference list.
2. *Form Population*: Current site attributes are loaded into an editable form.
3. *Modification*: The admin changes the name, type, display color, or description.
4. *Validation*: The system verifies that the updated name remains unique among other sites.
5. *Propagation and Persistence*: Changes are pushed to the site table and reflected across all visualization and management interfaces.

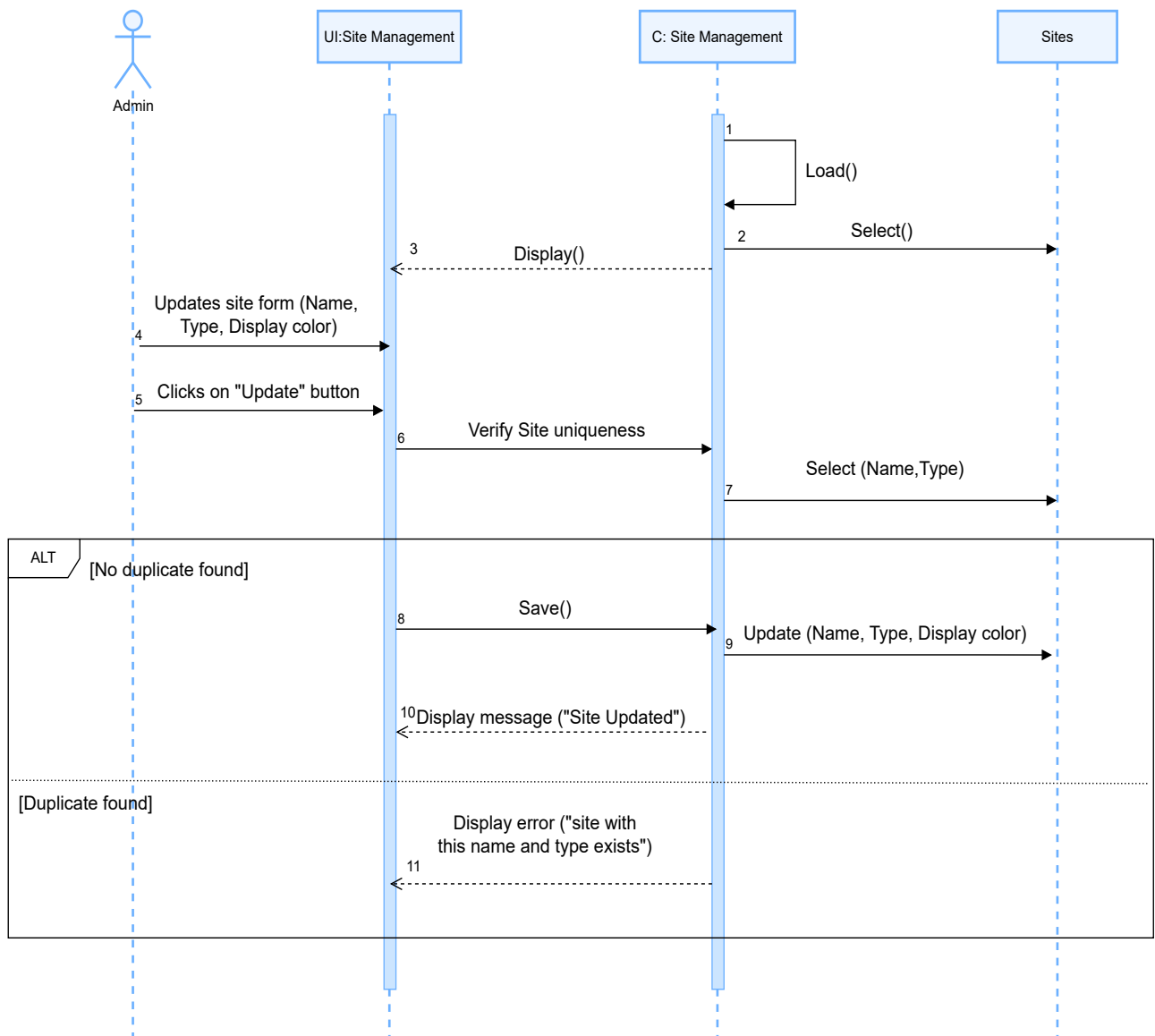


Figure 3.9: Sequence diagram for the Update Site workflow.

### 3.7.4 Delete Site

The **Delete Site** workflow outlined in Figure 3.10 includes an additional dependency check to prevent orphaned child entities:

1. *Selection*: The admin selects a site for deletion.
2. *Dependency Check*: The system verifies that no locations are still associated with the site.
3. *Confirmation*: If dependencies exist, the UI alerts the admin; otherwise, it prompts for final confirmation.
4. *Removal*: The **Site** record is removed from the database.

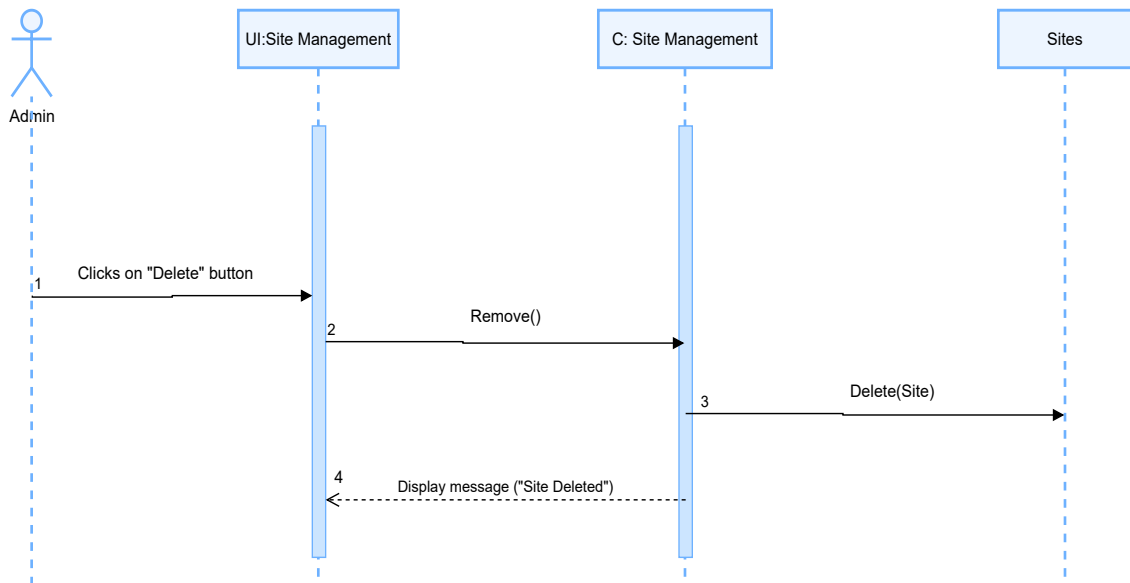


Figure 3.10: Sequence diagram for the Delete Site workflow.

The same conceptual CRUD patterns apply to **Location**, **SensorBox**, **SensorType**, and **SensorAssignment**. Each entity maintains its own validation rules and dependency checks, but the overall flow—Select, Populate, Validate, Persist, Propagate—remains constant.

## 3.8 Global Class Diagram

Figure 3.11 presents the global class diagram for SARA, encompassing user metadata, custom dataset management, and reference data entities. Each class maps directly to a database table and corresponds to one or more use cases described in Chapter 3. The design emphasizes clear separations of concern, support for multi-device sessions, and flexible many-to-many relationships.

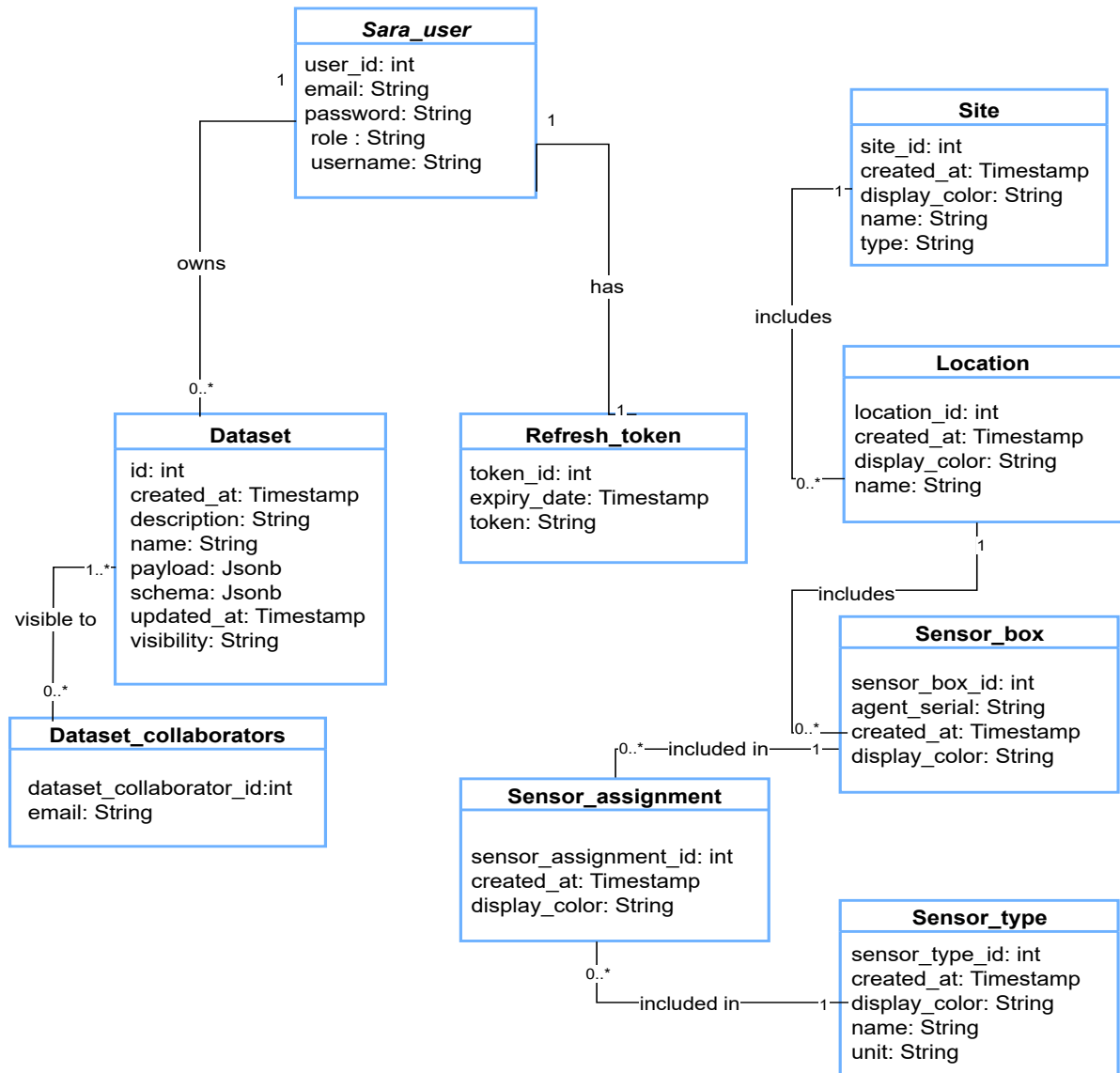


Figure 3.11: Global Class Diagram of the SARA Platform

## 1. User Metadata Classes

- a. **SaraUser** The `SaraUser` class encapsulates authentication and authorization data: `email`, `password` (hashed), `username`, and `role`. It underpins the Sign In and Sign Up use cases, providing identity and access control across the system.
- b. **RefreshToken** To maintain secure, long-lived sessions, the `RefreshToken` class holds `tokenId`, `token` (secure random string), and `expiryDate`. It has a one-to-one association with `SaraUser`. By storing refresh tokens separately, SARA can potentially support multiple concurrent device sessions per user and implement the 15-minute access-token rotation policy detailed in Section 3.3.

## 2. Custom Dataset Classes

- a. **Dataset** The `Dataset` class represents user-provided or collaboratively shared data. Attributes include `name`, `description`, `payload` (JSON), `schema`, `visibility`, `createdAt`, and `updatedAt` timestamps. Each `Dataset` is owned by one `SaraUser` (one-to-many).
- b. **DatasetCollaborator** This class models dataset sharing by linking multiple collaborators to datasets in a normalized many-to-many relationship. Rather than embedding collaborator lists within the `Dataset` entity itself, this table captures external associations, storing both the dataset ID and the collaborator's email address. This separation enhances schema normalization, prevents data duplication, and allows fine-grained sharing.

## 3. Reference Data Classes

- a. **Site, Location, SensorBox, SensorType** These four classes capture the hierarchical deployment context detailed in Section 3.7:
  - `Site` defines `siteId`, `name`, `displayColor`, and `creationTimestamp`.
  - `Location` belongs to one `Site` and adds `locationId`, `name`, `displayColor`, and `creationTimestamp`.
  - `SensorBox` represents a physical module with `agentSerial`, `name`, `displayColor`, and `creationTimestamp`, linked one-to-many under `Location`.
  - `SensorType` lists possible measurements (temperature, humidity, motion, etc.) with fields `typeId`, `name`, and `unit`.
- b. **SensorAssignment** The `SensorAssignment` class associates `SensorBox` and `SensorType`, specifying which sensor types are installed in which boxes. This many-to-many mapping (via an assignment table) supports dynamic reconfiguration of modules and drives the dynamic lists in the Historical Visualization interface.

# 3.9 Conclusion

This chapter presented the conceptual foundations of the SARA platform, highlighting its **modular**, **scalable**, and **reusable** design. By structuring the system around **self-contained components**, it ensures coherent interactions and facilitates future extensions. Use cases and sequence diagrams illustrated how core functionalities—authentication, real-time monitoring, historical processing, and dataset management—are orchestrated. The next chapter focuses on their concrete implementation.



# Chapter 4

## Implementation

### 4.1 Introduction

This chapter describes the concrete realization of the SARA platform design, detailing the software environment, core technologies, and user-facing interfaces that enable real-time monitoring, historical data visualization, custom dataset management, and administrative reference data configuration. We begin by presenting the development and operational tools in Sections 4.2, followed by an overview of the principal frameworks and libraries in Section 4.3. The subsequent sections then walk through each key use case implementation. Each section explains the rationale, user interactions, illustrating how the abstract component models from Chapter 3 have been brought to life.

### 4.2 Software Environment

To develop and maintain the SARA platform effectively, we adopted a suite of tools covering collaborative design, code editing, version control, containerization, and database management. These software components, presented in Table 4.1, ensured that we could work in concert, maintain reproducible environments, and inspect live data stores during development and testing:

### 4.3 Frameworks and Technologies Used

The SARA platform’s architecture relies on a combination of backend frameworks, frontend libraries, and in-memory caching to deliver high-performance, interactive data services. In Table 4.2, we summarize the key technologies and their roles:

### 4.4 User Authentication and Sign-Up Implementation

Secure access to the SARA platform is enforced via a two-step process: user authentication (sign-in) and, if necessary, user registration (sign-up). The interface appears immediately upon loading the application, directing users to the appropriate workflow based on their account status.

Table 4.1: Software Environment Tools





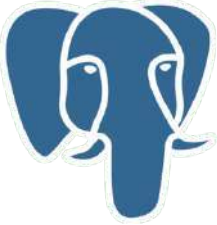





| Software  | Definition and Role in SARA development  |
|---|--|
|    | <b>Visual Studio Code:</b> A lightweight, extensible code editor from Microsoft with integrated Global Information Tracker (Git) support, debugging, and a rich plugin ecosystem, serving as the primary Integrated Development Environment (IDE) for implementing frontend and backend code [44]. |
|    | <b>Docker:</b> A containerization platform that packages the SARA microservices and their dependencies into isolated, portable containers, ensuring consistent environments across development, testing, and production [4].   |
|   | <b>GitHub:</b> GitHub is a web-based platform for hosting and reviewing Git repositories, offering distributed version control, issue tracking, and pull-request workflows to facilitate collaborative software development [17].  |
|  | <b>Monday.com:</b> Monday.com is a cloud-based work operating system that enables teams to plan, organize, and track tasks and projects via customizable boards, automations, and timeline views [23].   |
|  | <b>RedisInsight:</b> RedisInsight is a graphical user interface for Redis, providing real-time monitoring, key browsing, and data visualization tools to inspect in-memory data structures and performance metrics [30].   |
|  | <b>pgAdmin:</b> pgAdmin is an open-source administration and development platform for PostgreSQL, offering graphical tools for schema design, Structured Query Language (SQL) query execution, and database maintenance [26].  |

Table 4.2: Key Technologies and Role in SARA development

| Technology   | Definition and Role in SARA  |
|--|--|
|  <b>FastAPI</b>   | <b>FastAPI:</b> A modern, high-performance web framework for building RESTful APIs in Python, leveraging standard Python type hints for automatic data validation and interactive documentation—used to implement the real-time and historical data visualization microservices [5]. |
|                  | <b>Spring Boot:</b> A Java framework that simplifies development of stand-alone, production-grade applications; served as the API gateway responsible for authentication, request routing, and CRUD operations on metadata [39].   |
|  <b>ECHARTS</b> | <b>Apache ECharts:</b> A powerful JavaScript visualization library providing interactive, customizable charting components, employed on the frontend to render time-series and comparison visualizations [2].  |
|                 | <b>Angular:</b> A TypeScript-based open-source web application framework for building dynamic single-page applications; used to structure the SARA UI, and bind to SSE data streams [1].   |
|  <b>redis</b>   | <b>Redis Cache:</b> An in-memory key-value store known for sub-millisecond latency; used to cache API access tokens and precomputed aggregates for fast retrieval by microservices [30].   |

### 4.4.1 Login Interface

When the application launches, the user is presented with the *Login* screen illustrated in Figure 4.1. This form collects:

- **Email:** the user’s registered address.
- **Password:** the secret passphrase (minimum 6 characters).
- **Sign In** button to submit credentials.
- “Don’t have an account? **Sign Up**” link to access the registration form.

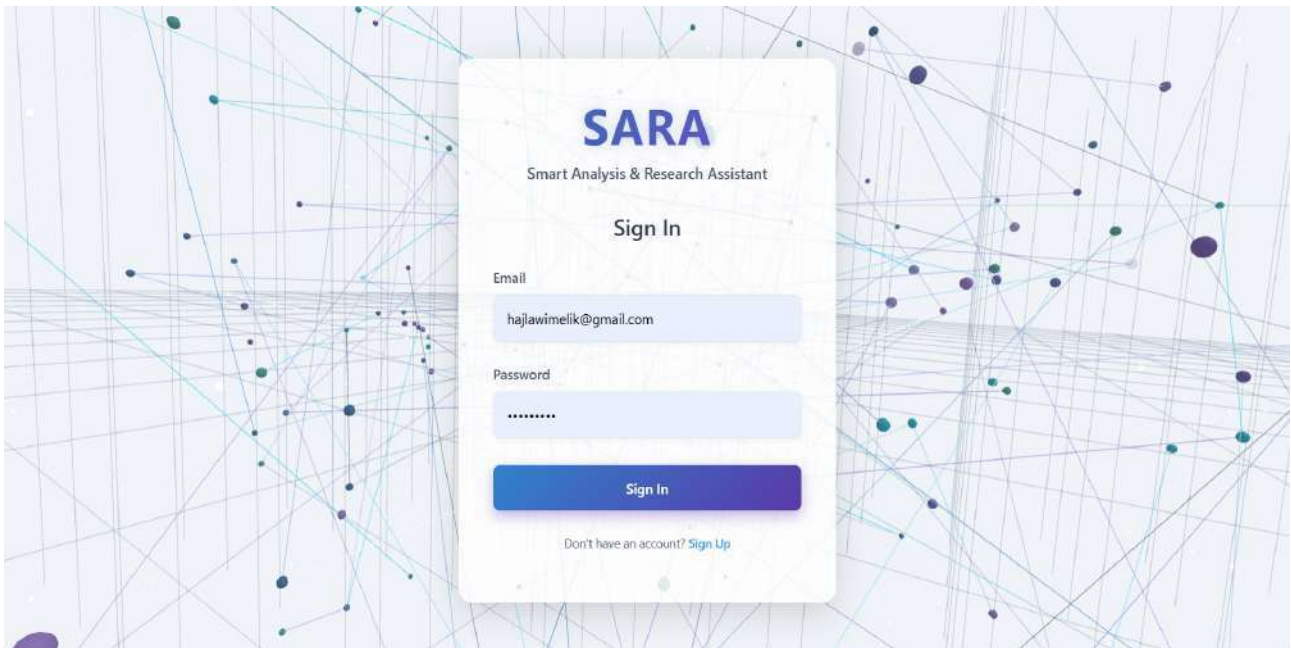


Figure 4.1: Login interface prompting for email and password.

Upon clicking *Sign In*, the system validates the credentials. On success, the user is redirected according to their role:

- **Administrator:** forwarded to the Admin Dashboard showcased in Section 4.8.1.
- **Researcher:** forwarded to the real-time monitoring landing page shown in Section 4.5.

If validation fails (e.g. incorrect password), a clear error message appears above the form.

### 4.4.2 Sign-Up Interface

Users without an account click the *Sign Up* link to open the registration form in Figure 4.2. The form requires:

- **Username:** chosen display name.
- **Email:** valid address for login and notifications.

- **Role:** selection between *Administrator* or *Researcher*.
- **Password:** at least 6 characters.
- **Confirm Password:** must match the password field.
- **Create Account** button to submit the registration.

Figure 4.2: Sign-Up interface for new user registration.

After submission, the system checks if the Email is unique. On successful validation, the new account is created.

## 4.5 Real-Time Monitoring Implementation

The real-time monitoring interface is designed to give researchers immediate insight into live sensor streams across multiple facilities, without requiring manual polling. As soon as new data becomes available, the frontend automatically receives a Server-Sent Event (SSE) from the Real-Time Monitoring Service and updates the relevant visualizations.

### 4.5.1 Interface Overview

The interface is organized as a tabbed dashboard as outlined in Figure 4.3, with each tab grouping related sensor types (see Table 1.2 for sensor details):

- **Occupancy:** As shown in Figure 4.3, the occupancy tab encompasses presence count, motion probability, microphone readings, BLE readings, and a  $32 \times 24$  thermal-map.
- **Environmental:** As displayed in Figure 4.4, the tab groups humidity, temperature and luminance.

- **Thermal:** As can be seen in Figure 4.5, the tab groups Temperature alongside the thermal-map visualization.

In the application bar at the top, a dropdown list of *Locations* allows the researcher to select a global context (e.g. “ISTIC A006” or “Aizu Cafeteria”). This global selection applies to all visuals by default, but each card also offers a local override.

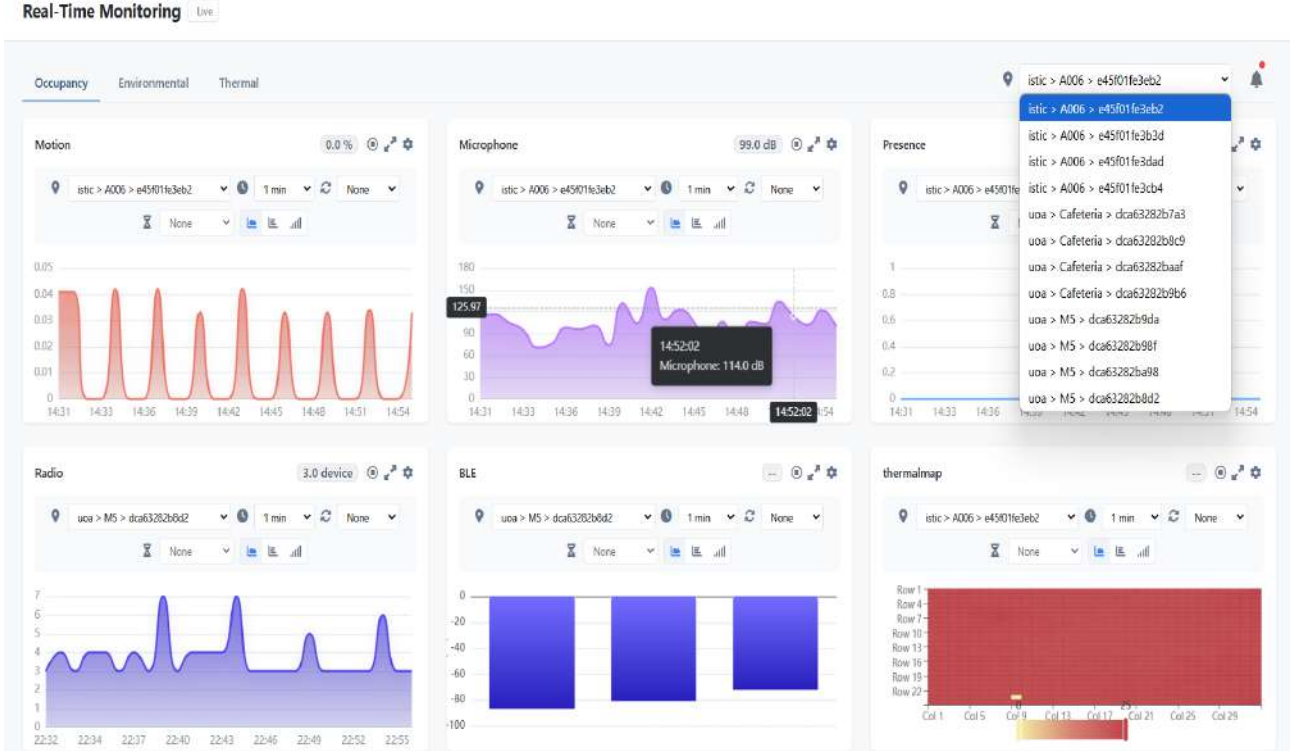


Figure 4.3: Real-Time Monitoring: “Occupancy” tab with live-updating cards.

### 4.5.2 User Controls

Each visualization card includes a control panel as shown in Figure 4.6 allowing the researcher to:

- **Change Location:** Override the global location for this specific stream.
- **Select Metric:** Choose among *min*, *max*, *sum*, *count*, or *average*.
- **Set Time Window:** Define a rolling interval (e.g. 3 min, 10 min) over which the metric is computed.
- **Adjust Update Frequency:** Modify the SSE polling interval (default: 60 s).
- **Pause/Resume:** Temporarily halt live updates for closer inspection.
- **Expand/Collapse:** Expand the chart to full width for detailed examination.
- **Switch Chart Type:** Toggle between *area*, *line*, and *bar* representations.





Figure 4.4: Real-Time Monitoring: “Environmental” tab with luminance chart expanded and humidity paused.

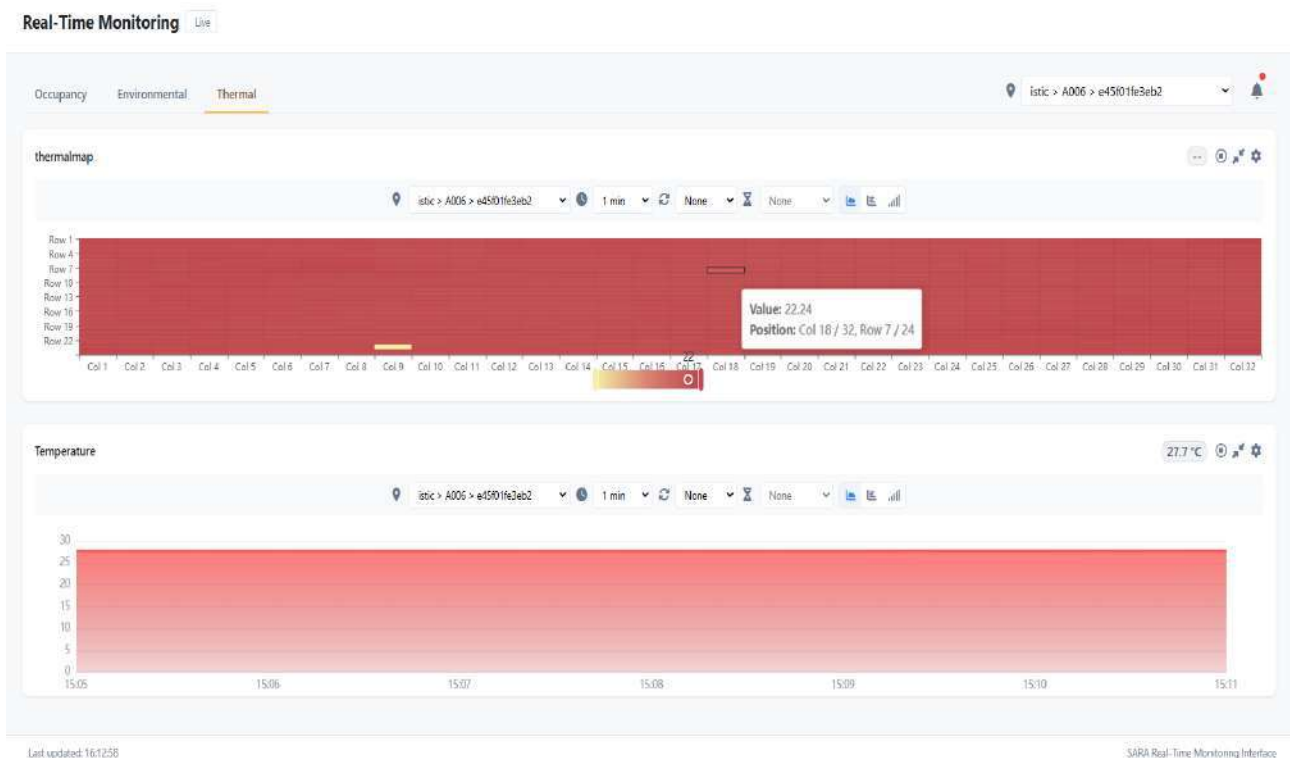


Figure 4.5: Real-Time Monitoring: “Thermal” tab with thermalmap and temperature charts both expanded.

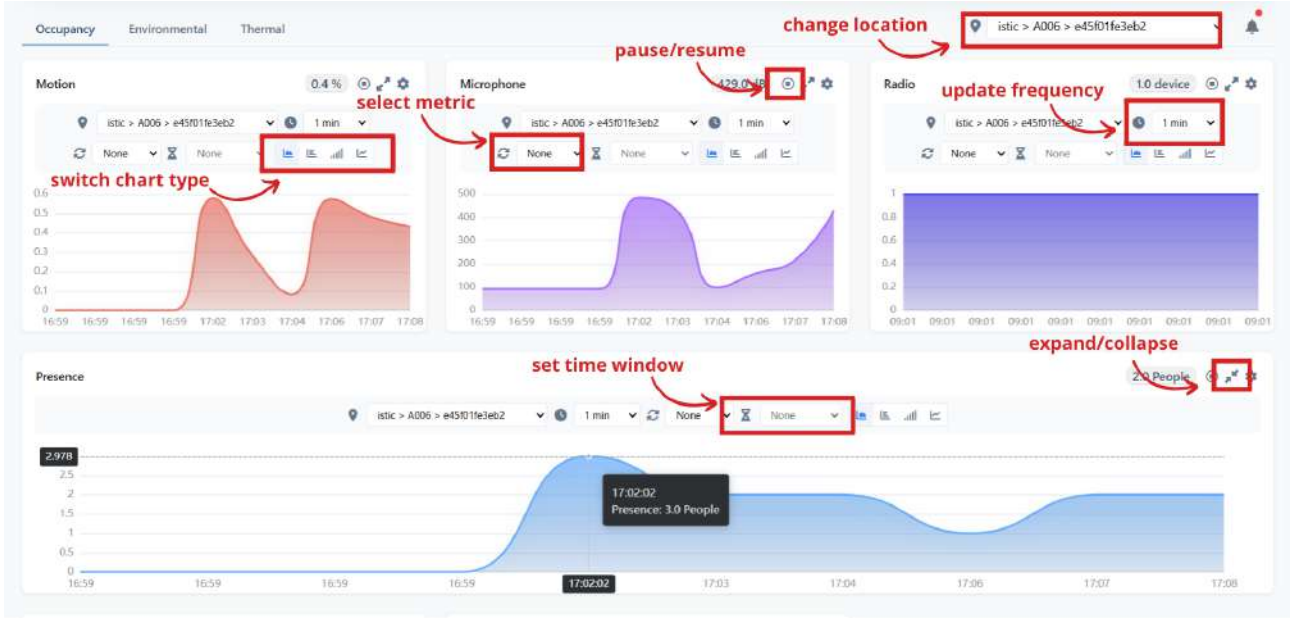


Figure 4.6: Control panel for an individual sensor stream.

### 4.5.3 Visualization Example: BLE and Radio Streams

Consider the *BLE Signal Detection* and *Radio Device Count* streams for the Aizu University classroom M5, displayed side by side as shown in the Figure 4.7. Both cards have been expanded to full width to facilitate comparison.

- The global facility is set to “Aizu University Classroom M5.”
- No additional metric transformations or frequency adjustments have been applied—the visuals reflect raw, per-interval data.
- The *Radio Device Count* card (top) shows the number of radio devices detected in each 60-second interval.
- The *BLE Signal Detection* card (bottom) displays the corresponding BLE readings (signal strength) for each device, identified by an IMEI, within the same time bins.

This configuration allows the researcher to observe, within the same temporal window, how many radio-broadcasting devices are present and to correlate each radio appearance count with the detected device’s BLE signal, enabling a direct comparison of device visibility across the two modalities.

## 4.6 Historical Data Visualization Implementation

The Historical Data Visualization interface enables researchers to explore archived sensor streams and custom datasets through three complementary views:

- *General Trends*,
- *Same-Type Comparison*,
- *Cross-Type Comparison*.



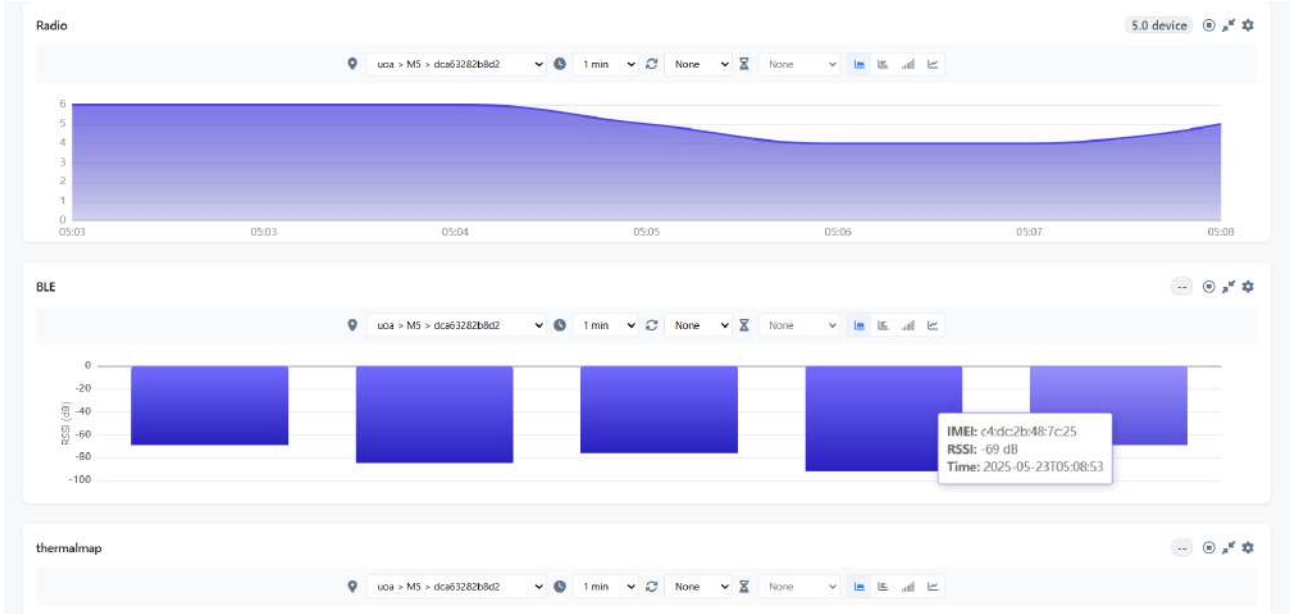


Figure 4.7: Radio Device Count (top) and BLE Signal Strength Detection (bottom) in Aizu classroom M5—raw counts and events per 60 s interval, aligned by time and device IMEI.

All three views maintain a consistent card layout and control panels. Figure 4.8 highlights the main regions of the interface with numbered bounding boxes.

#### 4.6.1 Data Sources Panel

In box (1) of Figure 4.8, the *Data Sources* accordion toggles between *Sensor Data Mode* and *Custom Dataset Mode*. In **Sensor mode**, the hierarchy adapts to the selected view:

**General Trends** (SensorType-centric):

SensorType → Facility → Location → SensorBox.

**Same-Type Comparison** (Location-centric):

Facility → Location → SensorType → SensorBox.

**Cross-Type Comparison** (Sensor-Type centric): each mini-view uses type-centric hierarchy as in General Trends.

This design minimizes user clicks by presenting the most relevant expansion order for each task. In **Custom Dataset mode**, users see datasets they own or collaborate on, with JSON attributes exposed as a tree for drag-and-drop as illustrated in box (1) of Figure 4.9.

#### 4.6.2 Configuration Panel

Box (2) in Figure 4.8 shows the *Configuration* panel. Researchers can:

- Select **Chart Type** (eg, area, line, bar, heatmap).
- Choose a **Statistical Metric** (mean, median, min, max, sum, count, skewness).

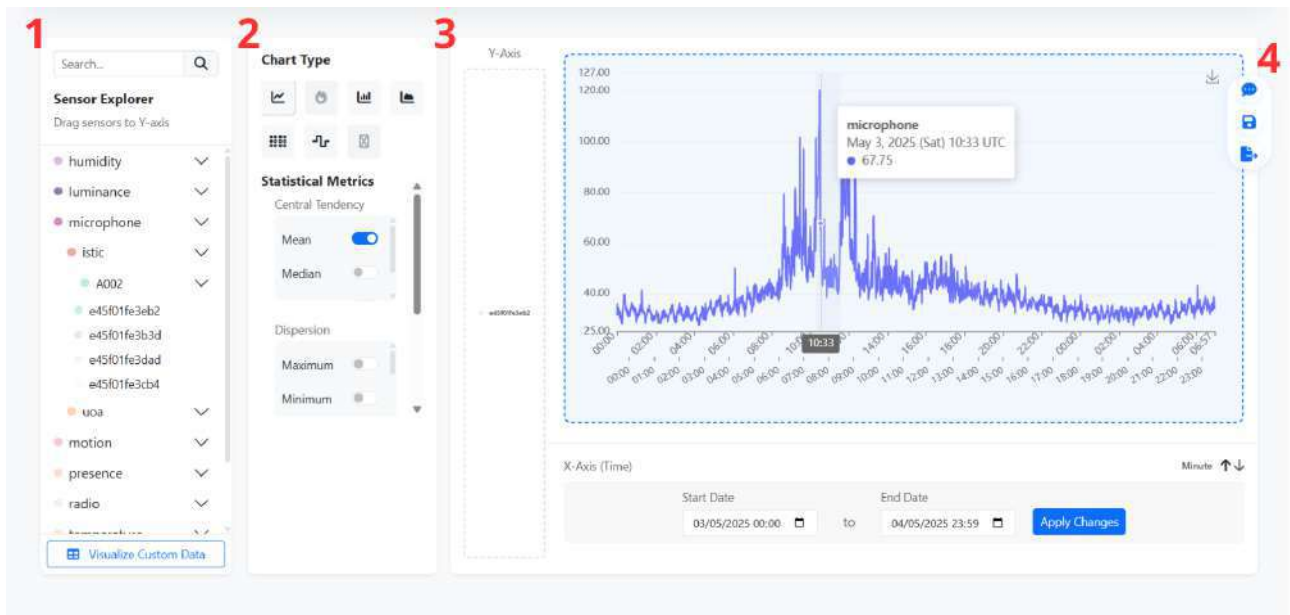


Figure 4.8: Annotated overview of the General Trends interface: (1) Data Sources panel, (2) Configuration panel, (3) Visualization card area showing minute microphone readings in classroom A006 between 2,3 May 2025, (4) Controls bar.

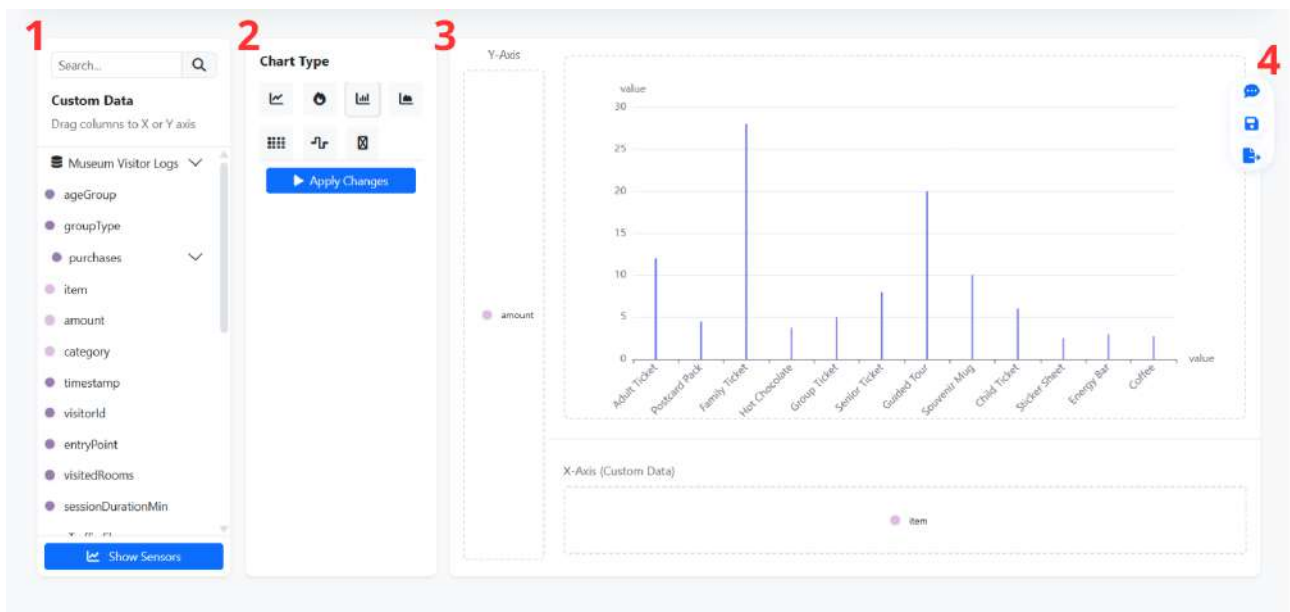


Figure 4.9: Annotated overview of the General Trends interface in custom-datasets mode.

### 4.6.3 Visualization Card Layout

The core visualization area appears in box (3) of Figure 4.8. Each card contains:

1. **Y-Axis Drop Target:** designate a drag-and-drop target for data sources (sensor stream or dataset attribute).
2. **Chart Canvas:** displays the resulting time series.
3. **X-Axis Panel:**
  - In Sensor mode: a date-range picker to set start/end times and an up/down arrows to adjust the **Resolution** (minute, hourly, daily, monthly, yearly) .
  - In Custom mode: a second drop target for the X-axis attribute as illustrated in box (2) of Figure 4.9.
4. **Notes & Export Controls:** A set of icons (shown in box (4) of Figure 4.8) allows researchers to annotate the visualization using a floating text area, capture and save a snapshot of the chart, and export the underlying data in either CSV or JSON format, as illustrated in Figure 4.11.

### 4.6.4 Example: Thermal Map Snapshot at Aizu Cafeteria

Figure 4.10 focuses on a single visualization card (box (4) in Figure 4.8) configured to display a thermal map of the Aizu University cafeteria on October 12, 2023 at 06:00 AM. In this example, the researcher:

1. Drags a **sensor box within the Cafeteria** onto the Y-axis drop zone.
2. Switches the chart type to the built-in *heatmap* renderer.
3. Opens the date-picker and selects October 12, 2023 at 6 am.
4. Sets the time picker to 06:00 AM and clicks “Apply Range” to fix the snapshot.
5. Observes the color-coded thermal grid, which spans from a minimum of 20 °C (blue) to a maximum of 26 °C (red), revealing localized temperature variations across the room.

This static thermal snapshot allows the researcher to pinpoint cooler and warmer zones within the cafeteria at an early-morning time, facilitating analyses of spatial temperature distributions and their potential correlation with occupancy

### 4.6.5 Same-Type Comparison

In the Same-Type Comparison view, box (3) subdivides to accept multiple Y-axis drops as showcased in Figure 4.12. Researchers place identical sensor types from different locations and share a common time axis for direct comparison.

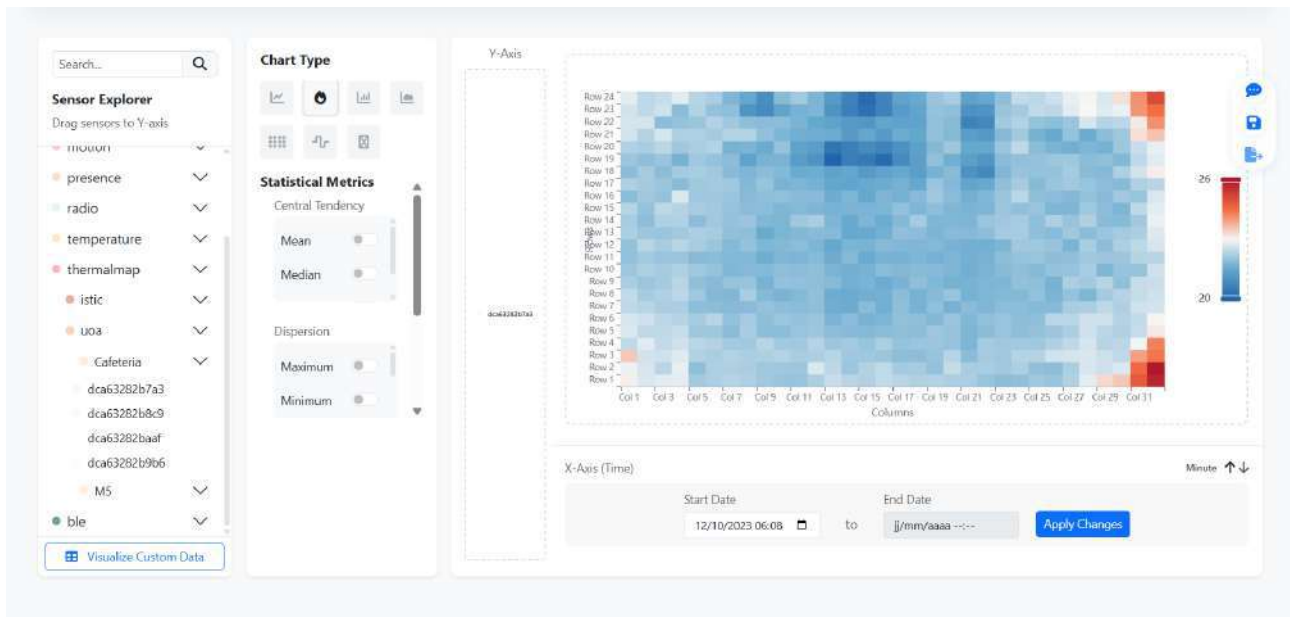


Figure 4.10: Thermal map of Aizu University cafeteria on October 12, 2023 at 06:00 AM (temperature range: 20–26 °C).

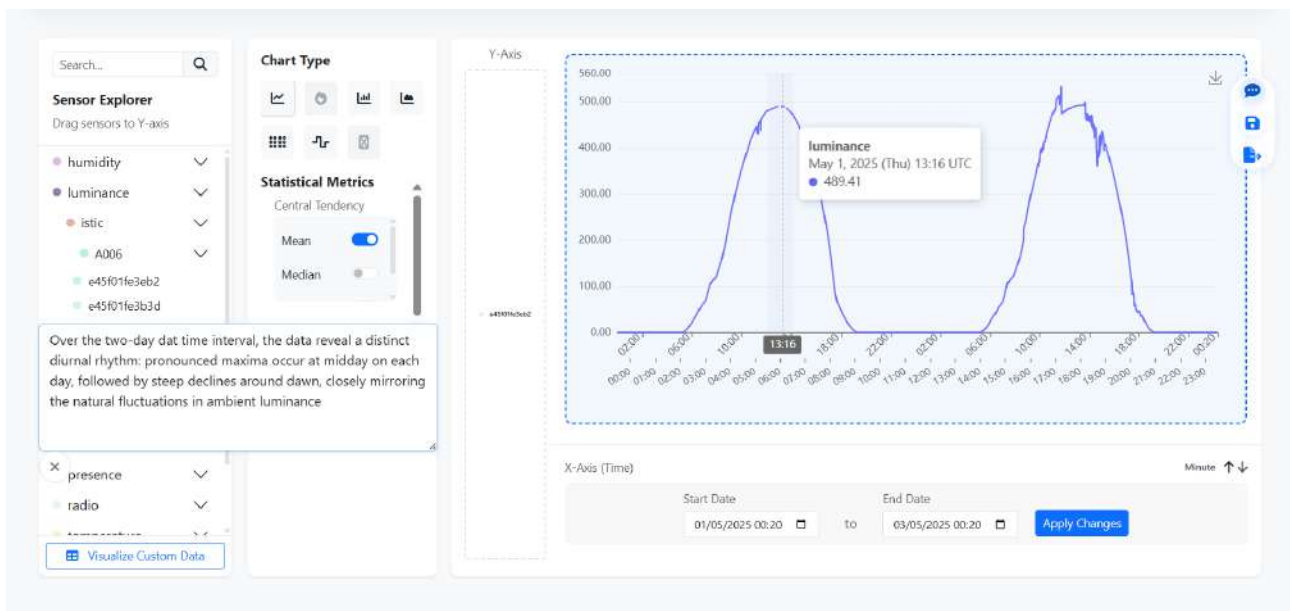


Figure 4.11: Hourly average luminance in ISTIC A006 for 1,3 May 2025.

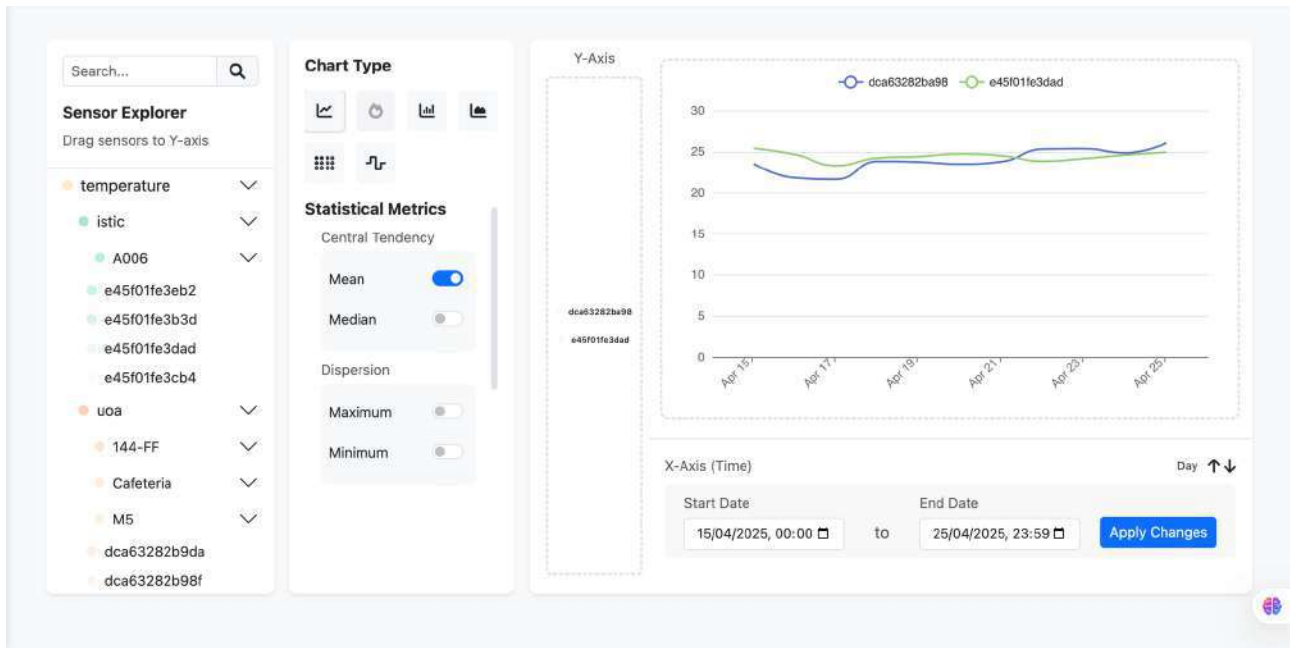


Figure 4.12: Same-Type Comparison: temperature streams from classroom A006 and classroom M5 over 5 days span.

#### 4.6.6 Cross-Type Comparison

The Cross-Type Comparison view arranges multiple mini-cards within, each with its own Y-axis (Figure 4.13). Users drop different sensor types into each sub-chart, and can rearrange or remove cards to tailor the layout.

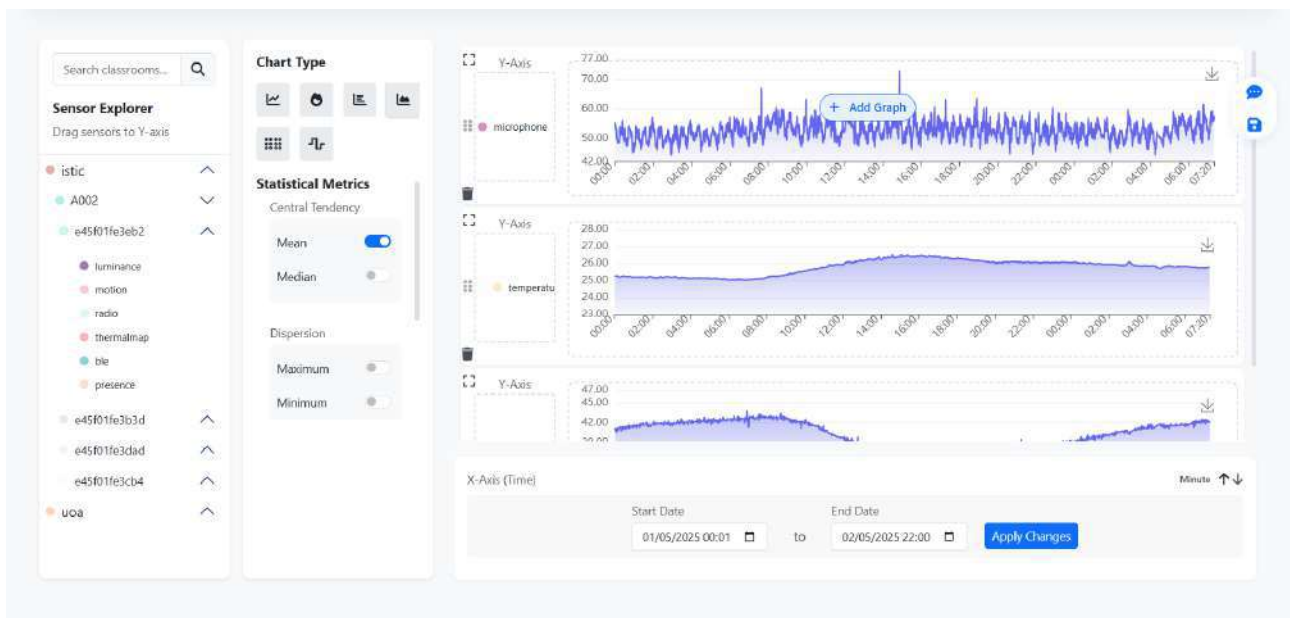


Figure 4.13: Cross-Type Comparison: individual sub-charts for humidity, motion, and temperature.

### 4.6.7 Data Export Feature

In the Controls bar (box (4) in Figure 4.8), researchers can export the currently displayed data within the selected time range or custom attribute selection. By clicking the **Export** icon, a modal prompts the user to choose between:

- **CSV:** comma-separated values preserving headers and rows for downstream analysis;
- **JSON:** an array of objects reflecting the axis bindings and computed metrics.

Upon confirmation, the interface fetches the dataset within the selected time range and initiates a download.

## 4.7 Custom Dataset Management Interface

Custom dataset management enables researchers to integrate their own data into SARA, define sharing policies, and view metadata—thereby extending historical visualization to domain-specific sources. This use case comprises two interfaces: the *Dataset Creation Wizard* and the *Dataset Management Dashboard*. Figures 4.14 and 4.17 illustrate their primary screens.

### 4.7.1 Dataset Creation Wizard

The creation wizard guides the user through three sequential steps, presented in Figure 4.14:

The screenshot displays the 'Create New Dataset' wizard. At the top, a progress bar shows three steps: 1. Data (active), 2. Schema, and 3. Preview. The main content area is titled 'Upload Your Data' and includes instructions: 'Upload a CSV file or paste JSON data to create your dataset.' Below this, there are two required fields: 'Dataset Name' (with a placeholder 'Enter dataset name') and 'Description' (with a placeholder 'Describe your dataset'). A large dashed box with an upload icon and text 'Drag & drop your CSV file here or click to browse files' is provided for file uploads. To the right, there is a section 'Or paste JSON data' with a 'Format JSON' button and a text area containing a JSON array: 

```
[{"timeStamp": "2023-04-01T12:00:00Z", "temperature": 22.5, "humidity": 45}]
```

. At the bottom, the 'Dataset Visibility' section has three radio buttons: 'Public' (selected), 'Private', and 'Restricted', each with a brief description of access levels. A 'Next: Define Schema' button is located at the bottom right.

Figure 4.14: Dataset Creation Wizard: (1) Data input, (2) Schema definition, (3) Preview & Create.

#### Step 1: Data Input

In tab (1) of Figure 4.14, the researcher supplies the data:

- **Upload Area:** Drag-and-drop or browse to select a JSON or CSV file.

- **Inline JSON:** Alternatively, paste a JSON array into the text area.
- **Visibility Radio:** Choose *Private*, *Public*, or *Restricted*.
- **Collaborator Input:** If *Restricted* is selected, an entry field appears for one or more email addresses.

Automatic validation ensures the uploaded or pasted data is well-formed before proceeding.

## Step 2: Schema Definition

As illustrated in Figure 4.15, the user defines the data schema:

- **Schema Text Area:** Enter a JSON Schema corresponding to the dataset.
- **Show errors:** Upon click, the schema is checked against the provided data.
- **Error Modal:** If mismatches occur, a pop-up lists the discrepancies for correction.

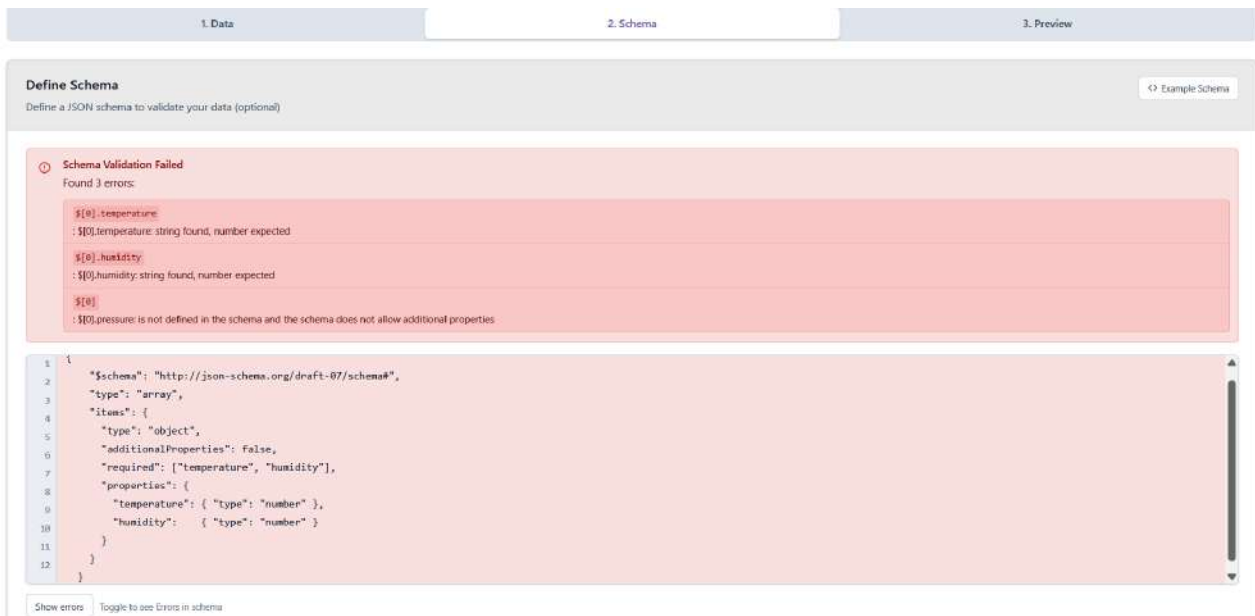


Figure 4.15: Dataset Creation Wizard, Schema step: schema input and *show errors* button

Only when the schema validates successfully can the user advance to the preview step.

## Step 3: Preview & Create

Figure 4.16 focuses on the preview tab ( box (3)), which offers a table preview of the first rows:

- **Table Preview:** Displays the first 10 rows and columns for sanity checking.
- **Create Dataset:** Commits the dataset; a confirmation message indicates success or failure.

Once created, the dataset appears under *Custom Dataset Mode* in the historical visualization interface (Section 4.6).



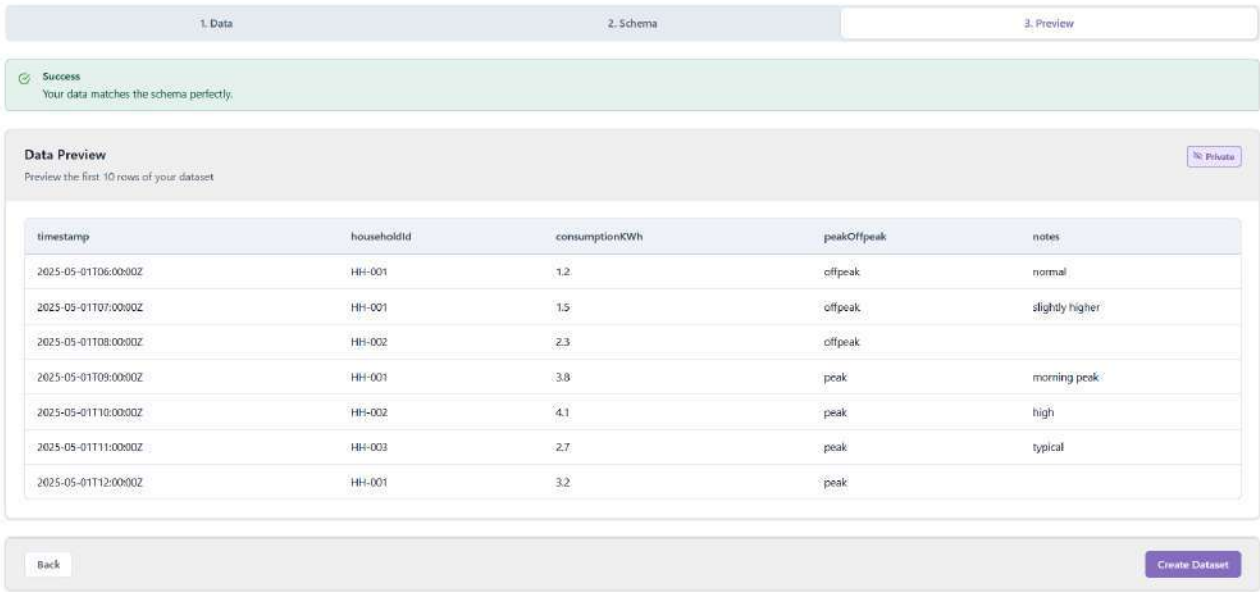


Figure 4.16: Dataset Creation Wizard, Preview step: table view and *Create* button.

### 4.7.2 Dataset Management Dashboard

The management dashboard in Figure 4.17 lists all datasets accessible to the user as cards summarizing:

- Dataset name, owner, visibility, and collaborator list (if any).
- Row and column counts, creation timestamp.
- Description and schema.

and provides owner-only actions:

Clicking a card opens the *Dataset Detail* view as illustrated in Figures 4.18 and 4.19, which displays all metadata fields including:

- **Delete:** Available only to the owner, removes the dataset after confirmation as demonstrated in Figure 4.20.
- **Edit Metadata:** Modify name, description, visibility, or collaborator list.
- **Schema Update:** Replace or refine the JSON Schema and re-validate.

Through these interfaces, the platform supports end-to-end custom dataset workflows—from ingestion and schema enforcement to metadata management and deletion—ensuring researchers maintain control over their data while seamlessly integrating it into the historical analysis pipeline.

## 4.8 Admin Reference Data Management

Reference data management is a critical administrative use case within the SARA platform. It allows administrators to centrally manage dynamic system configurations related to the sensor data network, supporting the flexible and scalable nature of research environments.



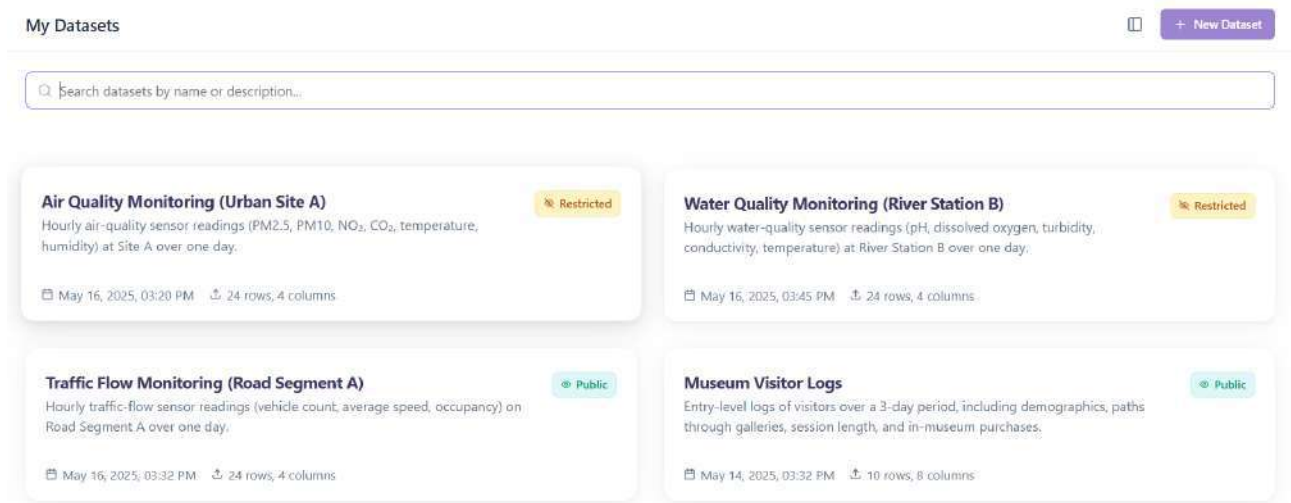


Figure 4.17: Dataset Management Dashboard: cards for each dataset with metadata.

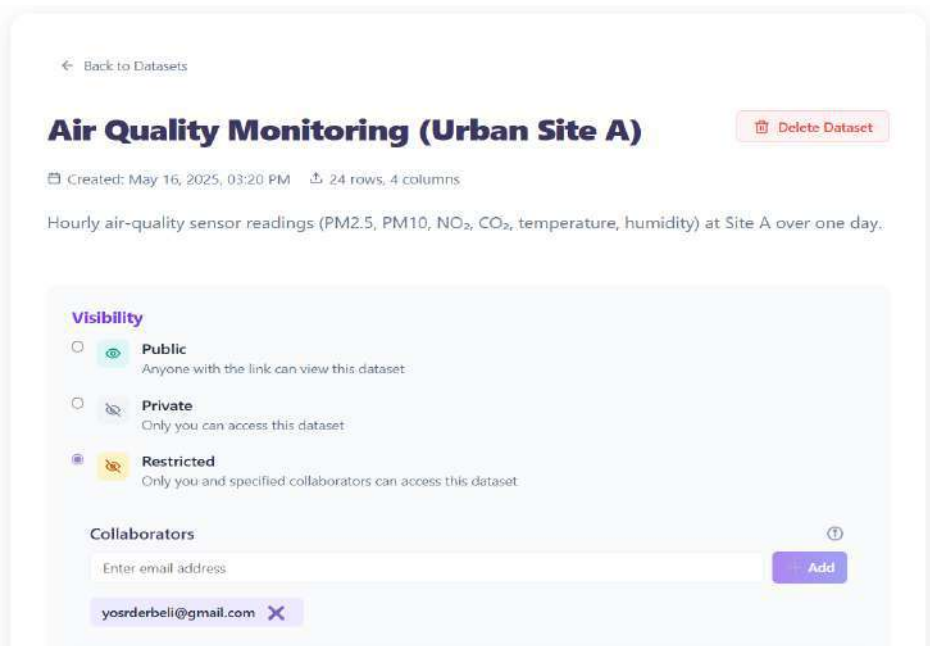


Figure 4.18: Dataset Detail view-2: metadata display and management actions.

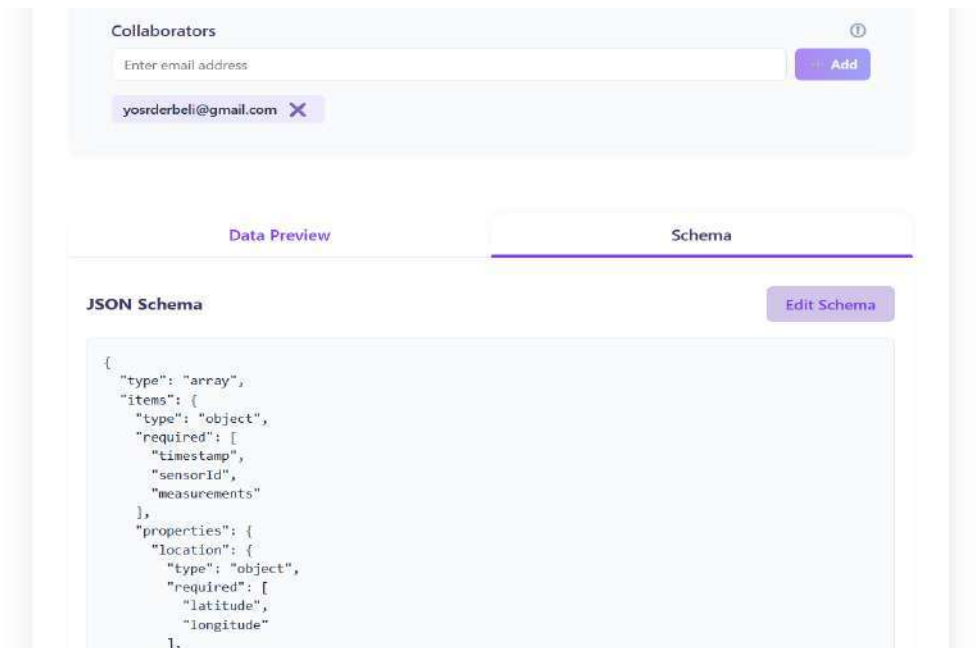


Figure 4.19: Dataset Detail view-2: metadata display and management actions.

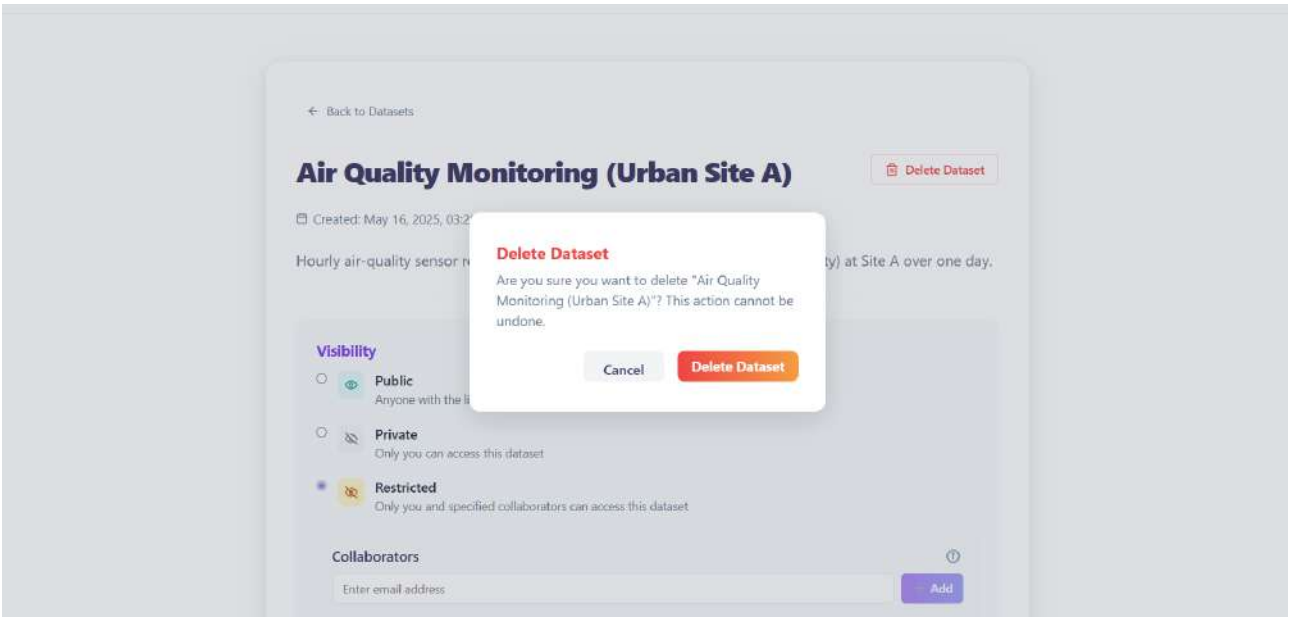


Figure 4.20: Dataset Detail view-3: deleting a dataset

### 4.8.1 Administrative Dashboard

When the administrator logs in, they are presented with a centralized dashboard shown in Figure 4.21. At the top of the page, summary cards display counts of total sensor boxes, sensor types, locations, and facilities. This provides a quick system overview and helps track growth over time.

Below the summary cards, a location-centric tree hierarchy is displayed using collapsible accordions, showing:

Facility → Location → Sensor Box (Agent Serial)

At the center of the interface is a table listing all known locations. Each row contains:

- **Facility:** the parent site that hosts the location,
- **Name:** the identifier of the location (e.g., A006).
- **Sensor Box Count:** the number of sensor boxes deployed at that location.
- **Display Color:** a color code used for identifying the location in visualizations.
- **Creation Timestamp:** used for auditability and tracking changes over time.
- **Actions:** buttons for *delete* and *update*.

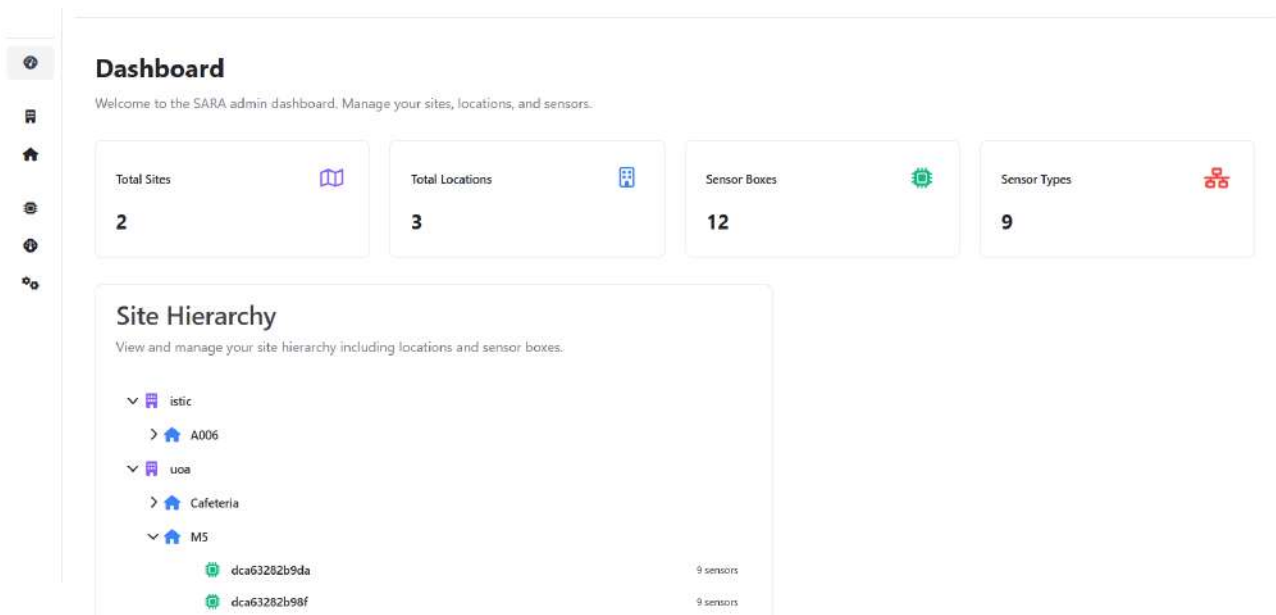


Figure 4.21: Admin dashboard showing summary statistics and location-centric hierarchy.

### 4.8.2 Location Management Interface

To illustrate how reference data is managed, we focus on the *Location Management Interface* (Figure 4.22). Similar interfaces are available for other reference entities.

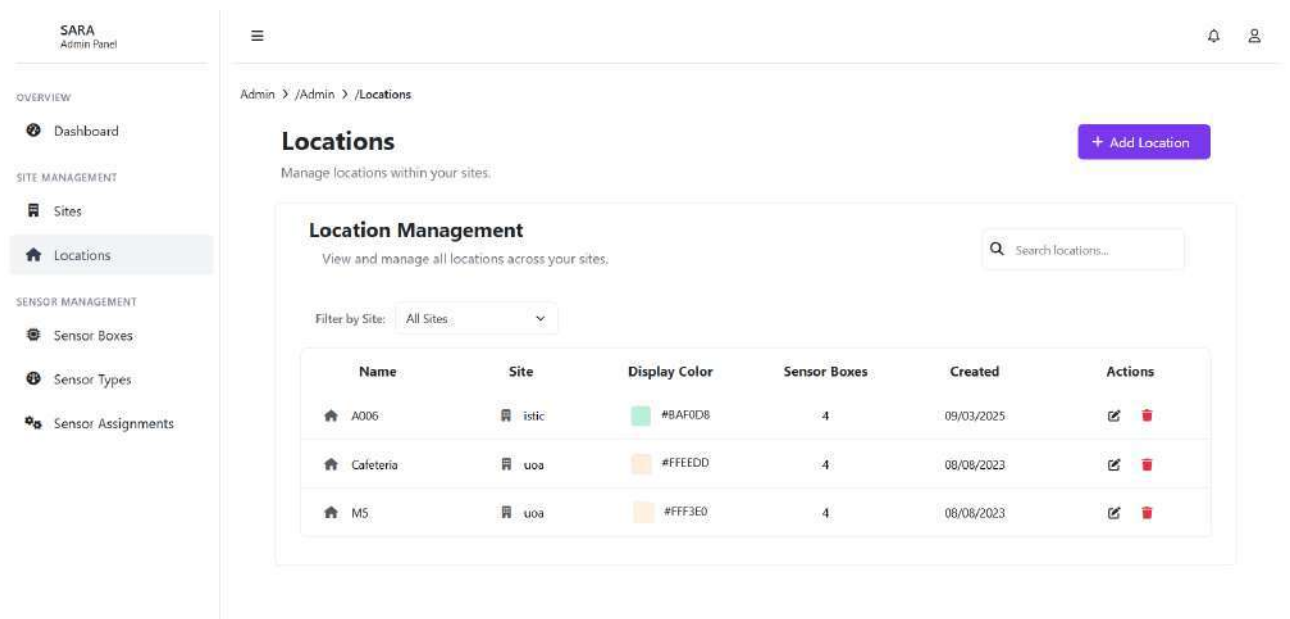


Figure 4.22: Location Management table showing editable entries and action controls.

### Update Workflow

Clicking the *Update* button opens a modal window, which can be seen in Figure 4.23, where the administrator can change the location’s name, associated facility, and display color. Upon submission, if the new configuration does not create a duplicate entry, the table is updated, and the changes are immediately reflected across all interfaces that use the location list dynamically.

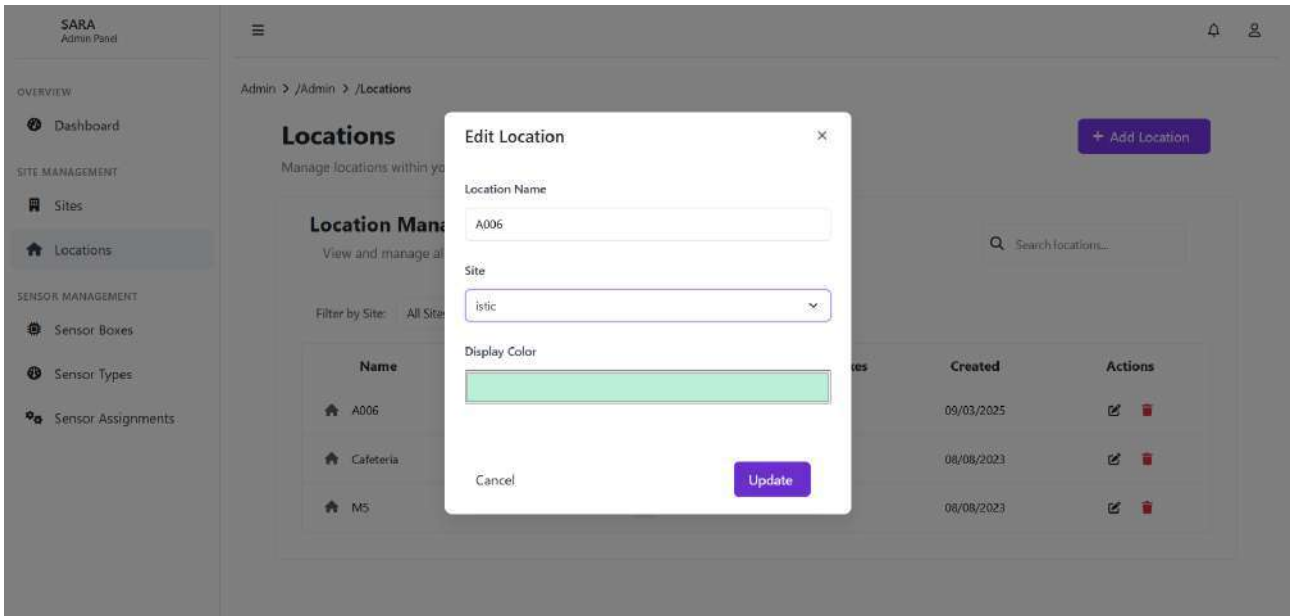


Figure 4.23: Location Management Interface: Update Modal Window

### Deletion Workflow

The *Delete* button prompts for confirmation before removing the entry. Once confirmed, the location is deleted from the database and automatically removed from dependent visualizations and configurations.

## Adding New Locations

At the top of the interface, an *Add Location* button opens a modal similar to the update window. The administrator provides the location name, selects a facility, and assigns a display color. After validation, the new location is added to the table and propagated system-wide, as can be seen in Figure 4.24.

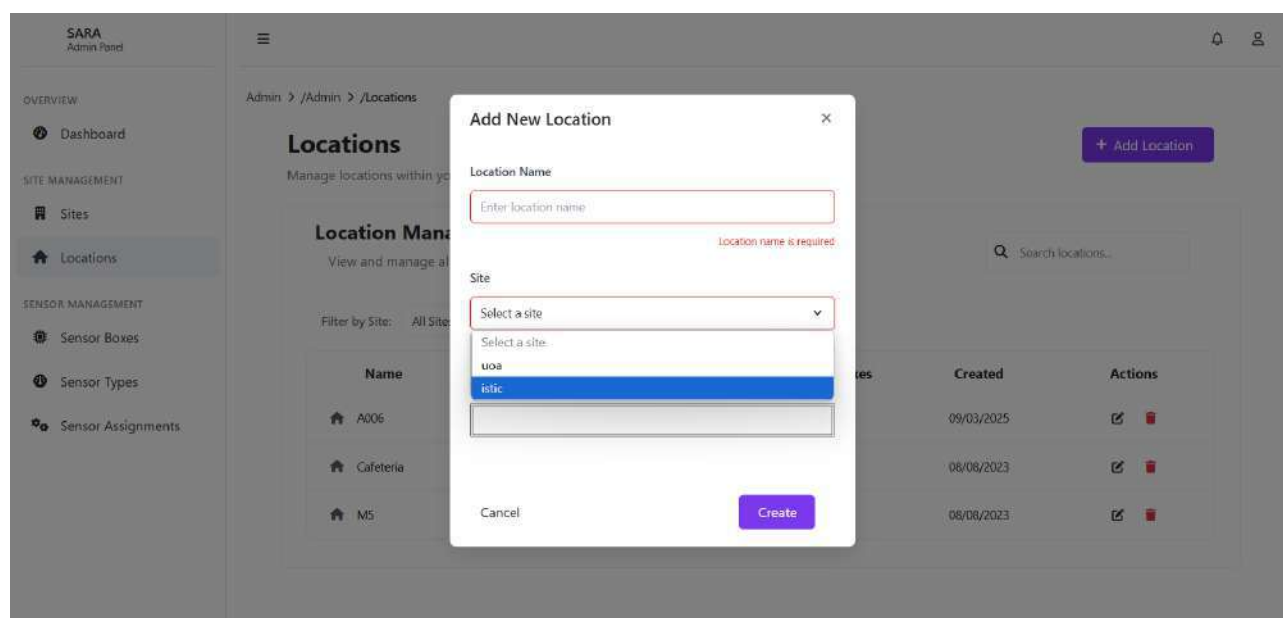


Figure 4.24: Add Location modal for defining new location metadata.

Through these interfaces, the system provides centralized and dynamic control over the reference data. This ensures that the platform remains aligned with the evolving structure of research environments, simplifies future integrations, and maintains consistency across all analytical and monitoring interfaces.

## 4.9 Conclusion

In this chapter, we have demonstrated how the SARA platform’s modular architecture (Chapter 3) translates into a working system. We detailed the development environment and selected technologies in Sections 4.2 and 4.3, then walked through each primary user interface and its implementation logic. From secure user authentication (Section 4.4) to dynamic real-time dashboards (Section 4.5), from flexible historical visualizations (Section 4.6) to end-to-end custom dataset workflows (Section 4.7), and finally administrative reference data controls (Section 4.8).

# General Conclusion

The SARA project began by situating itself within the Takai Daigaku collaboration, identifying the need for a unified, extensible research platform for smart environments. We then formulated precise objectives: to ingest heterogeneous data (live sensor streams, historical archives, custom uploads), enforce security and scalability constraints, and provide researcher-centric interfaces. A modular design followed, decomposing the system into discrete components—UI, API gateway, microservices, cache, and database—with well-defined interfaces to enable future extensions. Finally, we implemented the design using containerized FastAPI and SpringBoot services orchestrated by Angular and Apache ECharts on the frontend, supported by Redis for low-latency caching.

SARA has demonstrated its ability to:

- Authenticate users securely with short- and long-lived tokens across multiple devices.
- Deliver sub-second real-time dashboards via SSE and a hierarchical aggregation strategy for historical queries.
- Allow seamless integration of researcher-provided datasets alongside sensor feeds, with fine-grained visibility controls.
- Empower administrators to manage reference data dynamically, ensuring all visualizations reflect current configurations.

These achievements confirm that SARA meets its original objectives, offering a cohesive framework for data-driven exploration, analysis, and visualization in distributed research settings.

Looking ahead, the modular architecture of SARA opens numerous avenues for enhancement:

- **Integrating Pre-Trained Models:** Building on the *Model* (OSEMN framework) requirements, a logical extension would be to embed existing Embedding student-activity classifiers—such as Tasnime Ben Hamouda’s SVM framework [0] and Safa Meddeb’s Smart Behavior Estimation model [0]—enabling in-app training and prediction.
- **Expanded Statistical and Visualization Options:** Incorporating additional metrics (e.g. percentiles, entropy) and chart types (e.g. heatmaps, violin plots) would deepen analytical capabilities. Thanks to SARA’s plugin-style microservices, these features can be added without altering core components.
- **Reproducible Experimentation:** Automating experiment snapshotting—saving not only the chart image but the full configuration and raw data subset—would facilitate reproducibility and collaboration. A metadata store of experiments could allow researchers to revisit, share, and compare past analyses.

- **Domain Adaptation:** SARA's event-driven, plugin-style architecture readily extends to other IoT research domains by implementing new data-source microservices.

By bridging live sensor streams, historical archives, and custom datasets within a modular, microservice-oriented platform, SARA empowers researchers to explore, model, and interpret complex phenomena across distributed locations.

# Netography

- [1] *Angular*. <https://angular.io/>. Accessed: 2025-05-13. 2025.
- [2] *Apache ECharts*. <https://echarts.apache.org/>. Accessed: 2025-05-13. 2025.
- [3] Google Cloud. *Looker Pricing*. Accessed: 2025-06-13. 2025. URL: <https://cloud.google.com/looker/pricing>.
- [4] *Docker*. <https://www.docker.com>. Accessed: 2024-02-13.
- [5] *FastAPI: Python Web Framework*. <https://fastapi.tiangolo.com/>. Accessed: 2025-05-13. 2025.
- [6] Gartner, Inc. *Google Looker: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/google/product/looker>.
- [7] Gartner, Inc. *Microsoft Power BI: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/microsoft/product/microsoft-power-bi>.
- [8] Gartner, Inc. *Oracle Analytics Server: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/oracle/product/oracle-analytics-server>.
- [9] Gartner, Inc. *Qlik Sense: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/qlik/product/qlik-sense>.
- [10] Gartner, Inc. *Tableau: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/salesforce-tableau/product/tableau>.
- [11] Gartner, Inc. *ThoughtSpot Analytics: Reviews and Ratings*. Accessed: 04-03-2025. 2024. URL: <https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms/vendor/thoughtspot/product/thoughtspot-analytics>.
- [12] GeeksforGeeks. *Actors in UML*. Accessed: 25/02/2025. 2023. URL: <https://www.geeksforgeeks.org/unified-modeling-language-uml-actor/>.
- [13] GeeksforGeeks. *Functional vs Non-Functional Requirements*. Accessed: 25/02/2025. 2023. URL: <https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/>.
- [14] GeeksforGeeks. *Iterative Waterfall Model in Software Engineering*. Accessed: 25/02/2025. 2023. URL: <https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/>.
- [15] GeeksforGeeks. *Software Development Life Cycle (SDLC)*. Accessed: 25/02/2025. 2023. URL: <https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/>.



- 
- [16] GeeksforGeeks. *Use Case Diagrams*. Accessed: 25/02/2025. 2023. URL: <https://www.geeksforgeeks.org/use-case-diagram/>.
- [17] *GitHub*. <https://github.com>. Accessed: 2025-05-13. 2025.
- [18] IBM. *What is UML?* Accessed: 25-02-2025. 2023. URL: <https://www.ibm.com/docs/en/rational-soft-arch/9.7?topic=overview-what-is-uml>.
- [19] *InfluxDB*. <https://www.influxdata.com>. Accessed: 2024-02-13.
- [20] ISTIC. *The Higher Institute of Information and Communications Technologies (ISTIC)*. Accessed: 2025-02-23. 2011. URL: <https://www.istic.rnu.tn>.
- [21] Microsoft. *Power BI Pricing*. Accessed: 2025-06-13. 2025. URL: <https://powerbi.microsoft.com/en-us/pricing/>.
- [22] *Milvus: Open Source Vector Database*. <https://milvus.io>. Accessed: 2024-02-13.
- [23] *Monday.com*. <https://monday.com>. Accessed: 2025-05-13. 2025.
- [24] Oracle. *Oracle Analytics Cloud Pricing*. Accessed: 2025-06-13. 2025. URL: <https://www.oracle.com/analytics/cloud/pricing.html>.
- [25] OWASP Foundation. *OWASP Top 10: The Ten Most Critical Web Application Security Risks*. Accessed: 2025-05-10. 2021. URL: <https://owasp.org/Top10>.
- [26] *pgAdmin*. <https://www.pgadmin.org/>. Accessed: 2025-05-13. 2025.
- [27] Data Science PM. *OSEMN Framework*. Accessed: 2023-10-15. 2023. URL: <https://www.datascience-pm.com/osemn/>.
- [28] *PostgreSQL*. <https://www.postgresql.org/>. Accessed: 2024-02-13.
- [29] Qlik. *Qlik Sense Pricing*. Accessed: 2025-06-13. 2025. URL: <https://www.qlik.com/us/pricing>.
- [30] *RedisInsight*. <https://redis.com/redis-enterprise/redis-insight/>. Accessed: 2025-05-13. 2025.
- [31] ResearchGate. *Blockchain Enabled Smart Contract Based Applications: Deficiencies with the Software Development Life Cycle Models - Scientific Figure*. [https://www.researchgate.net/figure/SDLC-Iterative-Model-2\\_fig4\\_338710620](https://www.researchgate.net/figure/SDLC-Iterative-Model-2_fig4_338710620). [Accessed: 21 May 2025]. 2020.
- [32] Salesforce. *Tableau Pricing Plans*. Accessed: 2025-06-13. 2025. URL: <https://www.tableau.com/pricing>.
- [33] G. Software. *Google Looker*. visited on 25/02/2025. 2025. URL: <https://cloud.google.com/looker>.
- [34] M. Software. *Power BI*. visited on 25/02/2025. 2025. URL: <https://powerbi.microsoft.com/>.
- [35] O. Software. *Oracle Analytics*. visited on 25/02/2025. 2025. URL: <https://www.oracle.com/analytics/>.
- [36] Q. Software. *Qlik*. visited on 25/02/2025. 2025. URL: <https://www.qlik.com/>.
- [37] T. Software. *Tableau*. visited on 25/02/2025. 2025. URL: <https://www.tableau.com/>.
- [38] Th. Software. *ThoughtSpot*. visited on 25/02/2025. 2025. URL: <https://www.thoughtspot.com/>.
- [39] *Spring Boot*. <https://spring.io/projects/spring-boot>. Accessed: 2025-05-13. 2025.

- [40] *The University of Aizu*. <https://www.u-aizu.ac.jp>. Accessed: 2025-02-23. 1993.
- [41] ThoughtSpot. *ThoughtSpot Pricing Plans*. Accessed: 2025-06-13. 2025. URL: <https://www.thoughtspot.com/pricing>.
- [42] CX Today. *Gartner Magic Quadrant for Analytics and Business Intelligence (ABI) Platforms 2024*. Accessed: [2025-02-25]. 2024. URL: <https://www.cxtoday.com/customer-data-platform/gartner-magic-quadrant-for-analytics-and-business-intelligence-abi-platforms-2024/>.
- [43] University of Aizu. *Smart Museum Project: Objective Measurement of Visitor Behavior and Learning Experience Enhancement*. <https://u-aizu.ac.jp/research/researchlist/endetail?cd=90143>. [Accessed: 15-Oct-2023]. 2023.
- [44] *Visual Studio Code*. <https://code.visualstudio.com>. Accessed: 2024-02-13.

# Bibliography

- [I] Ben Hamouda, Tasnime. *Student Activity Classification Based on Multi-Model and Distributed Sensing*. Higher Institute of Information and Communication Technologies (ISTIC), University of Carthage, 2024. Mémoire de Licence. Accessed via internal repository.
- [II] Meddeb, Safa. *Smart Campus Analytics: Data Analysis and Knowledge Discovery for Student Behavior Estimation*. Higher Institute of Information and Communication Technologies (ISTIC), University of Carthage, 2024. Mémoire de Mastère. Accessed via internal repository.