IoT Room Selection Decision Support System

Architectural Design Report

BPINFOR-124 - Introduction to IoT
University of Luxembourg

Anthony Stassart Federico Newton Filip Zekonja

January 2026

Contents

1 Introduction

This report documents the architectural design choices made for our IoT Room Selection Decision Support System. The system helps users pick the best room in a university building based on real-time environmental sensor data and personal preferences, using the Analytic Hierarchy Process (AHP) as its core decision algorithm.

The system addresses the following high-level expectations:

1. Communication protocols for sensor data acquisition and transport
2. Database design for heterogeneous sensor and facility data
3. Decision criteria grounded in EU standards (EN 16798-1)
4. A multi-criteria decision algorithm (AHP)
5. REST API with Swagger documentation
6. End-user interface for room selection (UI1)
7. Admin monitoring dashboard (UI2)
8. Bonus: JWT-based authentication

2 Team Organization

2.1 Role Assignment

We divided responsibilities according to each member's strengths and the natural boundaries of the system architecture:

Member	Role	Scope
Anthony	Backend / Database	FastAPI, MongoDB, REST APIs
Federico	Algorithm / Data	AHP implementation, EU standards research, JWT
Filip	Frontend / UI	React interface, Grafana dashboards, UI, Hardware

Table 1: Team role assignment

While each member owned a vertical slice of the system, we designed integration points (e.g. API contracts, data formats) together. Code reviews and integration sessions happened weekly.

2.2 Milestones

The project spanned six weeks (December 2, 2025 to January 15, 2026), including a holiday break (December 20 to 31). We organized work into nine epics:

Epic	Owner	Timeline
Architecture & Research	All	Week 1
Communication Protocols	Anthony	Weeks 1–2
Database Setup	Anthony	Week 2
Decision Criteria Research	Federico	Weeks 1–2
AHP Algorithm	Federico	Weeks 3–4
REST API Development	Anthony	Weeks 3–4
End-User UI (UI1)	Filip	Weeks 4–5
Admin Dashboard (UI2)	Filip	Week 5
Hardware & Protocol choice	Filip	Week 4–6
Integration & Testing	All	Week 6

Table 2: Project milestones by epic

2.3 Gantt Chart

We maintained a live Gantt chart tracking all 37 tasks across the nine epics. The chart is deployed at `https://fede-oss.github.io/iot/` and driven by a structured `tasks.json` file versioned in the repository.

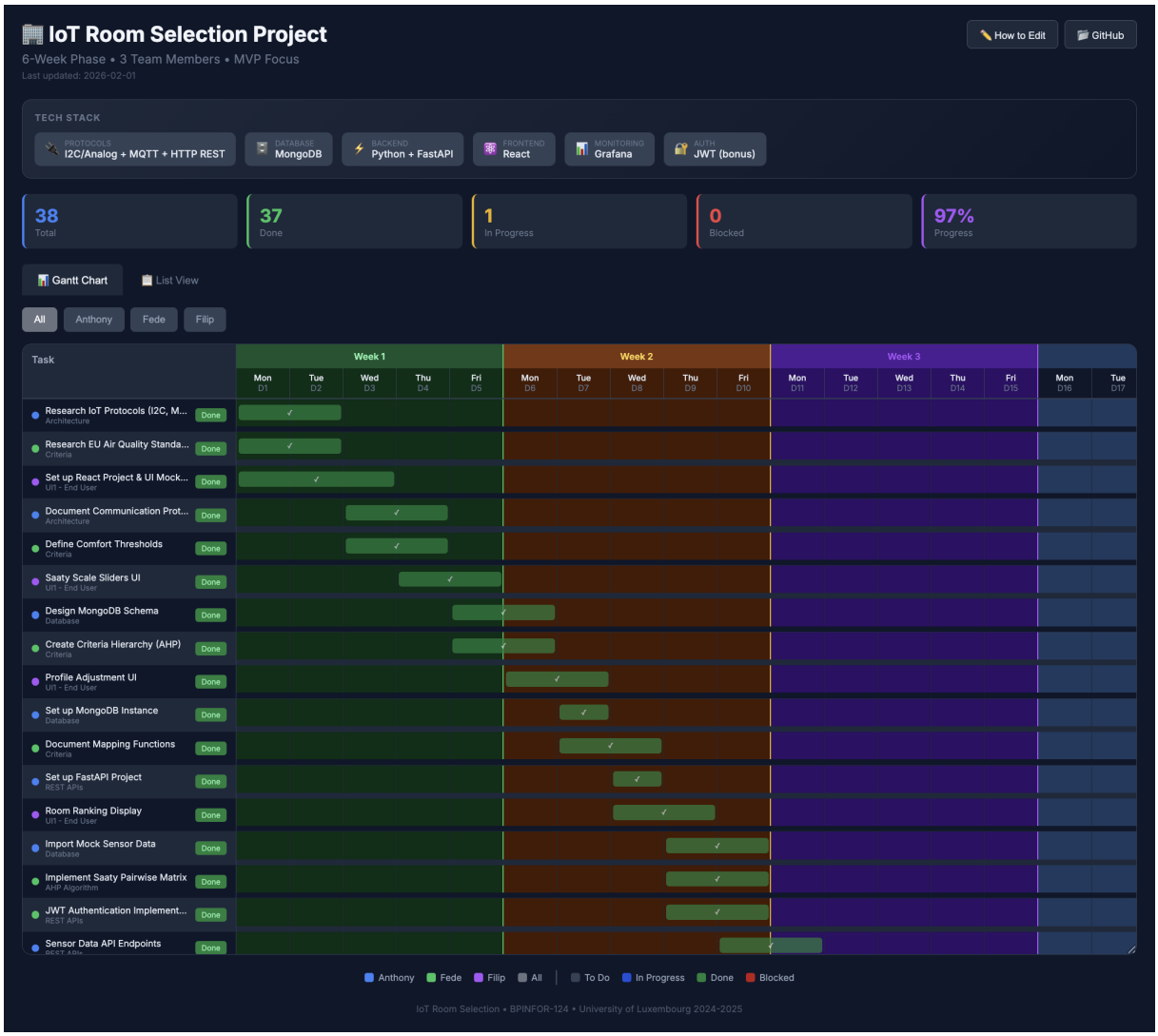


Figure 1: Project Gantt chart showing planned and actual progress

Most tasks were completed on schedule; the final task (this report) is the only one remaining (excluded for simplicity). The holiday break was accounted for in the initial plan, and no significant deviations from the timeline occurred.

3 System Architecture

3.1 High-Level Overview

The system follows a three-tier architecture with containerized deployment:

1. **Data Layer**: MongoDB 4.4 stores time-series sensor readings, room metadata, calendar events, and user accounts.

2. **Application Layer**: A FastAPI backend exposes REST endpoints, runs the AHP algorithm, and manages authentication.

3. **Presentation Layer**: A React 19 single-page application for end users, and Grafana dashboards for administrators.

All services are orchestrated via Docker Compose on a shared `iot-network`, enabling single-command deployment.

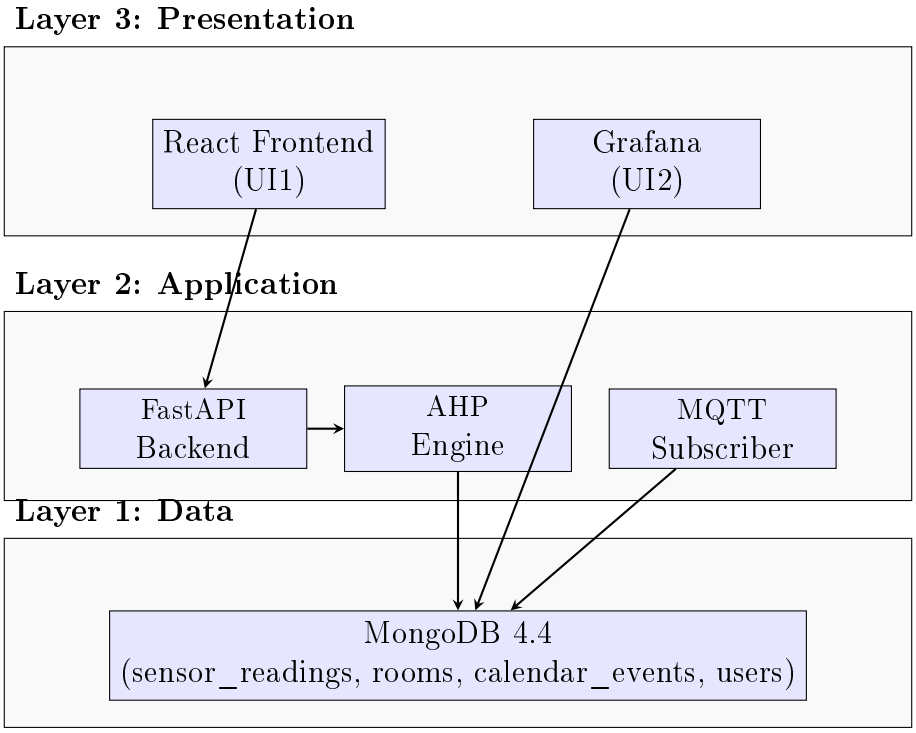


Figure 2: Three-tier system architecture

4 Communication Protocols

The project specification required four communication layers (Comm A through D). Our implementation maps them as follows:

Layer	Protocol	Direction	Justification
Comm A	I^2C / Analog	Sensor → MCU	Low-power, short-range bus for reading environmental sensors
Comm B	MQTT	MCU → Broker	Lightweight pub/sub ideal for constrained IoT devices
Comm C	HTTP REST	Broker → DB	Reliable ingestion into MongoDB via FastAPI endpoints
Comm D	HTTP REST	Client → API	Standard web API consumed by React frontend and Grafana

Table 3: Communication protocol mapping

Why MQTT for Comm B: MQTT's publish/subscribe model decouples sensor nodes from the backend. It uses minimal bandwidth (important for constrained MCUs), supports QoS levels for delivery guarantees, and is the de facto standard for IoT telemetry.

Why HTTP REST for Comm C/D: REST is a standard stateless protocol. Using HTTP for both ingestion and client access simplifies the stack and allows automatic Swagger documentation via FastAPI.

5 Database Design

5.1 Why MongoDB

We chose MongoDB for several reasons:

- **Schema flexibility**: Sensor readings vary by type (temperature has different fields than air quality). A document model accommodates this without schema migrations.
- **Time-series affinity**: MongoDB's document structure maps naturally to timestamped sensor readings, and compound indexes on `(room_name, sensor_type, timestamp)` enable efficient range queries.
- **Async driver**: The Motor async driver integrates natively with FastAPI's async request handling, avoiding thread-pool overhead.

5.2 Collections and Indexing

Collection	Purpose	Key Indexes
`sensor_readings`	Time-series environmental data	`(room_name, sensor_type, timestamp)`
`rooms`	Room metadata and facilities	`(name)` unique
`calendar_events`	Room bookings / availability	`(room_name, start_time, end_time)`
`users`	Authentication accounts	`(username)` unique

Table 4: MongoDB collections and primary indexes

Indexes are created programmatically on application startup in `database.py`, ensuring they exist regardless of deployment method.

6 Decision Criteria: EU EN 16798-1

Room ranking is grounded in the European standard EN 16798-1 for indoor environmental quality. We selected **Category II** thresholds (normal level of expectation for new and renovated buildings), which define optimal and acceptable ranges for each parameter:

Parameter	Optimal Range	Acceptable Range	Standard
Temperature	20–24 °C	18–26 °C	EN 16798-1
CO_2	≤ 800 ppm	≤ 1000 ppm	EN 16798-1
Humidity	40–60 %	30–70 %	EN 16798-1
Noise	< 35 dBA	< 45 dBA	WHO Guidelines
Lighting	300–500 lux	200–750 lux	EN 12464-1
VOC	< 200 ppb	< 400 ppb	WELL Standard
$PM_{2.5}$	$< 10\ \mu g/m^3$	$< 25\ \mu g/m^3$	WHO Guidelines
PM_{10}	$< 20\ \mu g/m^3$	$< 50\ \mu g/m^3$	WHO Guidelines

Table 5: Environmental thresholds used for score mapping

Each raw sensor value is normalized to a $[0, 1]$ score using piecewise linear mapping functions (implemented in `score_mapping.py`). Values within the optimal range receive a score of 1.0; values outside the acceptable range receive 0.0; intermediate values are linearly interpolated.

Note on facility scoring: Room facilities (seating, equipment, A/V) are not binary filters but contribute scores to the Usability criterion. Rooms with capacity significantly exceeding requirements receive minor penalties to discourage resource waste, while rooms meeting exact requirements receive optimal scores.

Occupancy Criterion (Bonus): Beyond the six core environmental criteria, the production system can also use room occupancy detection via Vision AI V2 modules (Section 8). The `map_occupancy()` function in Section 8 scores lower occupancy higher, favoring rooms conducive to focused individual work over crowded collaborative spaces.

7 AHP Algorithm

7.1 Why AHP

The Analytic Hierarchy Process (Saaty, 1980) was chosen because:

- It handles **multi-criteria** decisions with both quantitative (sensor data) and qualitative (user preferences) inputs.
- The **pairwise comparison** approach is intuitive for users - they compare criteria two at a time on a 1–9 scale rather than assigning abstract weights.
- Built-in **consistency checking** (CR < 0.1) ensures that user preferences are logically coherent.

7.2 Three-Level Hierarchy

Our AHP model uses a three-level criteria hierarchy:

1. **Level 1 – Main Criteria**: Comfort, Health, Usability
2. **Level 2 – Sub-Criteria**:
   - Comfort → Temperature, Lighting, Noise, Humidity
   - Health → CO_2, Air Quality (AQI), VOC, $PM_{2.5}$, PM_{10}
   - Usability → Seating Capacity, Equipment, AV Facilities
3. **Level 3 – Alternatives**: The candidate rooms (system evaluates 5 rooms in the prototype)

7.3 Implementation

The AHP engine is modular, split across five files in `backend/app/ahp/`:

`pairwise_matrix.py` Constructs and validates Saaty-scale pairwise comparison matrices. Enforces reciprocality ($a_{ji} = 1/a_{ij}$) and diagonal unity.

`eigenvector.py` Computes priority weight vectors using the principal eigenvector method. Also provides geometric mean and normalized sum as alternatives. Calculates the Consistency Ratio:

$$CR = \frac{CI}{RI}, \quad CI = \frac{\lambda_{\max} - n}{n - 1}$$

where RI is the Random Index for matrix size n.

`score_mapping.py` Normalizes raw sensor readings to $[0, 1]$ scores based on EU standard thresholds (Section 6). Implements mappings for all environmental sensors (temperature, CO_2, humidity, VOC, $PM_{2.5}$, PM_{10}, light, noise) and facility criteria (seating capacity, equipment, A/V facilities). Rooms exceeding capacity by more than 50% receive minor score reductions.

`aggregation.py` Combines weighted scores using a hybrid method: 70% Weighted Sum Model + 30% Weighted Product Model. This balances compensatory and non-compensatory ranking behavior.

`ahp_engine.py` Runs the full pipeline: loads user preferences, builds pairwise matrices, extracts weights, maps sensor data to scores, aggregates, and returns ranked rooms.

7.4 Consistency Checking

Every pairwise matrix submitted by users is validated against the standard AHP consistency threshold ($CR < 0.1$). If preferences are inconsistent, the system uses default expert-defined matrices as fallback, ensuring the ranking always produces meaningful results.

8 Hardware Architecture & Deployment

Each monitored room has a physical sensor node that collects environmental and occupancy data. Arduino-based nodes read sensors and publish readings over MQTT to a central Raspberry Pi, which hosts the MQTT broker, MongoDB, and FastAPI backend.

8.1 Platform Selection

Component	Model/Specification	Purpose
Coordinator	Raspberry Pi 4 Model B (4GB RAM)	MQTT broker host, Docker orchestration, backend services
Sensor Node	Arduino Uno R3 + Ethernet Shield 2 (W5500)	Environmental data acquisition & publishing
Occupancy Detection	Vision AI V2	Computer vision person counting
Network Fabric	Ethernet (Cat5e/Cat6 wired)	Sensor node → coordinator connectivity

Table 6: Platform components

All hardware components (Raspberry Pi 4, Arduino Uno, Ethernet Shield 2, Vision AI V2, and the sensor kit) were provided by the professor as the standard platform for this course.

8.2 Sensor Specifications & Wiring

Each sensor node monitors six environmental parameters aligned with EN 16798-1 standards (Section 6) plus local status feedback. The Arduino firmware samples sensors at intervals optimized for response time versus MQTT message frequency, publishing aggregated readings as JSON payloads.

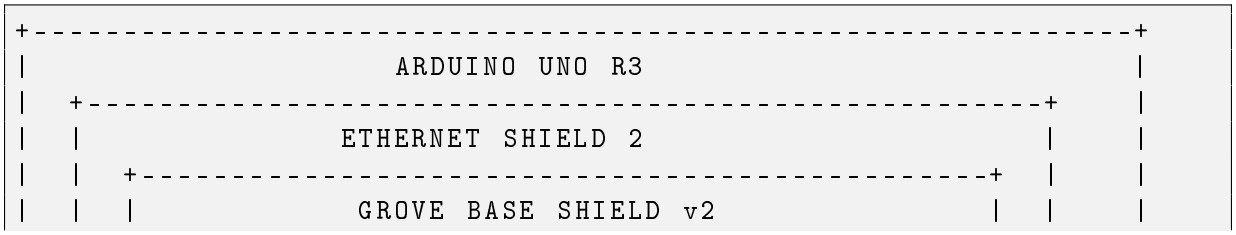
Sensor Type	Model/ Interface	Arduino Pin	Range	Sampling	AHP Criterion
Temperature/ Humidity	DHT11 Digital	GPIO D2	0 to 50°C, 20-90% RH	Every 5s	Comfort (thermal, hygroscopic)
Sound Level	Analog Microphone	Analog A0	30-130 dB	Every 100ms	Comfort (acoustic)
Light Intensity	Photoresistor LDR	Analog A1	0-1000 lux	Every 500ms	Comfort (visual, EN 12464-1)
Air Quality	Grove Air Quality Sensor v1.3	Analog A3	0-500 AQI	Every 5s	Health (air quality)
Status Display	Grove LCD RGB Backlight	I^2C (auto-detect 0x62)	16×2 characters	Event-driven	Local debugging feedback
Connectivity Status	Green LED	GPIO D4	Binary on/off	Continuous	MQTT connection / air quality indicator

Table 7: Sensor pinout and specifications

Communication protocols (I^2C for LCD, MQTT for telemetry) are detailed in Section 4. The Arduino firmware implements the Comm A (sensor → MCU analog/digital) and Comm B (MCU → broker MQTT) layers defined in Table 3.

Wiring Notes: The Grove Base Shield v2 stacks on top of the Ethernet Shield 2, providing convenient 4-pin connectors for all sensors. The DHT11 connects via the D2 Grove connector (white 4-pin cable), analog sensors use yellow 4-pin cables on A0, A1, and A3, and the LCD uses the I^2C Grove connector. The Ethernet Shield occupies SPI bus pins D10-D13, leaving sufficient GPIO and analog pins for the sensor array.

Wiring Diagram:



```
|   |   |                                                    |   |     |
|   |   |   [D2]  <- Temp&Humidity (white 4-pin cable)       |   |     |
|   |   |   [D4]  <- Green LED                               |   |     |
|   |   |                                                    |   |     |
|   |   |   [A0]  <- Sound Sensor (yellow 4-pin cable)       |   |     |
|   |   |   [A1]  <- Light Sensor (yellow 4-pin cable)       |   |     |
|   |   |   [A3]  <- Air Quality (yellow 4-pin cable)        |   |     |
|   |   |                                                    |   |     |
|   |   |   [I2C] <- LCD RGB Backlight (4-pin cable)         |   |     |
|   |   +----------------------------------------------------+   |     |
|   |                                                            |     |
|   |   [RJ45] -----> Router/Switch (same network as Pi)         |     |
|   +------------------------------------------------------------+     |
|                                                                      |
|   [USB] -----> Computer (for programming/power)                      |
|   [DC Jack] -> 9V power supply (alternative)                         |
+----------------------------------------------------------------------+
```

8.3 Vision AI Integration for Occupancy Detection

In addition to environmental sensors, we use Vision AI V2 modules for occupancy detection, which adds a seventh criterion to the AHP algorithm. Vision-based person counting is more accurate than PIR motion sensors, and the `map_occupancy()` scoring function (Section 7) to penalize overcrowded rooms in ranking results.

UART Connection:

```
Vision AI V2 -> Raspberry Pi 4 GPIO Header
-----------------------------------------
TX (White) -> GPIO 15 (Pin 10) - RXD
RX (Green) -> GPIO 14 (Pin 8)  - TXD
GND (Black) -> GND (Pin 6)
VCC (Red)  -> 3.3V (Pin 1) - Max 50mA draw

Configuration: 115200 baud, 8N1
(8 data bits, no parity, 1 stop bit)
```

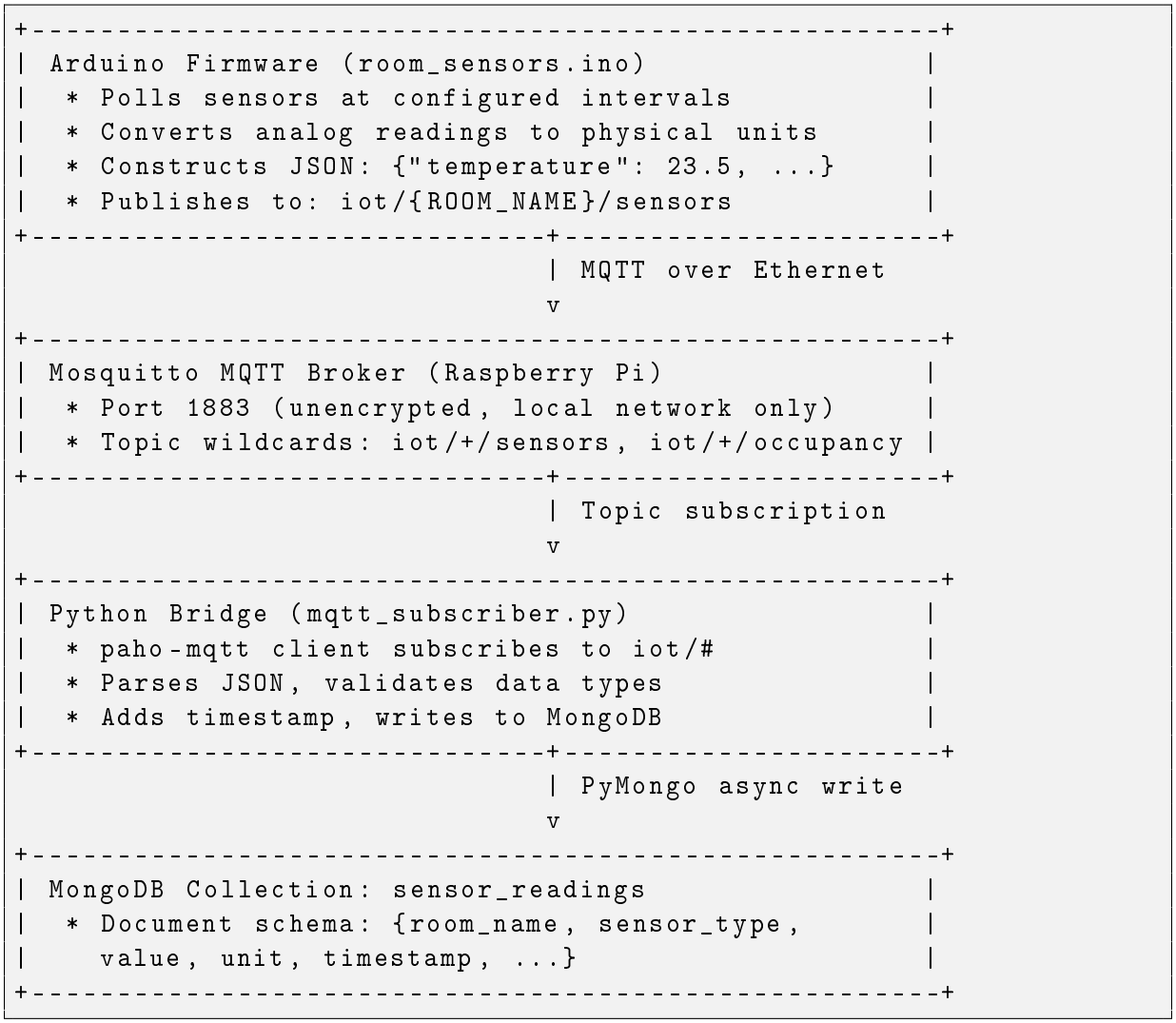
Data Flow: The Vision AI module processes video frames locally using a pre-trained person detection model, outputting integer person counts over UART at 1Hz. A Python daemon script (`vision_ai_reader.py`) running on the Raspberry Pi reads the serial stream via `/dev/ttyAMA0` and publishes counts to MQTT topic `iot/{ROOM_NAME}/occupancy`. The MQTT subscriber service writes occupancy readings to the same `sensor_readings` MongoDB collection with `sensor_type: 'occupancy'` and `unit: 'count'`.

Scoring Integration: The backend's AHP implementation includes `OCCUPANCY_CONFIG` in `score_mapping.py`, which scores lower occupancy higher (empty room = 1.0, overcapacity = 0.0-0.2). This favors emptier rooms for focused work and penalizes crowded ones.

8.4 Data Collection Architecture

The firmware-to-database pipeline uses MQTT's publish-subscribe model, so sensor nodes don't need to know about the backend. Each Arduino publishes JSON-formatted sensor

readings to room-specific topics – adding more rooms just means adding more publishers.



Sample MQTT Payload:

```
{
  "temperature": 23.5,
  "humidity": 55,
  "sound": 42,
  "light_intensity": 450,
  "air_quality": 35
}
```

Production Deployment: Both `mqtt_subscriber.py` and `vision_ai_reader.py` deploy as systemd services (`mqtt_subscriber.service`, `vision_ai.service`) on the Raspberry Pi, configured for automatic startup on boot. Services implement graceful signal handling (SIGTERM/SIGINT) for clean shutdown and MQTT disconnection, preventing message loss during restarts.

8.5 Power & Deployment Considerations

Component	Voltage	Current Draw	Power Supply	Notes
Arduino Uno + Ethernet Shield	5V DC	200-500mA (avg)	USB or 7-12V barrel jack	Peaks during Ethernet TX
Raspberry Pi 4 (4GB)	5V DC	600-1200mA (avg)	USB-C (5V/3A recommended)	Peaks during Docker container startup
Vision AI V2	3.3V DC	<50mA	Pi GPIO header (Pin 1)	Powered directly by Pi
DHT11 + Analog Sensors	3.3-5V DC	<20mA total	Arduino 5V rail	Negligible power draw

Table 8: Power requirements

Physical Installation: Sensor nodes mount in standard electrical junction boxes (single-gang) near room entrances, with Ethernet cable routed through conduit to centralized network switch. The Raspberry Pi coordinator resides in a server room or network closet alongside the Ethernet switch, minimizing cable runs. Vision AI modules mount at doorway height (1.5-2m) for optimal people counting accuracy.

Scalability Analysis: The current prototype instruments 5 rooms (Room_1 through Room_5) with a Bill of Materials cost of approximately $40 USD per room (Arduino $25, Ethernet Shield $10, sensors $5 total). The Mosquitto MQTT broker on Raspberry Pi 4 can handle many concurrent MQTT clients efficiently, so the system can scale to full building floors (20-50 rooms) without hardware changes. Vertical scaling to multiple buildings requires additional Raspberry Pi coordinators, each operating an independent MQTT broker with backend API endpoints federated via DNS load balancing.

9 REST API Design

9.1 Why FastAPI

- **Automatic OpenAPI/Swagger**: Every endpoint is self-documented with request/response schemas, satisfying the Swagger documentation requirement with zero extra effort.
- **Async-native**: Built on Starlette/ASGI, it handles concurrent sensor queries without blocking.
- **Pydantic validation**: Request bodies are validated against typed models at the framework level, meaning less repetitive code.
- **Lightweight**: Suitable for deployment on a Raspberry Pi 4.

9.2 Endpoint Structure

All endpoints are versioned under `/api/v1/`:

Method	Endpoint	Purpose
POST	`/api/v1/rank`	Room ranking (main DSS endpoint)
GET	`/api/v1/rooms/`	List rooms with facilities
GET	`/api/v1/sensors/{room}/{type}`	Sensor readings (time range)
GET	`/api/v1/sensors/{room}/latest`	Latest sensor snapshot
GET	`/api/v1/calendar/availability/{room}`	Room availability
POST	`/api/v1/auth/login`	JWT authentication
POST	`/api/v1/auth/register`	User registration
GET	`/health`	Health check (DB connectivity)

Table 9: REST API endpoints

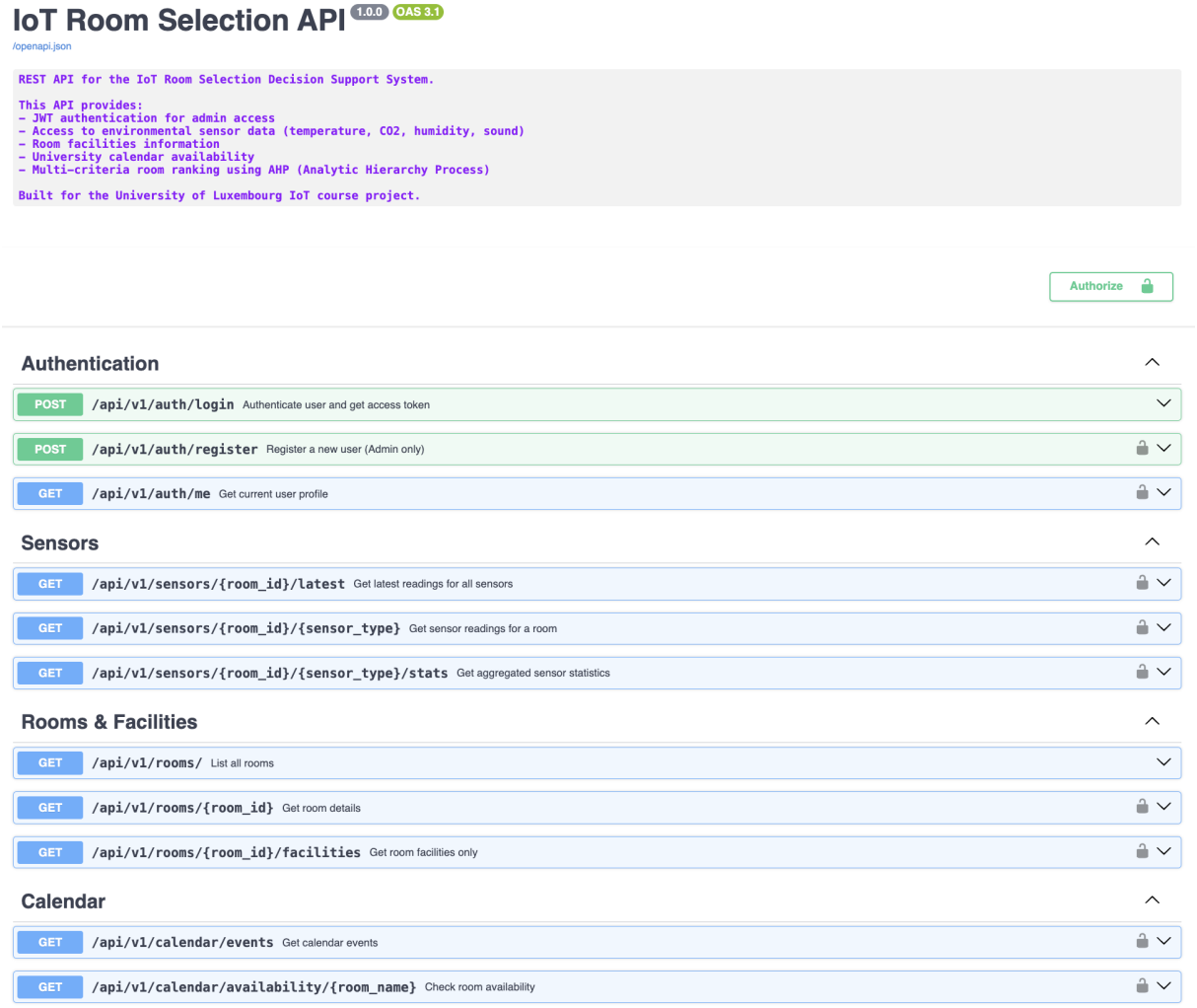


Figure 3: Swagger UI showing API documentation

10 Frontend – End-User Interface (UI1)

10.1 Technology Choices

- **React 19 + Vite 7**: Vite provides sub-second hot module replacement during development and optimized production builds. React's component model maps cleanly to our UI structure (preference input, ranking display, score breakdown).
- **Tailwind CSS**: Utility-first styling avoids context-switching between component logic and separate stylesheets.
- **Recharts**: Declarative charting library for visualizing score breakdowns.
- **Mock API fallback**: Setting `VITE_USE_MOCK_API=true` enables fully offline frontend development, decoupling UI work from backend availability.

10.2 User Flow

1. User navigates to the room selection page (`/select-room`).
2. The `PreferenceMatrix` component presents Saaty-scale sliders for pairwise comparisons between criteria (comfort, health, usability).
3. The `ProfileAdjuster` allows optional threshold overrides (e.g. minimum temperature).
4. On submission, preferences are transformed to backend format by `apiClient.js` and sent to `POST /api/v1/rank`.
5. Results are displayed via `RoomRanking` (ordered list) and `ScoreBreakdown` (per-criterion charts).

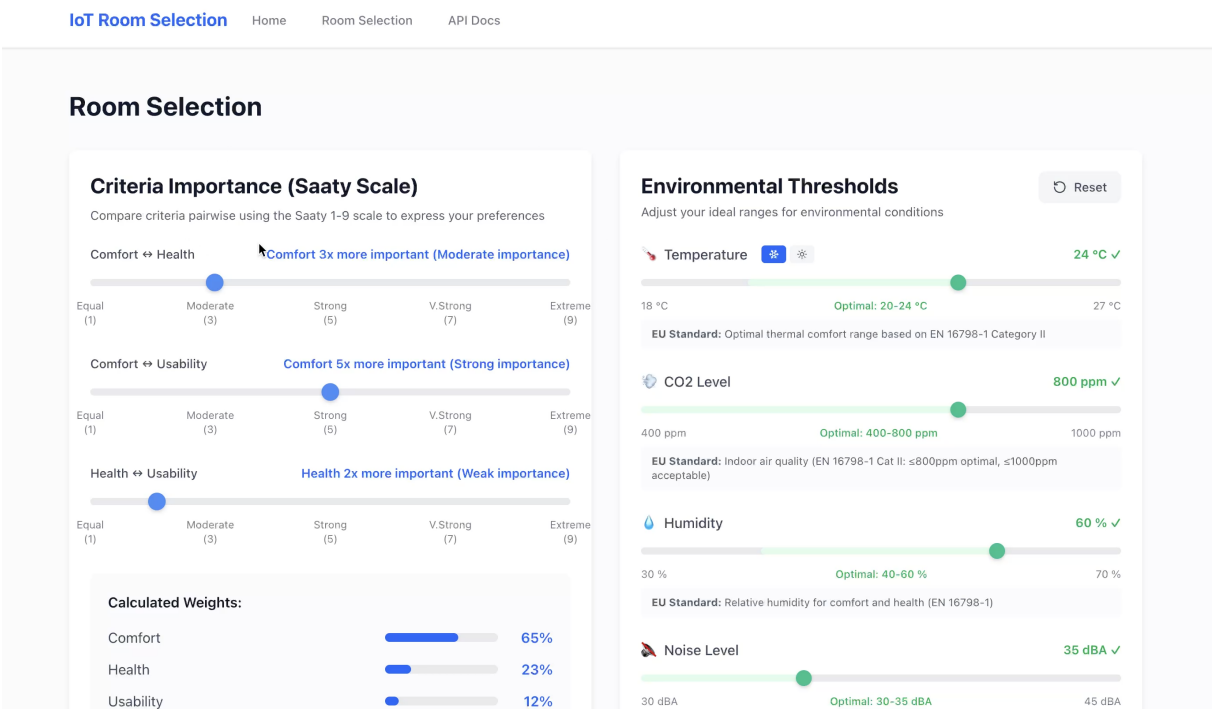


Figure 4: Room selection interface with preference input

11 Admin Dashboard (UI2) – Grafana

We chose Grafana for the admin monitoring dashboard because:

- It connects natively to MongoDB via plugins, requiring no custom dashboard code.
- It provides real-time auto-refreshing panels for sensor time-series data.
- Pre-built visualization types (gauges, time-series graphs, heatmaps) cover all monitoring needs.
- Alerting rules can be configured for threshold violations (e.g. CO_2 exceeding 1000 ppm).

Dashboard configurations are versioned in the `grafana/` directory and provisioned automatically via Docker Compose. The dashboard is accessible at `http://localhost:3000`.

Figure 5: Grafana admin dashboard showing real-time sensor data

12 Authentication (Bonus)

We implemented JWT-based authentication as a bonus feature:

- **Algorithm**: HS256 with a server-side secret key.
- **Token lifetime**: 30 minutes (configurable via `ACCESS_TOKEN_EXPIRE_MINUTES`).
- **Password storage**: Bcrypt hashing with automatic salt generation.
- **Frontend integration**: Tokens are stored in `localStorage` and attached to requests via an Axios interceptor.
- **Password policy**: Minimum 8 characters, at least one uppercase letter, one lowercase letter, and one digit.

This secures the ranking endpoint and and would let us save per-user preferences in the future.

13 Containerization and Deployment

13.1 Docker Compose Architecture

All services are defined in a single `docker-compose.yml`:

Service	Image	Port	Notes
`mongodb`	mongo:4.4	27017	Persistent volume, health check
`backend`	Custom (Python 3.11)	8000	Waits for MongoDB health
`mongo-express`	mongo-express:latest	8081	Dev profile only

Table 10: Docker Compose services

Design decisions:

- Services communicate over a private `iot-network` bridge, isolating traffic from the host.
- The backend uses `depends_on` with a health check condition, ensuring MongoDB is ready before accepting requests.
- The backend Dockerfile uses a multi-stage build optimized for ARM (Raspberry Pi 4 deployment) and runs as a non-root user for security.
- Mongo Express is gated behind a `dev` profile (`docker-compose -profile dev up`) to avoid exposing it in production.

14 Testing

14.1 AHP Unit Tests

The `unit/` directory contains five test modules covering each AHP component:

- `test_pairwise_matrix.py` – Matrix construction, reciprocality, validation
- `test_eigenvector.py` – Weight extraction, consistency ratio calculation
- `test_score_mapping.py` – Sensor value normalization against EU thresholds
- `test_aggregation.py` – WSM, WPM, and combined aggregation
- `test_ahp_engine.py` – End-to-end ranking pipeline

14.2 Backend Integration Tests

The `backend/tests/` directory contains API-level tests using `pytest` and `httpx`, covering JWT authentication flows and endpoint response validation.

15 Conclusion

Our IoT Room Selection DSS covers all eight project requirements with a straightforward design: sensor data flows through standardized communication protocols into MongoDB, the AHP algorithm produces clear rankings grounded in EU standards, and two distinct interfaces serve end users and administrators. Docker containerization ensures reproducible deployment across development machines and the target Raspberry Pi hardware.