1. Description

NRG CHAMP

Responsive Goal-driven Cooling and Heating Automated Monitoring Platform

Cantarini Andrea, Cantone Giuseppe Valerio

1. Description

NRG CHAMP is an advanced system designed for the efficient management of HVAC systems in large corporate and public buildings, with the primary goal of reducing energy consumption and promoting sustainable behaviors.

1.1. Key Features

- **Data Collection and Environmental Monitoring:**
  A dense network of IoT sensors continuously gathers essential parameters such as temperature and humidity, providing a precise and constant overview of both indoor and outdoor conditions.

- **Dynamic Control via the MAPE Cycle:**
  Utilizing the Monitor, Analyze, Plan, Execute (MAPE) framework, the system adjusts the operation of HVAC units in real time, ensuring optimized performance and reduced energy use.

- **Blockchain-Enabled Traceability and Transparency:**
  All collected data is securely recorded on a blockchain. This immutable ledger not only enables external regulatory bodies to verify the energy footprint of the building but also ensures transparency in reporting and compliance.

- **Data Analysis and Gamification:**
  The platform processes the blockchain-stored data to generate performance scores. These scores power a gamification mechanism that incentivizes users to achieve energy-conscious targets. Competitions can be structured hierarchically — for instance, comparing different offices within the same building, floors, or even among various organizations — ensuring that every participant has the opportunity to excel, regardless of their initial ranking.

1.2. Scalability and Adaptability

While the solution is primarily developed for large-scale, B2B deployments, its design is inherently scalable. With appropriate modifications, NRG CHAMP can be adapted to smaller settings such as apartment complexes or individual residences.

1.3. Summary

In summary, NRG CHAMP provides a holistic and innovative approach to energy management in buildings, combining real-time monitoring, intelligent control, blockchain traceability, and gamification to drive sustainable energy practices.

2. Software Requirements

2. Software Requirements

2.1 Functional Requirements

2.1.1. Data Collection and Environmental Monitoring

- **Sensor Integration:**
  - Support for a wide range of IoT sensors to monitor temperature, humidity, energy consumption, and other relevant environmental parameters (e.g., CO_2 levels, occupancy).
  - Real-time data acquisition from indoor and outdoor sensor networks.
- **Data Aggregation:**
  - Collection and aggregation of sensor data for further analysis.
  - Support for data buffering and recovery in case of temporary network failures.

2.1.2. Dynamic HVAC Control via MAPE Cycle

- **Monitoring Module:**
  - Continuously monitor real-time sensor data and energy consumption metrics.
- **Analysis Module:**
  - Process and analyze collected data to detect anomalies or inefficiencies.
  - Generate insights regarding energy consumption patterns.
- **Planning Module:**
  - Formulate optimization strategies to adjust HVAC operations based on current conditions.
- **Execution Module:**
  - Send control commands to the HVAC systems to dynamically adjust operation parameters.
  - Provide feedback loops to refine the control strategy continuously.

2.1.3. Blockchain Integration for Traceability

- **Data Recording:**
  - Store collected sensor data and control actions in an immutable blockchain ledger.
  - Ensure data is timestamped and cryptographically secured.
- **Access and Verification:**
  - Provide APIs for external regulators and stakeholders to query the blockchain.
  - Implement access control to manage different user roles (e.g., administrators, auditors).

2.1.4. Data Analysis and Gamification

- **Analytics Engine:**
  - Process historical and real-time data to compute performance metrics and energy consumption scores.
- **Gamification Module:**
  - Define scoring algorithms and leaderboards for various competitive categories (e.g., different floors, offices, or entire buildings).
  - Enable both centralized and decentralized ranking systems to support intra- and inter-organizational competitions.
- **User Interface (future development):**
  - Develop dashboards and visualizations to display performance scores, trends, gamification leaderboards, and real-time system status.
  - Support for multi-platform access (web, mobile).

2.1.5. Scalability and Adaptability

- **Modular Design:**
  - Design the system architecture to allow scaling from large corporate/public buildings down to smaller units like apartment complexes.
- **Configuration Management:**
  - Provide mechanisms for system configuration and parameter adjustments tailored to different building types.

2.1.6. Interoperability and Integration

- **API Services:**
  - Expose RESTful APIs for data access, control commands, and integration with third-party systems.
- **External System Integration:**
  - Support integration with existing building management systems and external monitoring services.

2.2. Non-Functional Requirements

2.2.1. Performance

- **Real-Time Response:**
  - Ensure that sensor data processing and HVAC control commands are executed with minimal latency.
- **Scalability:**
  - The system must support the addition of millions of sensors and devices and thousands of organizations without performance degradation.

2.2.2. Reliability and Availability

- **Fault Tolerance:**
  - Implement distributed architectures and redundancy strategies (e.g., Circuit Breaker pattern) to ensure high availability.
- **Data Integrity:**
  - Guarantee that data recorded on the blockchain remains secure and tamper-proof.

2.2.3. Security

- **Data Security:**
  - Encrypt data in transit and at rest, ensuring secure communication between sensors, control systems, and the blockchain.
- **Access Control:**
  - Implement role-based access control (RBAC) for all modules, especially for blockchain data queries and control operations.

2.2.4. Maintainability and Extensibility

- **Modular Architecture:**
  - Use microservices architecture patterns (e.g., Database per Service, Saga) to support independent development and maintenance of components.
- **Logging and Monitoring:**
  - Incorporate comprehensive logging and monitoring mechanisms for both system performance and security audits.

2.2.5. Usability

- **User-Friendly Interface:**
  - Design intuitive dashboards and control panels for both administrators and end-users.
- **Customization:**
  - Allow customization of views and reports to cater to different user roles and building configurations.

2.2.6. Compliance

- **Regulatory Compliance:**
  - Ensure the system adheres to relevant energy efficiency and data protection regulations.
- **Auditability:**
  - Provide audit trails through blockchain records for external verification and regulatory reviews.

2.3. Technical Requirements

2.3.1. Technology Stack

- **Programming Language:**
  - The system is implemented in Go for performance and concurrency.
- **Communication Protocols:**
  - Utilize MQTT, HTTP/HTTPS for device communication and RESTful APIs for data access, control commands, and integration with third-party systems.
- **Blockchain Framework:**
  - Integrate with an appropriate blockchain platform that supports immutable data storage and smart contracts.
- **Database Systems:**
  - Use distributed databases where needed to manage sensor data and system logs.

2.3.2. Architectural Patterns

- **Distributed System Patterns:**
  - Implement patterns like Circuit Breaker for fault tolerance, Database per Service for data isolation, and Saga for managing distributed transactions.

2.3.3. Deployment and Operations

- **Cloud-Native Deployment:**
  - Design for deployment on cloud platforms using containerization (e.g., Docker) and orchestration (e.g., Kubernetes).
- **CI/CD Pipelines:**
  - Establish continuous integration and deployment pipelines for regular updates and maintenance.
- **Monitoring Tools:**
  - Deploy monitoring and alerting systems to track system health and performance metrics.

3. Blockchain Requirements

3. Blockchain Requirements

3.1. Functional Requirements for Blockchain Archiviation

3.1.1. Data Packaging and Preparation

- **F3.1.1.1 Data Aggregation:**
  The system shall collect and aggregate sensor and control data, including timestamps and relevant metadata, from the data collection modules.

- **F3.1.1.2 Data Validation:**
  The system shall validate the integrity and completeness of the aggregated data before it is packaged for blockchain storage.

3.1.2. Blockchain Transaction Management

- **F3.1.2.1 Transaction Creation:**
  The system shall create a new blockchain transaction that encapsulates the packaged data, including all necessary metadata (e.g., sensor ID, timestamp, data type).

- **F3.1.2.2 Transaction Submission:**
  The system shall submit the created transaction to the designated blockchain network for validation and inclusion in a block.

- **F3.1.2.3 Transaction Retry Mechanism:**
  In the event of blockchain network unavailability or transaction failure, the system shall cache the transaction and automatically retry submission at regular intervals.

3.1.3. Data Retrieval and API Integration

- **F3.1.3.1 API for Data Query:**
  The system shall expose a RESTful API that allows authorized external users (e.g., regulators, auditors) to query and retrieve recorded blockchain data.

- **F3.1.3.2 Access Control:**
  The system shall implement role-based access control (RBAC) for the API, ensuring that only authenticated and authorized users can access blockchain records.

- **F3.1.3.3 Data Formatting and Presentation:**
  The system shall format and present queried blockchain data in a user-friendly

format, supporting filters such as date range, sensor type, and building identifier.

3.1.4. Audit and Logging

- **F3.1.4.1 Transaction Logging:**
  The system shall log every transaction submission, confirmation, and any error events associated with blockchain operations.

- **F3.1.4.2 Audit Trail:**
  The system shall maintain an immutable audit trail of all blockchain interactions to support regulatory reviews and forensic analysis.

3.2. Non-Functional Requirements for Blockchain Archiviation

3.2.1. Performance

- **NF3.2.1.1 Transaction Throughput:**
  The blockchain archiviation component shall support a transaction throughput that meets or exceeds the expected data submission rate from the sensor network.

- **NF3.2.1.2 Latency:**
  The time between data packaging and blockchain confirmation should be minimized to support near-real-time transparency. Specific latency targets should be defined based on network performance benchmarks.

3.2.2. Scalability

- **NF3.2.2.1 Horizontal Scalability:**
  The system shall be capable of handling an increasing number of transactions by scaling horizontally (e.g., distributed nodes or services).

- **NF3.2.2.2 Future Blockchain Integration:**
  The design should allow integration with alternative or additional blockchain networks as system demands evolve.

3.2.3. Security

- **NF3.2.3.1 Data Integrity:**
  All data stored on the blockchain shall be tamper-proof and cryptographically

secured to prevent unauthorized modifications.

- **NF3.2.3.2 Authentication and Authorization:**
  The system shall enforce strict authentication and authorization mechanisms for all API interactions and blockchain transactions.

- **NF3.2.3.3 Encryption:**
  Data in transit between the system and the blockchain, as well as sensitive data at rest, shall be encrypted using industry-standard cryptographic protocols.

3.2.4. Reliability and Fault Tolerance

- **NF3.2.4.1 Redundancy:**
  The blockchain component shall include redundancy measures to ensure continued operation in case of node failures or network issues.

- **NF3.2.4.2 Error Handling:**
  The system shall implement robust error handling and recovery strategies for blockchain submission failures, including automatic retries and fallbacks.

3.2.5. Maintainability and Extensibility

- **NF3.2.5.1 Modular Architecture:**
  The blockchain module shall be designed in a modular manner, allowing independent updates and easy integration of new blockchain features.

- **NF3.2.5.2 Documentation:**
  Comprehensive documentation must be provided for all blockchain-related processes, including API endpoints, transaction formats, and error handling procedures.

3.2.6. Compliance and Auditability

- **NF3.2.6.1 Regulatory Compliance:**
  The system shall comply with relevant industry standards and regulatory requirements concerning data transparency, privacy, and energy efficiency reporting.

- **NF3.2.6.2 Audit Readiness:**
  The blockchain archiviation component shall be designed to support periodic audits, providing verifiable, immutable records to external regulators.

4. Use Case Document

4. Software Use Case Document

4.1. Use Case: Data Collection and Environmental Monitoring

Actors:

- IoT Sensors
- Data Aggregator Service
- Building Management System (BMS)

Description:
This use case describes the process of continuously acquiring and aggregating environmental data (e.g., temperature, humidity) from a distributed network of IoT sensors installed throughout a building.

Preconditions:

- IoT sensors are installed and configured.
- A reliable network connection exists between sensors and the central Data Aggregator Service.

Basic Flow:

1. **Sensor Activation:** IoT sensors are activated and begin collecting environmental data continuously.
2. **Data Transmission:** Each sensor transmits real-time measurements (temperature, humidity, etc.) to the Data Aggregator Service.
3. **Data Aggregation:** The Data Aggregator Service receives, timestamps, and consolidates the data from all sensors.
4. **Data Storage:** Aggregated data is temporarily stored in a local cache or directly forwarded to the central database for processing by other modules.

Alternative Flows:

- **AF1 – Network Interruption:**
    - **Step 2A:** If the sensor loses network connectivity, it buffers data locally.
    - **Step 2B:** Once the connection is re-established, the buffered data is transmitted to the aggregator.
- **AF2 – Sensor Failure:**
    - **Step 1A:** If a sensor malfunctions, the BMS detects the failure and generates an alert for maintenance.

Extension Points:

- **EP1:** Data Validation – Before storage, data can be validated for accuracy and consistency.
- **EP2:** Pre-Processing – Data can be pre-processed (e.g., filtered or aggregated) if required by the Analytics Engine.

Inclusions:

- **I1:** Operations Logging – In all flows, operations are logged and reported to the system monitoring service.

4.2. Use Case: Dynamic HVAC Control via MAPE Cycle

Actors:

- MAPE Engine (comprising Monitor, Analyze, Plan, Execute modules)
- HVAC Systems

Description:
This use case explains how the system employs the MAPE (Monitor, Analyze, Plan, Execute) cycle to dynamically adjust HVAC operations in response to real-time sensor data, optimizing energy consumption.

Preconditions:

- Up-to-date sensor data is available.
- HVAC systems are connected and responsive to control commands.

Basic Flow:

1. **Monitor:** The MAPE Engine continuously monitors environmental conditions and energy consumption metrics received from the Data Aggregator Service.
2. **Analyze:** The Analysis Module processes the monitored data to identify inefficiencies or anomalies in the current HVAC operation.
3. **Plan:** Based on the analysis, the Planning Module develops an optimization strategy (e.g., adjusting temperature setpoints or fan speeds).
4. **Execute:** The Execution Module sends control commands to the HVAC systems to implement the planned adjustments.
5. **Feedback Loop:** Updated environmental conditions are re-monitored, and the cycle repeats to ensure continuous optimization.

Alternative Flows:

- **AF1 – No Optimization Needed:**
    - **Step 3A:** If the analysis finds no actionable inefficiencies, the system continues operating under the current settings.

- **AF2 – Control Command Failure:**
    - **Step 4A:** If an HVAC unit does not acknowledge a command, the system retries the command or issues an alert for manual intervention.

Extension Points:

- **EP1:** Manual Override – Allows facility managers to override automated control during maintenance or emergency situations.
- **EP2:** Integration with External Weather Data – Optionally incorporate external weather forecasts into the planning process.

Inclusions:

- **I1:** Logging and Audit – Every cycle iteration and decision is logged for future auditing and performance analysis.

4.3. Use Case: Blockchain-Enabled Data Recording and Transparency

Actors:

- Data Recording Service
- Blockchain Network
- External Regulators / Auditors

Description:
This use case details the process of securely recording sensor and control data on an immutable blockchain ledger to ensure transparency and traceability.

Preconditions:

- Sensor and control data have been aggregated by the system.
- A blockchain network is set up and connected to the Data Recording Service.

Basic Flow:

1. **Data Packaging:** The Data Recording Service collects the latest sensor data and control actions, packaging them with timestamps.
2. **Blockchain Transaction Creation:** A new transaction is created containing the packaged data.
3. **Transaction Submission:** The transaction is submitted to the blockchain network for validation.
4. **Confirmation and Recording:** Once validated, the transaction is immutably recorded on the blockchain.
5. **API Notification:** The system updates an API endpoint to signal that new data is available for external queries.

Alternative Flows:

- **AF1 – Blockchain Network Delay:**
  - **Step 3A:** If the blockchain network is temporarily unresponsive, the Data Recording Service caches the transaction and retries submission.
- **AF2 – Data Integrity Issue:**
  - **Step 1A:** If the data package fails integrity checks, it is flagged for review and not sent to the blockchain.

Extension Points:

- **EP1:** Smart Contract Execution – Automatically trigger rewards or notifications based on the recorded data.
- **EP2:** Data Encryption – Optionally encrypt sensitive data before packaging.

Inclusions:

- **I1:** Audit Logging – All steps, including alternative flows, are logged to ensure a complete audit trail.

4.4. Use Case: Data Analysis and Gamification

Actors:

- Analytics Engine
- Gamification Module
- End Users (e.g., facility managers, building occupants)

Description:
This use case covers the analysis of collected data to compute energy consumption scores and the use of gamification elements to motivate energy-efficient behaviors.

Preconditions:

- Sufficient historical and real-time data is available (stored in the blockchain and/or central database).
- Gamification rules and scoring algorithms are defined and configured.

Basic Flow:

1. **Data Retrieval:** The Analytics Engine retrieves historical and current sensor and control data.
2. **Metric Calculation:** The Analytics Engine calculates energy consumption metrics and performance scores.
3. **Score Generation:** The Gamification Module applies predefined algorithms to generate scores and ranks.

4. **Leaderboard Update:** The system updates public and/or private leaderboards to reflect the latest scores.
5. **User Notification:** End users receive notifications and visual feedback (via dashboards) regarding their performance.

Alternative Flows:

- **AF1 – Insufficient Data:**
    - **Step 1A:** If the Analytics Engine detects insufficient data for analysis, it triggers a data collection alert and uses fallback estimates.
- **AF2 – Gamification Rule Exception:**
    - **Step 3A:** If a gamification rule fails (e.g., due to data inconsistencies), a default scoring mechanism is applied, and an error is logged for review.

Extension Points:

- **EP1:** Customizable Leaderboards – Facility managers can customize leaderboard views (e.g., by floor, department, or building).
- **EP2:** Reward System – Integration with external reward systems to provide tangible incentives based on scores.

Inclusions:

- **I1:** Reporting Service – Includes generation of detailed reports for internal review and compliance purposes.

4.5. Use Case: External Transparency and Regulatory Query

Actors:

- External Regulators / Auditors
- API Gateway
- Blockchain Query Service

Description:
This use case defines how external stakeholders (e.g., regulators, auditors) can access immutable system data via secure APIs to verify compliance with energy standards and monitor sustainability efforts.

Preconditions:

- The blockchain ledger contains the latest recorded data.
- Authentication and access controls are in place for external API usage.

Basic Flow:

1. **Query Submission:** An external auditor submits a data query through the API Gateway.
2. **Authentication and Authorization:** The system validates the auditor's credentials and verifies their authorization.
3. **Data Retrieval:** The Blockchain Query Service retrieves the requested data from the blockchain.
4. **Response Delivery:** The data is formatted and returned via the API to the external auditor.
5. **Audit Logging:** The query and the response are logged for audit purposes.

Alternative Flows:

- **AF1 – Invalid Credentials:**
  - **Step 2A:** If authentication fails, the API Gateway returns an error message and logs the failed attempt.
- **AF2 – Data Unavailability:**
  - **Step 3A:** If the requested data is not available (e.g., due to network delays), the system returns a "data temporarily unavailable" notice and schedules a retry.

Extension Points:

- **EP1:** Data Filtering – Allow auditors to filter data by date range, building, or sensor type.
- **EP2:** Notification Service – Trigger notifications to administrators when external queries exceed a defined threshold.

Inclusions:

- **I1:** Security Logging – Every access attempt is logged to ensure traceability and support forensic analysis.

4.6. Use Case: System Scalability and Adaptability Management

Actors:

- System Administrators
- Configuration Management Service
- Microservices (various)

Description:
This use case describes how system administrators configure and manage the platform to

adapt to different building sizes and operational loads, ensuring that the solution remains performant and scalable.

Preconditions:

- The system is deployed in a cloud-native environment with container orchestration.
- Initial configuration settings are defined.

Basic Flow:

1. **Configuration Input:** Administrators access the Configuration Management Service to input or adjust system parameters tailored to a specific building or deployment scenario.
2. **Service Deployment:** The system dynamically deploys and scales microservices based on the configuration settings.
3. **Monitoring and Feedback:** The system continuously monitors performance metrics and resource utilization.
4. **Auto-Scaling:** Based on defined thresholds, additional microservices are deployed or scaled down automatically.
5. **Reporting:** Administrators receive periodic reports detailing the system performance and scaling actions.

Alternative Flows:

- **AF1 – Manual Override:**
    - **Step 2A:** If auto-scaling fails or is inappropriate (e.g., during maintenance), administrators can manually override scaling actions.
- **AF2 – Configuration Error:**
    - **Step 1A:** If an invalid configuration is detected, the system rejects the changes and provides diagnostic feedback for correction.

Extension Points:

- **EP1:** Integration with Third-Party Monitoring Tools – Optionally connect external monitoring and alerting systems for enhanced oversight.
- **EP2:** Historical Analytics – Use historical scaling data to forecast future capacity requirements.

Inclusions:

- **I1:** Audit and Change Logging – All configuration changes and scaling events are logged to support compliance and troubleshooting.

5. System Architecture

5. System Architecture

5.1. Back-end Architecture

Components:

- **IoT Sensor Layer:**
    - Devices deployed throughout the building capture environmental data.
    - Data is transmitted over secure protocols (e.g., MQTT over TLS).
- **Data Ingestion Service:**
    - Receives sensor data in real time.
    - Performs initial buffering, validation, and formatting.
    - Uses a message queue (e.g., Kafka, RabbitMQ) to decouple data ingestion from processing.
- **Data Aggregator and Pre-Processing Module:**
    - Consolidates sensor data.
    - Applies basic data cleansing and pre-processing.
    - Sends data to both the MAPE Engine and the Blockchain Archiviation module.
- **MAPE Engine (Dynamic HVAC Control Module):**
    - **Monitor:** Continuously receives environmental data.
    - **Analyze:** Evaluates sensor readings against thresholds and historical data.
    - **Plan:** Generates optimized control commands.
    - **Execute:** Communicates with the HVAC control systems.
    - Uses microservices architecture for each MAPE stage (or a unified service with modular design).

- **Blockchain Archiviation Module:**
    - Packages aggregate data and control actions.
    - Creates blockchain transactions and handles submission.
    - Manages a local cache/retry mechanism for failed transactions.
    - Provides interfaces for audit logging and data integrity checks.
- **Analytics and Gamification Engine:**
    - Processes historical and real-time data to compute energy efficiency metrics.
    - Generates gamification scores and updates leaderboards.
    - Stores analytical results in a centralized database for dashboard access.
- **Databases and Storage:**
    - **Time-Series Database:** Stores raw sensor data and operational metrics.
    - **Relational/NoSQL Database:** Stores processed data, user profiles, and gamification results.
    - **Blockchain Ledger:** Provides immutable storage for critical transactions.
    - **Audit Logs:** Centralized logging for debugging, compliance, and auditing.

Infrastructure:

- **Containerization:**
    - All services are containerized using Docker.
- **Orchestration:**
    - Kubernetes (or an equivalent) manages scaling, load balancing, and service health.
- **CI/CD Pipeline:**
    - Automated pipelines ensure smooth development, testing, and deployment.

5.1. API Layers

API Gateway:

- **Function:**
    - Acts as a single entry point for all external requests.
    - Routes incoming requests to appropriate microservices (e.g., data retrieval, control commands, blockchain queries).
    - Implements authentication, authorization (using RBAC), and rate limiting.

RESTful API Services:

- **Data Query API:**
    - Provides endpoints for retrieving sensor data, aggregated metrics, and analytics results.
    - Supports filters (date range, sensor type, building identifier).
- **HVAC Control API:**
    - Accepts control commands from the MAPE Engine.
    - Allows manual overrides and status updates.
- **Blockchain Query API:**
    - Exposes endpoints for external regulators and auditors.
    - Returns immutable records from the blockchain ledger.
- **Gamification API:**
    - Serves endpoints to fetch performance scores, leaderboards, and user profiles.
    - Supports notifications and score updates.

Communication and Security:

- **Protocols:**
    - All API communications use HTTPS to ensure data security.
- **Documentation:**
    - API endpoints are documented using OpenAPI/Swagger for ease of integration and testing.

---

3. Distributed Data Flow

Data Flow Overview:

1. **Data Acquisition:**
    - **Sensors → Data Ingestion Service:**
      Sensors capture environmental parameters and send data via secure protocols.
2. **Data Processing:**
    - **Data Ingestion → Message Queue:**
      Ingested data is pushed into a message queue, decoupling ingestion from processing.
    - **Message Queue → Data Aggregator:**
      The aggregator retrieves messages, validates and pre-processes data, then forwards it.
3. **MAPE Cycle and Control:**
    - **Data Aggregator → MAPE Engine:**
      Pre-processed data feeds into the MAPE Engine.
    - **MAPE Engine → HVAC Systems:**
      Optimized control commands are sent to HVAC units, and feedback is received.
4. **Blockchain Archiviation:**

- **Data Aggregator → Blockchain Module:**
  Aggregated data and control logs are packaged into transactions.

- **Blockchain Module → Blockchain Network:**
  Transactions are submitted; failed transactions are retried and logged.

5. **Analytics and Gamification:**

   - **Data Aggregator / MAPE Feedback → Analytics Engine:**
     Collected data, performance metrics, and control feedback are analyzed.

   - **Analytics Engine → Gamification Module:**
     Results are processed into scores and updated in leaderboards.

6. **API and External Access:**

   - **Internal Databases → API Gateway:**
     Processed data, analytics, and blockchain records are accessible via RESTful APIs.

   - **API Gateway → External Stakeholders:**
     External systems and users can query data securely.

Diagram (Conceptual Representation):

[Sensors]
↓ (MQTT/TLS)
[Data Ingestion Service]
↓ (Message Queue)
[Data Aggregator] ----------
↓ ↓
[MAPE Engine] → [HVAC Control Systems] [Blockchain Archiviation Module] → [Blockchain Network]
↓ ↓
[Analytics & Gamification Engine]
↓
[Databases & Storage]
↓
[API Gateway] → [External Applications / Regulators / Dashboards]

Next Steps

- **Refinement:**
    - Collaborate with development teams to refine component boundaries and protocols.
- **Prototyping:**
    - Develop low-fidelity prototypes for critical modules (e.g., data ingestion, MAPE engine, blockchain integration).
- **Validation:**
    - Validate data flow and integration points through iterative testing in a simulated environment.
- **Documentation:**
    - Update architectural documents as design details solidify.

6. Core Modules

6. Core Modules

6.1. Ingestion Service & Data Aggregation

6.1.1. Ingestion Service

- **Protocols Supported:**
    - MQTT over TLS for high-frequency streams.
    - HTTP/HTTPS POST for lower-volume ingestion or testing.
- **Message Validation:**
    - JSON schema enforcement: required fields (sensorId, timestamp, readings).
    - Reject or flag out-of-range values (e.g. temperature < –10 °C or > 50 °C).
- **Buffering & Retry:**
    - Local in-memory buffer with back-off retry on broker/HTTP endpoint failures.
- **Metrics & Monitoring:**
    - Expose ingestion rate, error rate, and buffer size via Prometheus metrics.

6.1.2. Data Aggregator

- **Batching & Windowing:**
    - Group incoming messages into fixed-length time windows (e.g. 1 minute) for downstream consumption.
- **Pre-Processing:**
    - Outlier detection (e.g. spikes/drops) and simple smoothing filters.
- **Fan-out:**
    - Publish cleaned batches to:
        1. Time-Series Database (InfluxDB/Prometheus)

2. Internal Message Queue (Kafka/RabbitMQ) for MAPE & Blockchain modules

- **Health Checks:**
    - HTTP health endpoint, self-test on startup (DB connectivity, queue reachability).

6.2. MAPE Engine

6.2.1. Monitor Module

- **Subscription:**
    - Consume cleaned data batches from the message queue.
- **State Management:**
    - Maintain a sliding window of recent readings per zone.
- **Metrics Collection:**
    - Publish real-time KPI metrics (average temp/humidity, energy usage) for Prometheus.

6.2.2. Analyze Module

- **Rule-Based Analysis:**
    - Compare current readings against static thresholds or simple moving averages.
- **Anomaly Detection:**
    - Flag deviations > X % from baseline; generate anomaly events.
- **Historical Baseline:**
    - Optionally retrieve historical data for same time of day to refine analysis.

6.2.3. Plan Module

- **Optimization Strategies:**
    - Define rule set: e.g. “If temp > 25 °C, plan setpoint = 22 °C.”
- **Multi-Zone Coordination:**
    - For groups of zones: balance comfort vs. energy (e.g. share cooling resources).
- **Plan Packaging:**
    - Create “plan packets” containing zone ID, target setpoint, fan speed.

6.2.4. Execute Module

- **Command Dispatch:**
    - Send plan packets as commands to HVAC actuator simulators via REST or MQTT.
- **Acknowledgement Handling:**
    - Wait for ACK/NACK; retry up to N times; on repeated failure generate alert.
- **Feedback Loop:**
    - Report actual achieved setpoint and energy delta back into the Monitor for continuous adjustment.

6.3. Blockchain Simulation

6.3.1. Transaction Builder

- **Data Composition:**
    - Combine sensor batch summary + executed commands + zone metadata + timestamp.
- **Serialization:**

- Encode transactions as compact JSON or Protobuf messages.

6.3.2. Local Ledger

- **Append-Only Store:**
  - Use embedded LevelDB or simple JSON append file with CRC checks.
- **Block Emulation (Optional):**
  - Group transactions into “blocks” at fixed intervals, compute hash chain.

6.3.3. Retry & Cache Mechanism

- **Failure Simulation:**
  - Introduce artificial “network down” flags to test cache logic.
- **Retry Scheduler:**
  - Exponential back-off retry; configurable max attempts before dead-letter queue.

6.3.4. Query Interface

- **REST API Endpoints:**
  - `GET /ledger/transactions?start=&end=&zoneId=`
  - `GET /ledger/transaction/{txId}`
- **Pagination & Filtering:**
  - Support page size, sort order, and filter by transaction type (sensor vs. command).

6.4. Analytics & Gamification

6.4.1. Analytics Engine

- **Data Sources:**
    - Time-series DB for raw metrics; Ledger for immutable records.
- **KPI Calculations:**
    - Energy per m², % uptime within optimal temperature band, anomaly count.
- **Batch vs. Stream:**
    - Initial MVP: scheduled batch jobs (e.g. every hour).
    - Future: consider real-time stream processing (Kafka Streams/Flink).

6.4.2. Scoring Module

- **Scoring Rules:**
    - Lower energy → higher base score; penalties for anomalies.
- **Groupings:**
    - Support multiple organizational hierarchies: zone, floor, building.
- **Weighting & Normalization:**
    - Normalize scores to account for zone size or baseline consumption.

6.4.3. Leaderboard Manager

- **Public vs. Private:**
    - Public: compare all zones; Private: restricted to specific group IDs.
- **Ranking Algorithms:**
    - Sort by descending score; ties broken by fewer anomalies.
- **Historical Trends:**

    - Show score evolution over time (sparkline charts).

6.5. API Endpoints

6.5.1. Authentication & Authorization

- **JWT Tokens:**
    - Issued by Auth Service; contain roles (admin, auditor, occupant).
- **RBAC Enforcement:**
    - Admins can Override HVAC; Auditors can query Ledger; Occupants see only their zone.

6.5.2. Sensor Data APIs

- `GET /api/v1/sensors/{sensorId}/latest`
- `GET /api/v1/sensors/{sensorId}/history?from=&to=&interval=`

6.5.3. Control APIs

`POST /api/v1/hvac/commands`

{ "zoneId": "...", "action": "setPoint", "value": 22 }

-
- `GET /api/v1/hvac/status?zoneId=`
- `POST /api/v1/hvac/override` (admin only)

6.5.4. Blockchain APIs

- `GET /api/v1/ledger/transactions` (with filter params)
- `GET /api/v1/ledger/transaction/{txId}`

6.5.5. Gamification APIs

- `GET /api/v1/leaderboards/{groupId}`
- `GET /api/v1/scores/{zoneId}`
- `GET /api/v1/trends/{zoneId}`

6.6. Initial Dashboard Requirements

6.6.1. Real-Time Sensor Dashboard

- **Widgets:**
    - Line charts for each zone, selectable via dropdown.
    - Connectivity status icons (green/yellow/red).
- **Data Feed:**
    - Live via WebSockets; fallback to polling every 5 s.

6.6.2. HVAC Control Panel

- **Display:**
    - Current setpoint, actual temperature, fan speed.
- **Controls:**
    - Buttons: +/– 1 °C, set fan to low/medium/high.
- **Logs:**
    - Latest 10 commands with timestamps and ACK status.

6.6.3. Blockchain Audit Viewer

- **Table View:**

    - Columns: TxID, zoneId, type (sensor/command), timestamp, status.

- **Filters:**
    - Date range picker, zone selector, transaction type.

6.6.4. Gamification Leaderboard

- **Ranking Table:**
    - Rank, zone/floor name, score, anomaly count.
- **Personal Highlight:**
    - Highlight the currently viewed zone’s position.

6.6.5. Notifications & Alerts

- **Banner Alerts:**
    - Sensor offline, repeated command failures, ledger retry backlog.
- **Drill-down Links:**
    - Click alert to view detailed logs or corrective actions.

7. Blockchain Integration

7. Blockchain Integration

7.1. Overview

The blockchain integration ensures immutable, verifiable storage of both raw sensor data and control actions. It comprises three main sub-systems:

1. **Transaction Builder** – Packages incoming data into blockchain-ready transactions.
2. **Local Ledger & Submission Engine** – Maintains an append-only store, handles retries, and submits to the external blockchain network.
3. **Ledger Query API** – Exposes stored transactions to authorized external consumers (e.g., regulators, dashboards).

7.2. Component Diagram

(See Figure 1 above TODO DA FILLARE CON DIAGRAMMA SU DRAW.IO)

7.2.1. Data Flow

1. **Data Aggregator → Transaction Builder**
    - **Responsibility:** Receives cleaned, batched sensor readings and MAPE control events.
    - **Interface:** In-memory queue (Kafka/RabbitMQ).
2. **Transaction Builder → Blockchain Module**
    - **Responsibility:** Serializes data into a compact transaction format, appends metadata (zoneId, timestamp, type).
3. **Blockchain Module → Blockchain Network**
    - **Responsibility:** Submits transactions to the chosen blockchain node; on failure, caches locally and retries.
4. **API Gateway ↔ Ledger Query API**
    - **Responsibility:** Routes incoming REST calls to the Ledger Query API; enforces JWT/RBAC.

5. **Ledger Query API → External Auditors / Dashboards**
    - **Responsibility:** Returns paginated, filterable transaction data for audit and display.

7.3. Sequence Diagram: Transaction Flow

(See Figure 2 above TODO FARE DIAGRAMMA CON DRAW.IO)

7.3.1. Basic Flow

1. **sendBatch(data)** from Data Aggregator to Transaction Builder
2. **buildTransaction()** within Transaction Builder
3. **submit(tx)** from Blockchain Module to Blockchain Node
4. **ack/receipt(txId)** from Blockchain Node back to Blockchain Module
5. **store locally & update status** in the local ledger

7.3.2. Alternative & Error Flows

- **Network Unavailable:**
    - After step 3, if submission fails, the module caches the transaction and schedules retries (exponential back-off).
- **Data Integrity Failure:**
    - During `buildTransaction()`, if validation fails (e.g. missing fields), the transaction is rejected and an error log is emitted.
- **Timeout on Receipt:**
    - If no `ack` within T seconds, the module flags the transaction as “pending” and re-issues up to N times before marking it “failed.”

7.4. REST API Specification

7.4.1. Security

- **Scheme:** BearerAuth (JWT)
- **Roles:**
    - `auditor`: read-only access to `/ledger/*`
    - `admin`: full access including transaction submission stats

7.4.2. Endpoints

Path	Method	Description	Auth
`/v1/ledger/transactions`	GET	List all transactions with filtering & pagination	auditor
`/v1/ledger/transactions/{txId}`	GET	Retrieve a single transaction by ID	auditor

7.4.2.1. Query Parameters for `/transactions`

- `start` (string, date-time, optional) – ISO 8601 start timestamp
- `end` (string, date-time, optional) – ISO 8601 end timestamp
- `zoneId` (string, optional) – Filter by sensor/zone ID
- `page` (integer, default 1) – Page number
- `pageSize` (integer, default 20, max 100) – Items per page

7.4.2.2. Responses

(TODO SISTEMARE VISUALIZZAZIONE FILE JSON)
200 OK –

openapi: 3.0.3

info:

title: NRG CHAMP Ledger API

version: 1.0.0

```
servers:
  - url: https://api.nrgchamp.example.com/v1
components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
  parameters:
    startDate:
      name: start
      in: query
      description: ISO8601 start timestamp (inclusive)
      required: false
      schema:
        type: string
        format: date-time
    endDate:
      name: end
      in: query
      description: ISO8601 end timestamp (inclusive)
      required: false
      schema:
        type: string
        format: date-time
    zoneId:
```

```
      name: zoneId
      in: query
      description: Filter by sensor/zone identifier
      required: false
      schema:
        type: string
paths:
  /ledger/transactions:
    get:
      summary: List blockchain transactions
      security:
        - BearerAuth: []
      parameters:
        - $ref: '#/components/parameters/startDate'
        - $ref: '#/components/parameters/endDate'
        - $ref: '#/components/parameters/zoneId'
        - name: page
          in: query
          description: Page number (default 1)
          schema:
            type: integer
            example: 1
        - name: pageSize
          in: query
          description: Items per page (max 100)
          schema:
```

```
          type: integer
          example: 20
    responses:
      '200':
        description: A paginated list of transactions
        content:
          application/json:
            schema:
              type: object
              properties:
                total:
                  type: integer
                page:
                  type: integer
                pageSize:
                  type: integer
                transactions:
                  type: array
                  items:
                    $ref: '#/components/schemas/Transaction'
  /ledger/transaction/{txId}:
    get:
      summary: Get a single transaction by ID
      security:
        - BearerAuth: []
      parameters:
```

```
        - name: txId
          in: path
          required: true
          description: Transaction identifier
          schema:
            type: string
      responses:
        '200':
          description: Transaction details
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Transaction'
        '404':
          description: Transaction not found
components:
  schemas:
    Transaction:
      type: object
      properties:
        txId:
          type: string
          description: Blockchain transaction ID
        zoneId:
          type: string
        timestamp:
```

```
        type: string
        format: date-time
      type:
        type: string
        enum: [sensor, command]
      payload:
        type: object
        description: Original data or command details
      status:
        type: string
        enum: [pending, committed, failed]
    required:
      - txId
      - zoneId
      - timestamp
      - type
      - payload
      - status
security:
  - BearerAuth: []
```

404 Not Found (for single-transaction fetch) –

- { "error": "Transaction not found" }

7.5. Configuration & Extensibility

- **Blockchain Provider:** Configurable endpoint and credentials (e.g. Hyperledger, Ethereum testnet).
- **Retry Policy:** Parameters for max attempts, back-off intervals, and dead-letter queue.
- **Encryption Toggle:** Enable/disable payload encryption via environment variable.
- **Metrics Exposure:** Prometheus metrics for submission attempts, successes, failures, and ledger length.

Next Steps:

- Finalize data schemas and smart-contract interfaces.
- Implement the above API spec in the Ledger Query microservice.
- Integrate with a lightweight blockchain emulator for end-to-end tests.