



# Product Requirements and Specification Document

---

## Project Name

**Enerlytics - Energy Consumption Analytics Service**

## Overview

Enerlytics is a Spring Boot microservice designed to collect, store, and analyze energy consumption data from smart meters. The service ingests data asynchronously, persists it in PostgreSQL, provides analytics via REST endpoints, secures access with JWT, and is containerized using Docker.

---

## 1. Objectives

- Efficiently ingest and store smart meter data.
  - Provide real-time and historical analytics.
  - Ensure secure, scalable, and maintainable architecture.
  - Enable easy deployment via Docker containers.
- 

## 2. Stakeholders

Role	Responsibility
Product Owner	Requirements, prioritization
Developers	Implementation, testing
DevOps	Deployment, monitoring
Data Analysts	Consume analytics endpoints

---

## 3. Functional Requirements

ID	Requirement
FR1	Accept energy consumption data via REST API (JSON payload).
FR2	Ingest data asynchronously (e.g., using message queues or async tasks).
FR3	Persist data in PostgreSQL.
FR4	Expose analytics endpoints (e.g., total, average, peak usage).
FR5	Support querying by time range, meter ID, and aggregation type.
FR6	Secure all endpoints with JWT authentication.
FR7	Provide OpenAPI (Swagger) documentation.

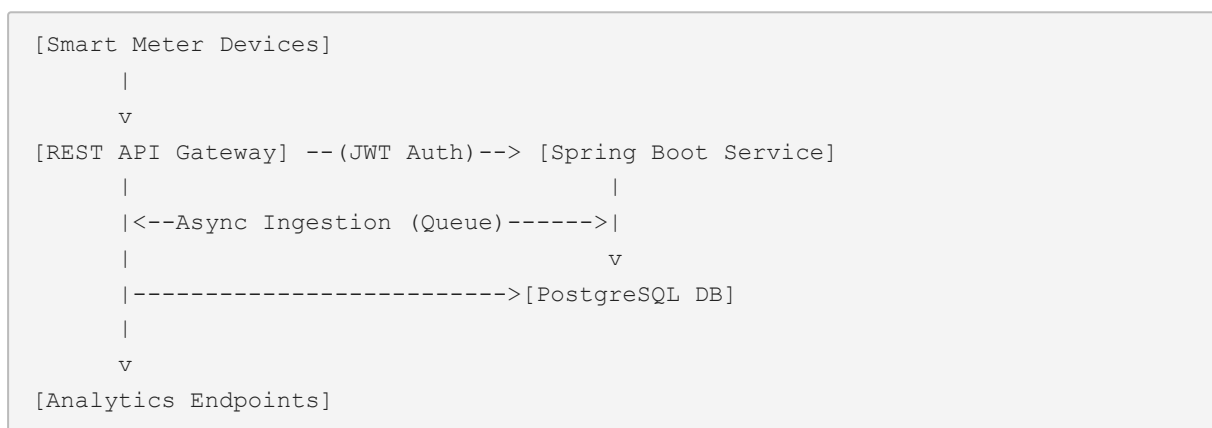
---



## 4. Non-Functional Requirements

ID	Requirement
NFR1	Average API response time < 500ms
NFR2	Handle at least 1000 concurrent requests
NFR3	Data consistency and durability
NFR4	Containerized deployment (Docker)
NFR5	Codebase managed with Maven

## 5. System Architecture



## 6. Data Model

Entity	Fields
MeterReading	id (UUID), meterId (String), timestamp (UTC), consumption (Decimal)
User	id (UUID), username, passwordHash, roles

## 7. API Endpoints (Sample)

Method	Endpoint	Description	Auth
POST	/api/readings	Ingest meter reading	Yes
GET	/api/analytics/total	Total consumption (params: meterId, from, to)	Yes
GET	/api/analytics/average	Average consumption	Yes
GET	/api/analytics/peak	Peak usage	Yes
POST	/api/auth/login	Obtain JWT token	No

## 8. Security



- All endpoints (except /auth/login) require JWT Bearer authentication.
  - Passwords stored as secure hashes.
  - Role-based access control for future extensibility.
- 

## 9. Deployment

- Service packaged as a Docker container.
  - Configuration via environment variables.
  - PostgreSQL runs as a separate container or managed service.
  - Example `docker-compose.yml` provided.
- 

## 10. Technology Stack

Component	Technology
Language	Java 17+
Framework	Spring Boot 3.x
Database	PostgreSQL 14+
Build Tool	Maven
Container	Docker
Auth	JWT (Spring Security)
Docs	OpenAPI/Swagger

---

## 11. Acceptance Criteria

- All functional requirements implemented and tested.
  - JWT authentication enforced.
  - Analytics endpoints return correct, performant results.
  - Service runs successfully in Docker.
  - OpenAPI documentation available at `/swagger-ui.html`.
- 

## 12. Out of Scope

- Real-time streaming analytics (batch only).
  - UI/dashboard (API only).
  - Multi-tenancy.
- 

## 13. Sample Data Ingestion Payload



```
{
  "meterId": "MTR-123456",
  "timestamp": "2024-06-01T12:00:00Z",
  "consumption": 5.75
}
```

## 14. Milestones

Milestone	Description	Target Date
API & Data Model	Define endpoints, entities	Week 1
Async Ingestion & Storage	Implement ingestion, DB persistence	Week 2
Analytics Endpoints	Implement analytics logic	Week 3
Security & Docs	JWT, OpenAPI, tests	Week 4
Dockerization	Containerize, compose setup	Week 5

## 15. Risks & Mitigations

Risk	Mitigation
High ingestion load	Use async processing, optimize DB
Security vulnerabilities	Use proven libraries, code review
Data loss	Regular DB backups

End of Document