**Energy Community Documentation**

**1. Introduction**

The Energy Community Monitoring System is a distributed microservices application designed to collect, process, and visualize energy production and consumption data within a community. The solution comprises the following components:

* **Producer Service**: Simulates local energy production.
* **User Service**: Simulates community energy consumption.
* **Usage Service**: Aggregates time‐bucketed usage and production metrics into an hourly\_usage datastore.
* **Percentage Service**: Calculates the share of grid usage vs. community depletion per hour.
* **Energy REST API**: Exposes endpoints for current, historical, and available‐range data; publishes percentage events.
* **JavaFX GUI**: Desktop dashboard for interactive visualization of current and historical metrics.
* **RabbitMQ**: Asynchronous message bus connecting services.
* **PostgreSQL** : Persistent storage for usage data.

All components can be built and started independently, and communicate via RabbitMQ queues.

**2. Design Ideas & Architecture**

**2.1 Microservices & Messaging**

Each service is a Spring Boot application with its own responsibilities. They communicate exclusively via RabbitMQ using domain‐specific DTOs (EnergyMessage, HourlyUsageMessage, PercentageData). This decouples producers from consumers, allows horizontal scaling, and ensures resilience to partial outages.

[Producer] -> energy.input -> [Usage Service] -> energy.update -> [Percentage Service] -> energy.percentage -> [REST API]

[User] -> energy.input -> [Usage Service]

* **Queues**:
  + energy.input: raw production/consumption messages
  + energy.update: enriched hourly‐usage messages
  + energy.percentage: percentage calculation results

**2.2 Persistence Layer**

The Usage Service uses Spring Data JPA to manage an hourly\_usage table via the HourlyUsageRepository. That repository hides SQL/DB details and provides CRUD operations plus custom queries if needed.

public interface HourlyUsageRepository extends JpaRepository<HourlyUsage, Instant> { }

**2.3 REST API**

The EnergyController exposes:

* **GET /energy/current**: latest percentage snapshot
* **GET /energy/historical?start=&end=**: list of hourly percentages
* **GET /energy/available-range**: minimal and maximal timestamps in DB
* **POST /energy/publish**: manual injection of percentage data

Responses use DTOs (EnergyData) with JSON serialization handled by Jackson.

**2.4 GUI**

A JavaFX application binds to REST endpoints with a lightweight RestClient. It displays:

* Current metrics in a text area
* Historical data in a table
* Date/time pickers limited by /available-range

**3. Lessons Learned**

1. **Decoupling via Messaging**: RabbitMQ enabled independent scaling and restart without data loss.
2. **Spring Data Repository Pattern**: Rapid DB integration and testing via mock repositories.
3. **Jackson Aliases & Formats**: Handling multiple JSON field names and Instant vs. LocalDateTime serialization required careful annotations (@JsonAlias, @JsonSetter, @JsonFormat).
4. **JavaFX Threading**: UI updates must occur on the FX thread; improper use causes exceptions or freezes.
5. **Integration Testing**: Mocking RestClient and JUnit/Spring Boot test slices simplified validation of each layer.
6. **Docker Compose**: Coordinating multi‐container startup and service hostname resolution (e.g., using localhost vs. service names).

**4. Time Tracking**

| **Component** | **Estimated Hours** | **Notes** |
| --- | --- | --- |
| Project Setup | 3 | Repo structure, Docker |
| Producer Service | 5 | Simulated production, tests |
| User Service | 6 | Consumption logic, schedule, tests |
| Usage Service | 7 | JPA mapping, hourly aggregation, tests |
| Percentage Service | 8 | Calculation logic, DTOs, tests |
| REST API | 10 | Controllers, service layer, endpoints, tests |
| JavaFX GUI | 8 | FXML layout, controllers, styling, tests |
| Integration & Debug | 10 | RabbitMQ glue, time parsing, error handling |
| Documentation & Demo | 3 | README, this document, presentation materials |
| **Total** | 60 |  |