**PayMaytrix -- Implementation – Electricity**

**Technology Stack:**

* Programming Language: Java
* Framework: Spring Boot
* Microservice Architecture using Spring Cloud (Eureka, Zuul)
* Database: PostgreSQL/MariaDB
* Smart Meter Integration: MQTT (Message Queuing Telemetry Transport)
* Billing System Integration: RESTful API
* Frontend: ReactJS

**Project Structure:**

├── billing-service

│ ├── src

│ ├── pom.xml

├── meter-service

│ ├── src

│ ├── pom.xml

├── user-service

│ ├── src

│ ├── pom.xml

├── gateway-service

│ ├── src

│ ├── pom.xml

├── pom.xml

**1. User Authentication and Authorization:**

User Service:

* Define a "User" entity with fields such as **id**, **email**, **password**, and **role**.
* Implement user registration and login endpoints using Spring Security and JWT for authentication.
* Use BCryptPasswordEncoder for secure password storage and validation.

**2. User Profile Management:**

User Service:

* Create a "UserProfile" entity with fields like **name**, **contactInformation**, and **address**.
* Implement APIs to allow users to view and update their profiles.

**3. Electricity Usage Monitoring:**

Meter Service:

* Implement APIs to receive real-time electricity usage data from smart meters via MQTT.
* Store the usage data in a "UsageData" entity associated with the user and property IDs.
* Use Spring Integration to handle MQTT message reception.

**4. Bill Generation and Payment:**

Billing Service:

* Create a "Bill" entity with fields like **billId**, **userId**, **propertyId**, **usageData**, **amount**, and **status**.
* Implement a scheduled job to generate bills based on electricity usage data and tariff rates.
* Store the generated bills in the database.

Payment Service:

* Integrate with one or more payment gateway providers to handle online payments.
* Implement APIs for consumers to record bill payments and update the bill status.

**5. Bill Reminders and Notifications:**

Billing Service:

* Implement scheduled tasks to send upcoming bill notifications and payment reminders.
* Use email or SMS gateways for notifications based on user preferences.

**6. Customer Support:**

User Service:

* Create a "SupportTicket" entity with fields like **ticketId**, **userId**, **issueDescription**, and **status**.
* Implement APIs to allow users to create support tickets and check their status.

**7. Security and Privacy:**

Gateway Service (Zuul):

* Use Spring Cloud Zuul as an API gateway to handle authentication, authorization, and routing.
* Set up filters for logging, rate limiting, and security checks.

**8. Scalability and Performance:**

Load Balancing:

* Use Spring Cloud Eureka for service discovery and load balancing.
* Deploy multiple instances of each microservice for scalability.

Caching:

* Implement caching using Spring Cache to reduce database queries and improve performance.

**9. Integration and Extensibility:**

Microservice Communication:

* Implement asynchronous communication between microservices using Spring Cloud Stream.

**10. Error Handling and Reporting:**

Global Exception Handling:

* Implement a global exception handler to handle errors and return appropriate HTTP status codes and error messages.

**11. Deployment and Infrastructure:**

Containerization:

* Use Docker to containerize each microservice for easier