Requirement Analysis

Functional requirements

1. **User Authentication and Authorization:**

a. User Registration:

- Create a "Users" table in the database to store user information.
- Include fields such as user ID, email, hashed password, and role.
- Implement a registration endpoint that receives user details, generates a unique user ID,hashes the password, and saves the user to the database.

b. User Login:

- Implement a login endpoint that accepts user credentials and verifies them against thestored hashed password.
- Generate a JWT token upon successful authentication and return it to the client.
- Secure the APIs by checking the validity of the JWT token for each request.

2. **User Profile Management:**

a. User Profile:

- Create a "UserProfile" table in the database to store user profile details, including name,contact information, and address.
- Associate each user profile with the corresponding user ID.

b. Multiple Properties:

- Create a "Properties" table in the database to store property information.
- Include fields such as property ID, address, meter number, and tariff plan.
- Establish a one-to-many relationship between the user profile and properties table to associate multiple properties with each user.

3. **Electricity Usage Monitoring:**

a. Real-time Usage:

- Integrate with smart meters or provide a manual entry form for users to submit their electricity usage data.
- Store the usage data in a "UsageData" table in the database, associating each recordwith the user and property IDs.

b. Historical Usage Analysis:

- Implement a data analysis module that processes the usage data and generateshistorical usage statistics.
- Use charting libraries or data visualization tools to present the analysis in a graphicalformat.

c. Abnormal Usage Detection:

- Develop algorithms to detect abnormal usage patterns based on predefined thresholdsor machine learning models.
- Implement a notification system to alert users when abnormal usage patterns are detected.

4. Bill Generation and Payment:

a. Bill Generation:

- Create a "Bills" table in the database to store bill details.
- Generate bills based on the user's electricity usage data and applicable tariff rates.
- Store the generated bills in the database, associating them with the user and propertyIDs.

b. Bill Presentation and Download:

- Implement an API endpoint to retrieve the user's bills in PDF format.
- Enable users to download their bills from their accounts.

c. Payment Integration:

- Integrate with one or more payment gateway providers that support credit/debit cards,net banking, and mobile wallets.
- Implement secure payment processing by encrypting and tokenizing sensitive payment information.

5. Bill Reminders and Notifications:

a. Upcoming Bill Notifications:

- Implement a notification system that sends reminders to users before their bill duedates.
- Configure the notification channels (email, SMS, push notifications) based on user preferences.

b. Payment Reminders:

- Send automated reminders to users for pending payments, with configurable intervalsand escalation mechanisms.

c. Payment Confirmation:

- Send notifications to users for successful payments, providing payment receipts or confirmation numbers.

6. Customer Support:

a. Support Ticket System:

- Create a "SupportTickets" table in the database to store support ticket details.
- Include fields such as ticket ID, user ID, issue description, status, and assigned staff ID.
- Implement an API endpoint to create and manage support tickets.

b. Ticket Assignment and Resolution:

- Assign support tickets to relevant staff members based on category or priority.
- Implement an update mechanism to keep users informed of the status of their tickets

Non-Functional requirements

Security and Privacy:

a. User Data Protection:

● User data should be securely stored and transmitted using encryption techniques.

● Implement industry-standard security practices, such as secure coding guidelines for secure development, including input validation, parameterized queries, and protection against common vulnerabilities (e.g., cross-site scripting, SQL injection).

● Use secure protocols (e.g., HTTPS) for data transmission over networks: Encrypt data during transit to prevent eavesdropping and ensure data integrity.

● Store sensitive data, like passwords and payment details, in an encrypted format using strong encryption algorithms: Apply encryption techniques (e.g., AES) to protect sensitive data at rest.

JWT Authentication:

We will be configuring Spring Security and JWT for performing 2 operations-

● Generating JWT - Expose a POST API with mapping /authenticate. On passing correct username and password it will generate a JSON Web Token(JWT), upon successful authentication and return it to the client.

● Validating JWT - If user tries to access GET API with mapping /hello. It will allow access only if request has a valid JSON Web Token(JWT)

This will ensure the security of the APIs by checking the validity of the JWT token for each request.

b. Access Controls:

● Implement authentication mechanisms to ensure only authorized users can access the application: Use secure authentication methods (e.g., username/password, multi-factor authentication (Email-Verification)) to verify user identities.

● Employ role-based access controls (RBAC) to assign different permissions to different user roles: Classify users into roles (e.g., admin, consumer, Department Officials) and define access permissions

based on these roles. Different type of user will be having the access to the different access points to avoid any kind of access point or data collision.

● Implement session management and enforce secure session handling to prevent unauthorized access: Generate secure session tokens, manage session expiration, and protect against session hijacking or fixation attacks.

Performance and Scalability:

Horizontal Scaling:

Horizontal scaling involves adding more instances (servers) to distribute the workload and handle increased user demand. Here are some ways to horizontally scale the application:

a. Load Balancing:

Implement a load balancer that distributes incoming requests across multiple application servers. The load balancer can use different algorithms (e.g., round-robin, least connections) to evenly distribute the traffic.

b. Auto-scaling:

Use an auto-scaling mechanism that automatically adjusts the number of application server instances based on the current workload. This can be achieved by setting up policies or rules that define when to add or remove instances based on CPU utilization, request queue length, or other performance metrics.

c. Database Scaling:

Scale the database by using techniques such as sharding, which involves partitioning data across multiple database servers. This can be done based on customer IDs, geographic regions, or any other logical division of data.

Vertical Scaling:

Vertical scaling involves increasing the resources (CPU, memory, storage) of individual servers to handle higher loads. Here are some approaches to vertically scale the application:

a. Upgrade Hardware:

Upgrade the existing servers with higher-capacity hardware components such as CPU, RAM, or storage. This can be achieved by adding more powerful processors, increasing the memory capacity, or switching to faster storage drives (e.g., SSD).

b. Vertical Database Scaling:

If the database is a performance bottleneck, vertically scale it by upgrading the database server hardware or using more powerful database instances. Additionally, optimize database configurations and indexing to improve performance.

c. Caching:

Implement caching mechanisms to offload the application and database servers. Use in-memory caches (e.g., Redis, Memcached) to store frequently accessed data, reducing the load on the backend systems and improving response times.

d. Asynchronous Processing:

Offload resource-intensive or time-consuming tasks to background workers or separate services. By performing these tasks asynchronously, the main application can respond faster to user requests, improving overall performance.

It's important to note that a combination of horizontal and vertical scaling can be employed based on specific requirements and constraints. Regular monitoring and performance testing will help identify scaling needs and ensure the appropriate scaling strategy is implemented.

Logging and Monitoring:

a. Logging:

Implement a logging framework to capture system logs, user activities, errors, and system events: Log relevant information for auditing, troubleshooting, and compliance purposes.

● We can use the SLF4J (Simple Logging Facade for Java) logger in our application for logging purposes. SLF4J provides a simple and flexible logging API that acts as a facade or abstraction layer for various logging frameworks, such as Logback, Log4j, and Java Util Logging (JUL).

● SLF4J allows you to write log statements in your code using a common API and provides the flexibility to switch between different logging implementations without changing your code. It provides various logging levels (e.g., INFO, DEBUG, WARN, ERROR) to differentiate the severity of log messages.

User Experience:

Design a user-friendly interface that is intuitive and easy to navigate, by using suitable frontend development. Ensure responsive design to provide a consistent experience across different devices and screen sizes. Optimize the application's performance to provide smooth and seamless user interactions. Provide clear and informative error messages and notifications to assist users in resolving issues.

Types of Exceptions:

Authentication and Authorization Exceptions:

- InvalidCredentialsException: Raised when the user provides incorrect login credentials.
- AccountLockedException: Raised when the user's account is locked or suspended.
- UnauthorizedAccessException: Raised when a user tries to access a feature or resource without sufficient permissions.

User Profile Management Exceptions:

- InvalidInputException: Raised when the user provides invalid or incomplete information while updating their profile.
- DuplicatePropertyException: Raised when a user tries to add a property that already exists in their profile.
- ProfileNotFoundException: Raised when a requested user profile is not found in the system.

Electricity Usage Monitoring Exceptions:

- MeterConnectionException: Raised when there is a problem connecting to the smart meter for real-time usage monitoring.

- InvalidReadingException: Raised when the user provides an invalid reading or meter data during manual entry.
- DataAccessException: Raised when there are issues accessing or storing the usage data in the database.

Bill Generation and Payment Exceptions:

- BillingCalculationException: Raised when there are errors in calculating the electricity bill based on usage data and tariff rates.
- PaymentProcessingException: Raised when there are issues with processing the payment transaction through the payment gateway.
- BillNotFoundException: Raised when a requested bill is not found in the system.

Bill Reminders and Notifications Exceptions:

- NotificationSendingException: Raised when there are errors in sending notifications to users (e.g., email sending failure).
- ReminderSettingException: Raised when there are issues in setting up or scheduling bill reminders for users.
- AbnormalUsageException: Raised whenever uses the electricity above from threshold of their tariff plan i.e. high consumption.

Customer Support Exceptions:

- InvalidQueryException: Raised when a user submits an invalid or incomplete query to the customer support feature.
- SupportTicketNotFoundException: Raised when a requested support ticket is not found in the system.
- SupportStaffUnavailableException: Raised when there are no available staff members to handle a support ticket.

Test Cases:

User Authentication and Authorization:

Test Case 1: Verify that a user can register a new account successfully.

Test Case 2: Validate that a registered user can log in with valid credentials.

Test Case 3: Ensure that an unauthorized user is restricted from accessing protected features.

Test Case 4: Validate that user roles are correctly assigned, and access permissions are enforced.

User Profile Management:

Test Case 5: Verify that users can view and update their personal information.

Test Case 6: Validate that users can add multiple properties and associate them with their profile.

Test Case 7: Ensure that changes made to user profiles are saved accurately.

Electricity Usage Monitoring:

Test Case 8: Verify that real-time electricity usage data is accurately retrieved and displayed.

Test Case 9: Validate that historical usage data is stored correctly and can be graphically presented.

Test Case 10: Ensure that notifications for abnormal usage patterns or high consumption are triggered correctly.

Bill Generation and Payment:

Test Case 11: Verify that bills are generated accurately based on the user's electricity usage data and tariff rates.

Test Case 12: Validate that users can view and download their bills in PDF format.

Test Case 13: Test the integration with payment gateways to ensure successful and secure payment processing.

Test Case 14: Validate that multiple payment options (credit/debit cards, net banking, mobile wallets) are working correctly.

Bill Reminders and Notifications:

Test Case 15: Verify that users receive notifications for upcoming bill due dates.

Test Case 16: Validate that automated reminders are sent for pending payments.

Test Case 17: Ensure that notifications are sent for successful payments and bill confirmations.

Customer Support:

Test Case 18: Verify that users can raise queries or report issues through the customer support feature.

Test Case 19: Validate that support tickets are assigned to relevant staff for timely resolution.

Test Case 20: Ensure that users receive updates on the status of their queries.

API endpoints to consider:

User Management Endpoints:

POST /api/register: Register a new user account.

POST /api/login: Authenticate a user and generate an authentication token.

GET /api/ user-profiles: Retrieve the user's profile information.

PUT /api/ user-profiles: Update the user's profile information.

GET /api/users/properties: Retrieve the properties (houses, apartments) associated with the user's profile.

POST /api/users/properties: Add a new property to the user's profile.

Electricity Usage Endpoints:

GET /api/ /usage-data: Retrieve real-time electricity usage data.

POST /api/ /usage-data: Submit manual entry for electricity usage.

GET /api/ /usage-data/history: Retrieve historical usage data for a specific user.

Bill Management Endpoints:

GET /api/bills: Retrieve a list of bills for a user.

GET /api/bills/{billId}: Retrieve details of a specific bill.

GET /api/bills/{billId}/pdf: Download the PDF version of a bill.

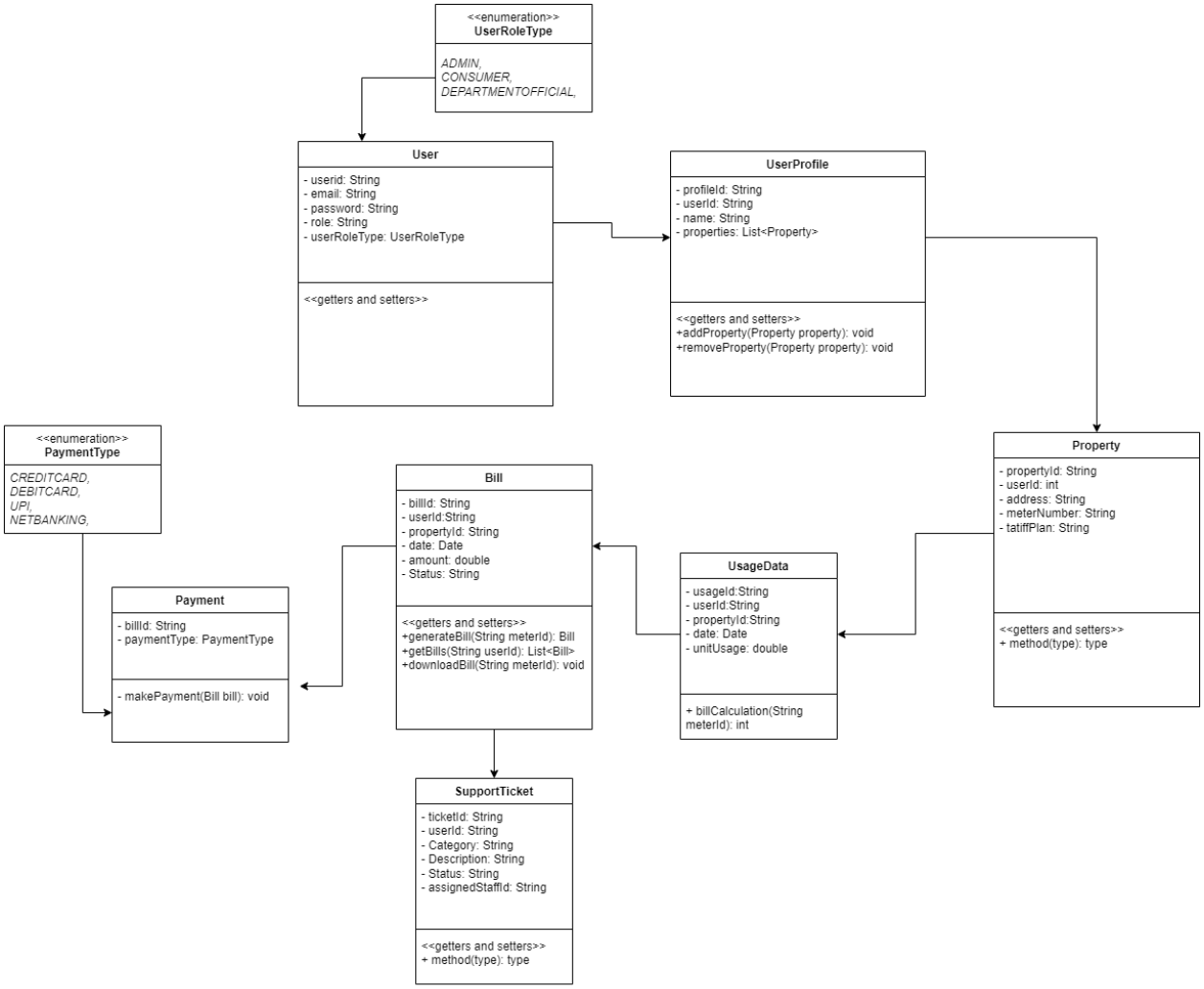
POST /api/payments: Submit a payment for a bill.

Notification Endpoints:

POST /api/notifications/reminder: Set a reminder for upcoming bill due dates.

POST /api/notifications/query: Raise a support query or report an issue.

UML DIAGRAM



User represents the user entity with attributes like userId, email, password, userRoleType and role.

UserProfile represents the user's profile entity with attributes like profileId, userId, name, contact, and address.

Property represents the property entity associated with each user, with attributes like propertyId, userId, address, meterNumber, and tariffPlan.

UsageData represents the electricity usage data entity with attributes like usageId, userId, propertyId, date, and usage.

Bill represents the bill entity with attributes like billId, userId, propertyId, date, amount, and status.

SupportTicket represents the support ticket entity with attributes like ticketId, userId, category, description, status, and assignedStaffId.

Code Link:

https://github.com/lavkumar3112/ElectricityBillSystem