Utility Bill Notification and Payment System

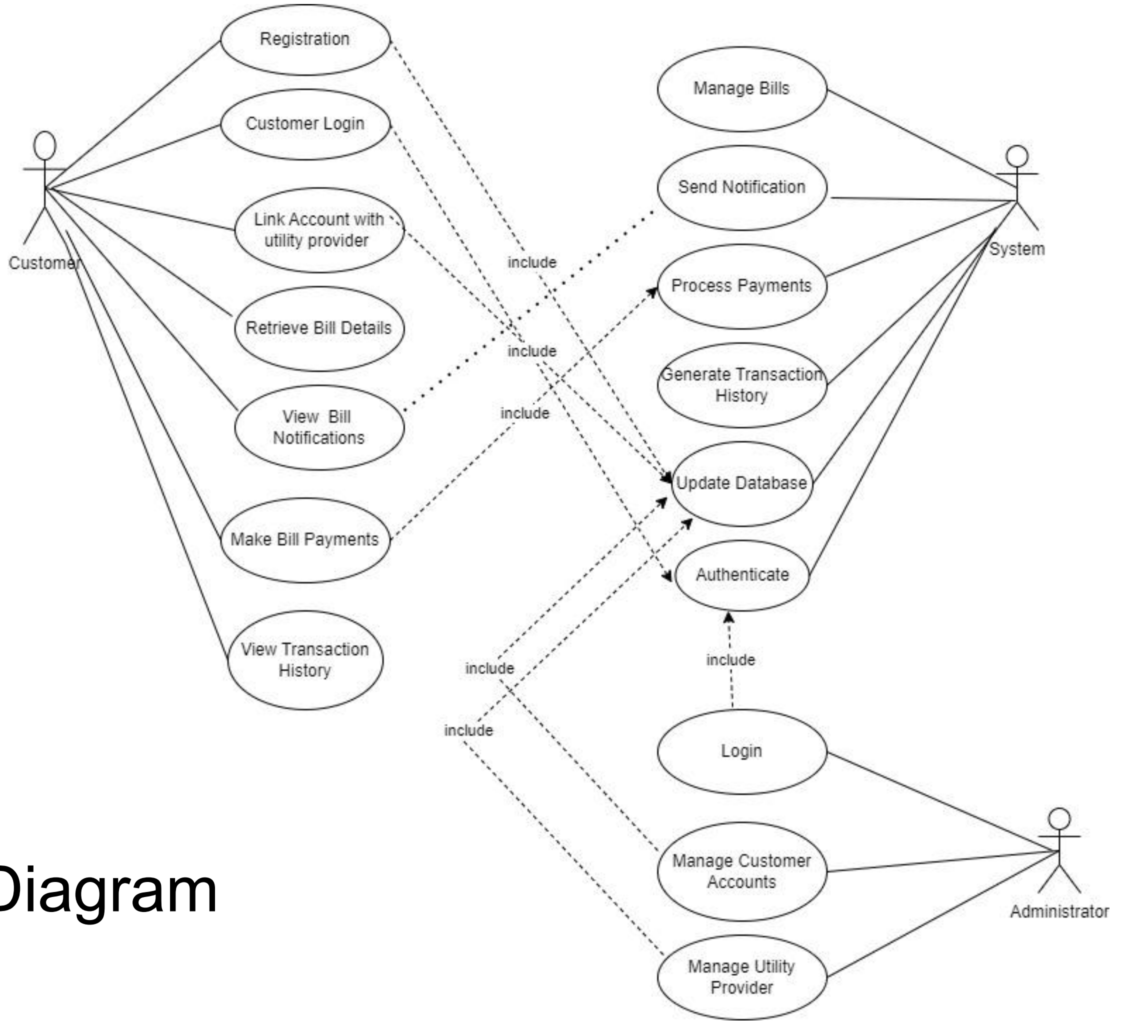
Vector Squad

Assumptions

- Availability of APIs from utility providers for bill retrieval.
- Third-party payment API for processing payments.
- The system will store customer data, including name and mobile number.
- The system will store utility provider details, including API access credentials.
- The system will have access to a scheduler to trigger bill retrieval and notifications.

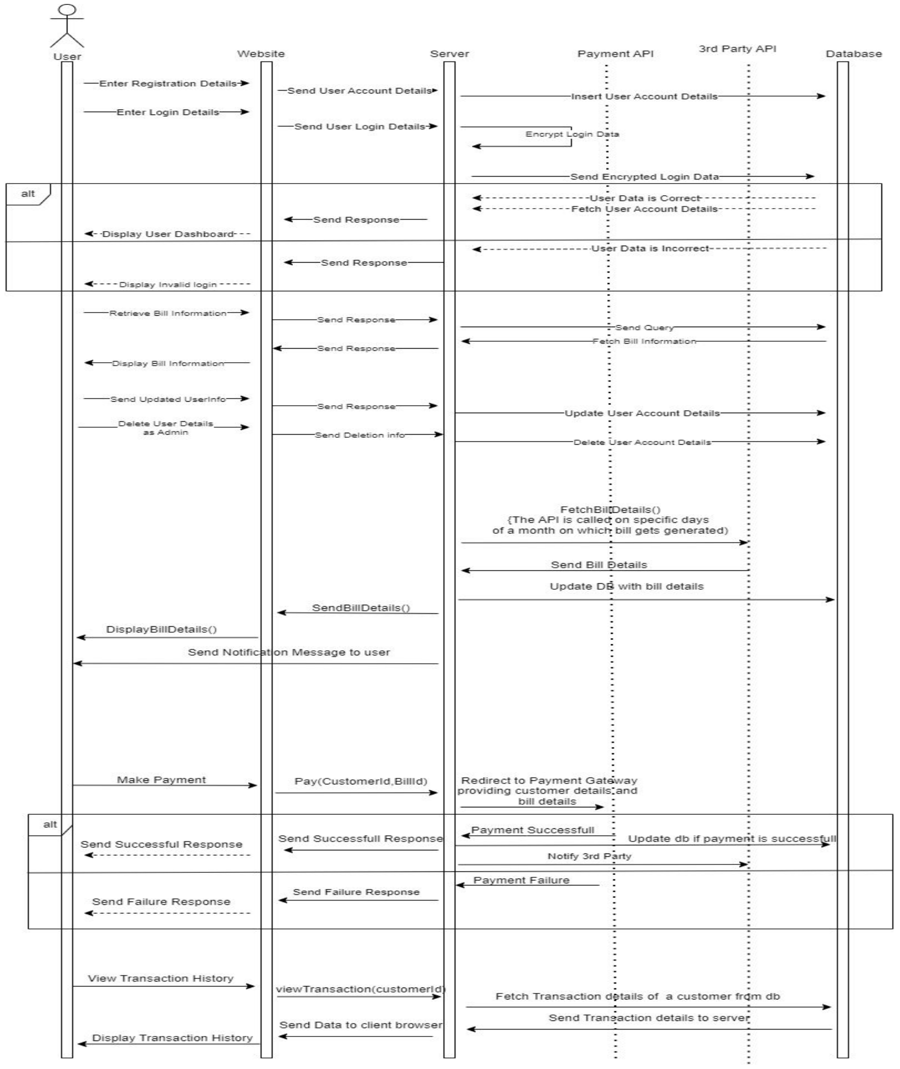
High Level Design Modules

- User Management Module:
  - Responsible for onboarding new customers and storing their data.
  - Handles customer authentication and access control.
- Utility Provider Registration:
  - Allows the admin to register new utility providers.
  - Stores API details for each provider.
- Bill Retrieval Module:
  - Scheduled task for bill retrieval based on the provider's schedule.
  - Utilizes the API details of registered utility providers to fetch bills.
  - Stores bill details in a database.
- Notification Module:
  - Sends notifications/reminders to customers about upcoming bills.
  - Utilizes customer data and bill details from the database.
  - Runs scheduled tasks for sending notifications.
- Payment Processing Module:
  - Initiates payments through the third-party payment API.
  - Records payment status and transaction history.

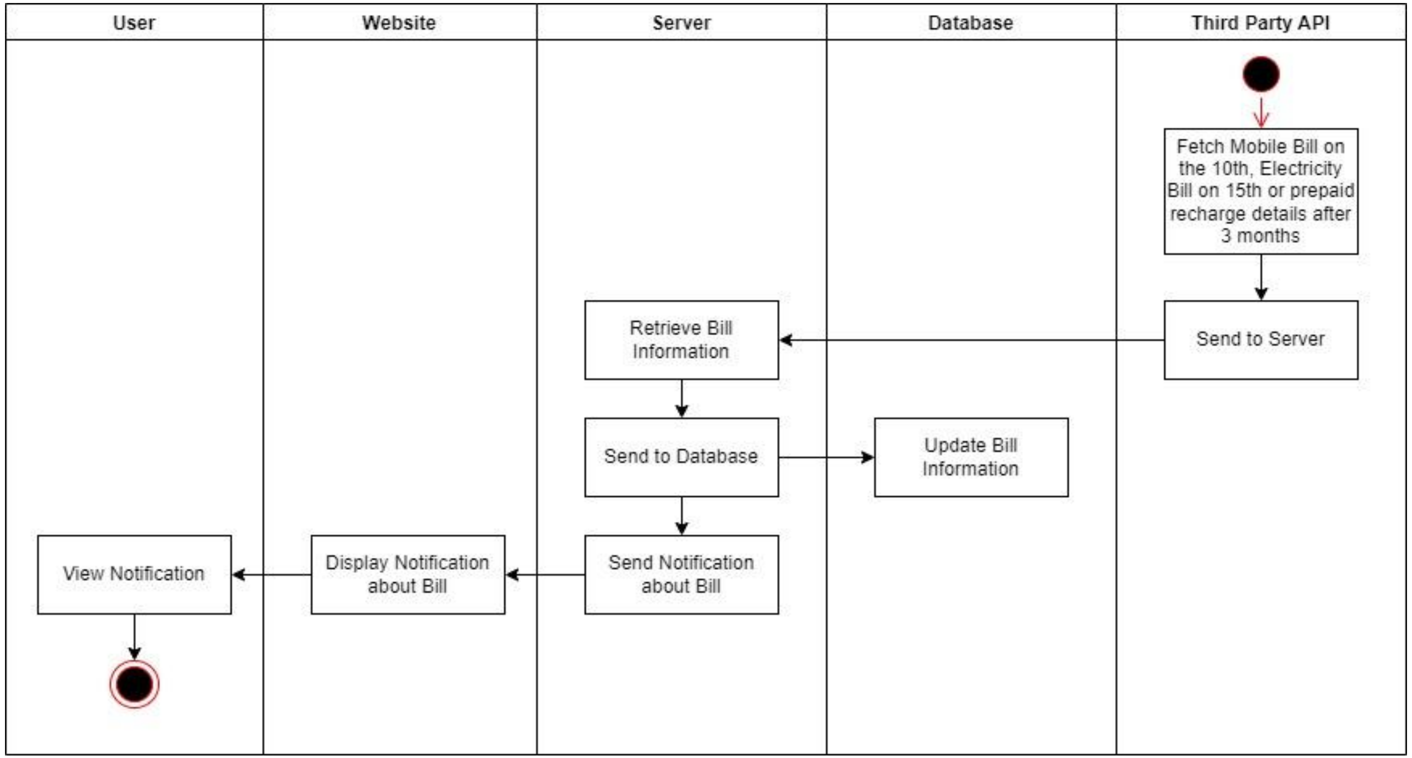


Use Case Diagram

Sequence Diagram



User
Website
Server
Payment API
3rd Party API
Database
Enter Registration Details
Send User Account Details
Insert User Account Details
Enter Login Details
Send User Login Details
Encrypt Login Data
Send Encrypted Login Data
alt
User Data is Correct
Fetch User Account Details
Send Response
Display User Dashboard
User Data is Incorrect
Send Response
Display Invalid login
Retrieve Bill Information
Send Response
Send Query
Fetch Bill Information
Send Response
Display Bill Information
Send Updated UserInfo
Send Response
Update User Account Details
Delete User Details as Admin
Send Deletion info
Delete User Account Details
FetchBillDetails() {The API is called on specific days of a month on which bill gets generated)
Send Bill Details
Update DB with bill details
SendBillDetails()
DisplayBillDetails()
Send Notification Message to user
Make Payment
Pay(CustomerId,BillId)
Redirect to Payment Gateway providing customer details and bill details
alt
Payment Successfull
Update db if payment is successfull
Send Successfull Response
Send Successful Response
Notify 3rd Party
Payment Failure
Send Failure Response
Send Failure Response
View Transaction History
viewTransaction(customerId)
Fetch Transaction details of a customer from db
Send Transaction details to server
Send Data to client browser
Display Transaction History


User
Website
Server
Database
Third Party API
Fetch Mobile Bill on the 10th, Electricity Bill on 15th or prepaid recharge details after 3 months
Send to Server
Retrieve Bill Information
Send to Database
Update Bill Information
Send Notification about Bill
Display Notification about Bill
View Notification

Activity Diagram- Notifications

Modules and Components

1) Customer Management Module:

- Description: This module manages customer-related data and actions.
- Components:
  - Customer Registration: Allows customers to onboard and create accounts.
  - User Authentication: Ensures secure access to customer accounts.
  - Customer Data Storage: Stores customer details, such as name, mobile number, and contact information.

2) Utility Provider Management Module:

- Description: This module handles the registration and management of utility providers and their API details.
- Components:
  - Provider Registration: Allows administrators to register new utility providers.
  - Provider API Configuration: Stores API credentials and details for utility providers.
  - Provider Management: Provides tools for admins to manage registered providers.

3) Bill Management Module:

- Description: This module manages the retrieval and storage of utility bills.
- Components:
  - Bill Retrieval: Communicates with utility provider APIs to retrieve customer bills.
  - Bill Storage: Stores bill details, including bill amount, due date, and customer reference.
  - Scheduling: Implements scheduling for bill retrieval based on provider billing cycles.

4) Notification Module:

- Description: This module handles bill notifications and reminders.
- Components:
    - Notification Scheduler: Schedules reminders for bill payments based on billing cycle and customer preferences.
    - Notification Sender: Sends notifications to customers via email, SMS, or app notifications.
    - Notification Templates: Manages templates for different types of notifications.

5) Payment Processing Module:

- Description: This module handles customer payments for utility bills.
- Components:
    - Payment Gateway Integration: Interfaces with third-party payment gateways for processing payments.
    - Payment Status Tracking: Tracks and records payment statuses for each bill.
    - Payment History: Provides customers with a payment history log.

6) Frontend Module:

- Description: The frontend module is responsible for the user interface and customer interactions.
- Components:
    - React.js Application: Develops the user interface using React.js.
    - User Dashboard: Provides customers with bill details, payment options, and notification settings.
    - Admin Dashboard: Offers administrators tools for managing customers, providers, and system configurations.

Services

1. **RESTful or GraphQL APIs:**
    - Description: These services facilitate communication between the frontend and backend modules and enable integration with utility providers.
    - Components:
        - Customer API: Manages customer-related operations (registration, authentication).
        - Bill API: Handles bill retrieval, storage, and retrieval for customers.
        - Notification API: Schedules and sends notifications to customers.
        - Payment API: Interfaces with payment gateways for processing payments.
        - Utility Provider API: Communicates with utility provider APIs to retrieve bills.
2. **Message Queue (Apache Kafka):**
    - Description: Apache Kafka enables asynchronous processing for tasks like bill retrieval and notification scheduling.
    - Components:
        - Message Topics: Different Kafka topics for various asynchronous tasks.
        - Kafka Consumers: Modules that consume and process messages from Kafka topics.
3. **Scheduler (Apache Airflow):**
    - Description: Apache Airflow manages scheduled tasks and cron jobs, such as bill retrieval and notification scheduling.
    - Components:
        - Task Schedulers: Define and manage scheduled tasks.
        - Task Executors: Execute scheduled tasks at specified intervals.

4. Database (MongoDB):

- Description: MongoDB stores customer data, utility provider details, bill information, and transaction records.
- Components:
  - Customer Data Storage: MongoDB collections for storing customer information.
  - Bill Storage: Collections for storing bill details.

5. Web Server (Nginx/Apache):

- Description: The web server handles incoming HTTP requests and serves static content for the React.js frontend.
- Components:
  - Reverse Proxy: Routes incoming requests to the appropriate backend components.
  - Static Content Delivery: Serves static assets like JavaScript, CSS, and images.

6. Monitoring and Logging Services:

- Description: These services provide monitoring and logging capabilities for system health and debugging.
- Components:
  - Monitoring Tools (e.g., Prometheus, Grafana): Collect and visualize system metrics.
  - Centralized Logging (e.g., ELK Stack): Collect and analyze logs for troubleshooting.

Proposed Technology Stack

- Backend: Python (Django)
- Database: MongoDB
- Web Framework: Django (Python)
- Message Queue: Apache Kafka for asynchronous processing.
- Frontend: React js
- API Integration: Use RESTful and GraphQL APIs for communication with utility providers.
- Scheduling: Apache Airflow for Cron jobs for bill retrieval and notification scheduling.
- Deployment: Deployment of the System to AWS

Advantages of the Proposed Stack

- Python, Django - It is a high level web framework with extensive libraries which offer features like authentication, ORM and built in admin interfaces making it suitable for building the proposed application.
- MongoDB - It is a NoSQL database that can be highly scalable for systems with potentially large volumes of structured and unstructured data. Since our application should be highly flexible we use MongoDB.
- Apache Kafka - It is a robust choice for asynchronous message processing that can handle large volumes of messages and has real-time data streaming capabilities. Our application uses kafka for real time processing of the bills and send notifications.
- React js - It is a popular and efficient JavaScript library for building user interfaces. We use React js for creating a responsive and user-friendly frontend.
- Apache Airflow - It is a powerful tool for managing scheduled tasks. We use Airflow to schedule API calls to bill providers and sends out notifications. Thus it can be a valuable addition to the stack for automating processes

Non-Functional Aspects

1. Scalability:

- Design for Horizontal Scaling: Choosing technologies and architectures that support horizontal scaling by using load balancers and distributed systems to handle increased load.
- Optimize Database: For MongoDB, proper indexing, sharding, and scaling strategies are in place to handle large datasets efficiently.
- Use Caching: Implementing caching mechanisms using Redis to reduce the load on the database and improve response times.
- Auto-scaling: Configuring our hosting environment AWS Auto Scaling to automatically adjust resources based on demand.

2. Reliability:

- Fault Tolerance: Use load balancing, redundancy, and failover mechanisms to ensure high availability.
- Monitoring and Logging: Implementing comprehensive monitoring and logging using Prometheus to quickly identify and address issues.
- Backup and Disaster Recovery: Regularly backup critical data and have a disaster recovery plan in place to minimize downtime in case of failures.

3.Security

- Data Encryption: Use encryption for data in transit and at rest. Secure sensitive data with strong hashing and encryption algorithms.
- Authentication and Authorization: Implement robust user authentication and authorization mechanisms to control access to sensitive data and features.
- API Security: Secure our RESTful and GraphQL APIs with proper authentication tokens and rate limiting to prevent abuse.
- OWASP Top Ten: Follow OWASP (Open Web Application Security Project) best practices to mitigate common web application vulnerabilities.

4. Performance

- Database Optimization: We have used optimized database queries, proper indexing, and denormalize data when necessary for better query performance.
- Content Delivery Network (CDN): We have used CDNs for static assets like images and scripts to reduce server load and improve load times for users.
- Load Testing: We perform load testing using tools like JMeter or Locust to identify performance bottlenecks and optimize resource allocation.

5. Usability

- User-Centered Design: We conduct user research and usability testing to ensure that the system is user-friendly and intuitive.
- Responsive Design: We ensure that the frontend (React) is responsive and works well on various devices and screen sizes.
- Accessibility: We follow WCAG guidelines to make our application accessible to users with disabilities.

6. Maintainability

- Code Quality: We enforce coding standards and best practices to maintain clean and readable code.
- Documentation: We maintain comprehensive documentation for code, APIs, and system architecture to aid future development and troubleshooting.
- Version Control: We use version control systems (Git) to manage and track changes to the codebase.

7. Compliance

- Regulatory Compliance: We ensure that our system complies with relevant regulations and standards, such as data privacy laws (e.g., GDPR) and industry-specific standards.
- Audit Trails: We have implemented audit trails and logs to track and report on system activities for compliance purposes.

Key Advantages

- Automates bill retrieval and notification, reducing manual effort to large extent.
- Improves bill payment efficiency, reducing late payments.
- Provides a centralized platform for bill management.
- Sends out bill notifications to the users.
- Users are able to view transaction history.
- Alert users when bills are overdue.

Risk

1) Third-Party API Dependency:

- **Risk:** The system relies on third-party utility provider APIs for bill retrieval. If these APIs change or become unavailable, it can disrupt bill retrieval and notification processes.
- **Mitigation:** Regularly monitor and communicate with utility providers to stay updated on API changes. Implement error handling and fallback mechanisms to gracefully handle API issues.

2) Security Risks:

- **Risk**: Storing customer data and payment information poses security risks. If not properly secured, the system could be vulnerable to data breaches or unauthorized access.
- **Mitigation**: Implement strong encryption for data in transit and at rest. Follow security best practices for user authentication, authorization, and auditing. Regularly conduct security audits and penetration testing.

3) Integration Challenges:

- **Risk:** Integrating with various utility providers may be complex due to differences in APIs, data formats, and authentication methods.
- **Mitigation:** Create adaptable integration modules that can handle diverse APIs. Develop robust error-handling mechanisms to manage integration failures.

4) Regulatory Compliance:

- **Risk:** Ensuring compliance with data privacy laws (e.g., GDPR) and other industry-specific regulations can be complex, particularly when handling customer data and payments.
- **Mitigation:** Thoroughly research and understand relevant regulations. Implement necessary data protection measures, consent mechanisms, and auditing to meet compliance requirements.

Alternate Approaches

1. **Blockchain-Based Solution**:
- Implementing certain aspects of the system on a blockchain can enhance security, transparency and trust. For instance, payment transactions can be recorded on a blockchain for immutability.
2. **Progressive Web App(PWA)**:
- Build a Progressive Web App that works both online and offline. PWAs offers a responsive, app-like user experience on web browsers and can be easily installed on mobile devices.
3. **Edge Computing**:
- Leverage edge computing for real-time data processing and low-latency interactions, especially useful for handling notifications and quick responses to customer actions.
4. **Microservices Architecture:**
- Decompose the system into smaller, independent microservices, each responsible for a specific function(e.g., customer management, billing,payments). This approach allows for greater scalability, maintainability and flexibility.

Deployment Architecture

1. **Web server layer:**
- Use a load balancer to distribute incoming requests across multiple web server instances for redundancy and load balancing. Enable caching for static assets to reduce server load. Nginx or Apache HTTP Server acts as a reverse proxy and serves static content.

2. **Application server layer:**
- Deploy multiple application server instances to ensure high availability and distribute the load.
- Configure Apache Kafka to handle message processing asynchronously.
- Monitor and manage cron jobs and scheduled tasks with Apache Airflow.

3. **Database layer:**
- Set up MongoDB replica sets for data redundancy and high availability.
- Implement proper indexing and sharding strategies for scalability.

4. **Message Queue layer:**
- Configure Kafka topics and partitions to optimize message processing.
- Monitor Kafka cluster health and performance.

5. Payment Gateway Integration:

- Ensure secure communication with the payment gateway using encryption (e.g., SSL/TLS).
- Implement error handling and transaction logging for payment processing.

6. Monitoring and Logging:

- Set up alerts and notifications for critical events and performance thresholds.
- Ensure logs are retained and secured for compliance and auditing purposes.

7. Security Layer:

- Regularly update security measures and apply patches.
- Conduct security audits and penetration testing to identify and mitigate vulnerabilities.
- This deployment architecture is a high-level representation of how the various components of the Utility Bills Notification and Payment System can be deployed for reliability, scalability, and security. Depending on your specific requirements and constraints, you may need to further refine and customize this architecture to meet the needs of your project.

Thank You