**Customer Management System Documentation**

## **1. Introduction**

### **1.1 Project Overview**

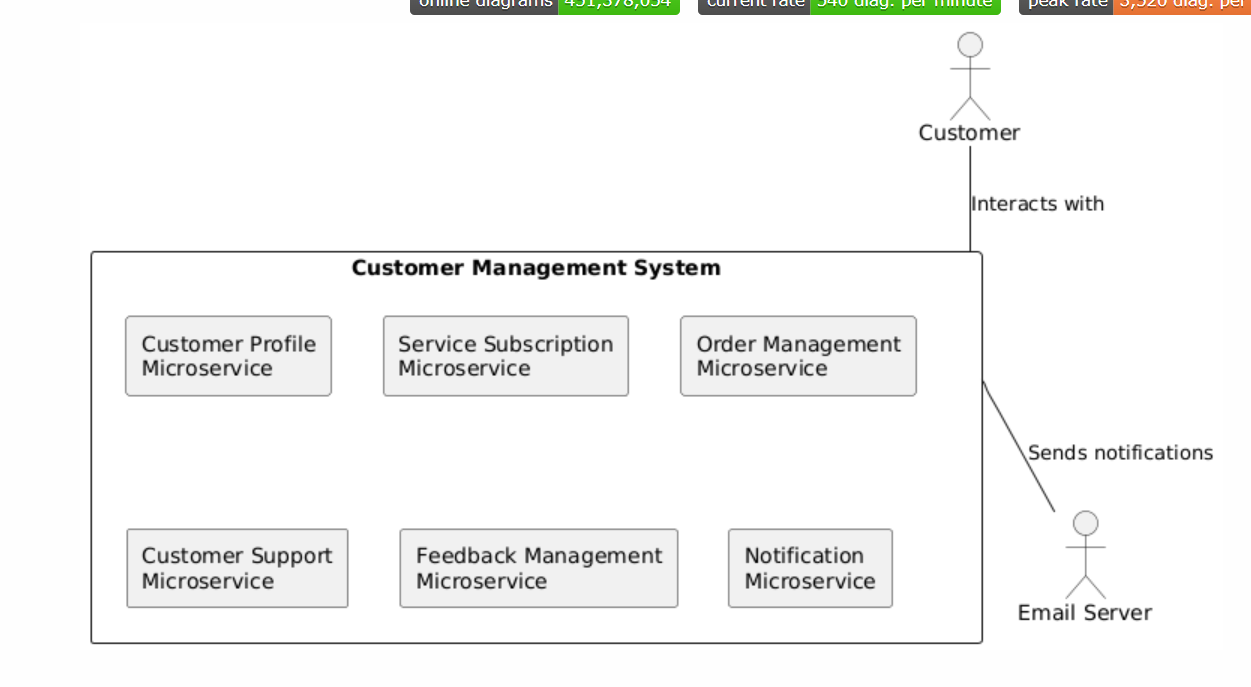
The Customer Management System is designed to manage and optimize customer interactions through a suite of microservices. This system includes components for managing customer profiles, service subscriptions, orders, customer support, feedback, and notifications. The goal is to provide a scalable, modular, and efficient architecture that ensures a high-quality customer experience.

### **1.2 Objectives**

* Centralize customer data management.
* Enable flexible service subscriptions and plan management.
* Streamline order processing and tracking.
* Facilitate customer support and feedback collection.
* Automate notifications and updates across services.

### **1.3 Stakeholders**

* **Project Manager:** [Name]
* **Lead Developer:** [Name]
* **System Architect:** [Name]
* **Quality Assurance Lead:** [Name]
* **Operations Manager:** [Name]



### **Table of Contents**

**Introduction**

* 1. Overview
  2. Technologies Used

**Customer Profile Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Service Subscription Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Order Management Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Customer Support Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Feedback Management Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Notification Microservice**

* 1. Definition
  2. Database Schema
  3. Service Endpoints
  4. Integration Points

**Integration and Communication**

* 1. API Gateway
  2. Inter-Service Communication
  3. Security

**Database Design**

* 1. PostgreSQL Schema
  2. Entity Relationships

**Conclusion**

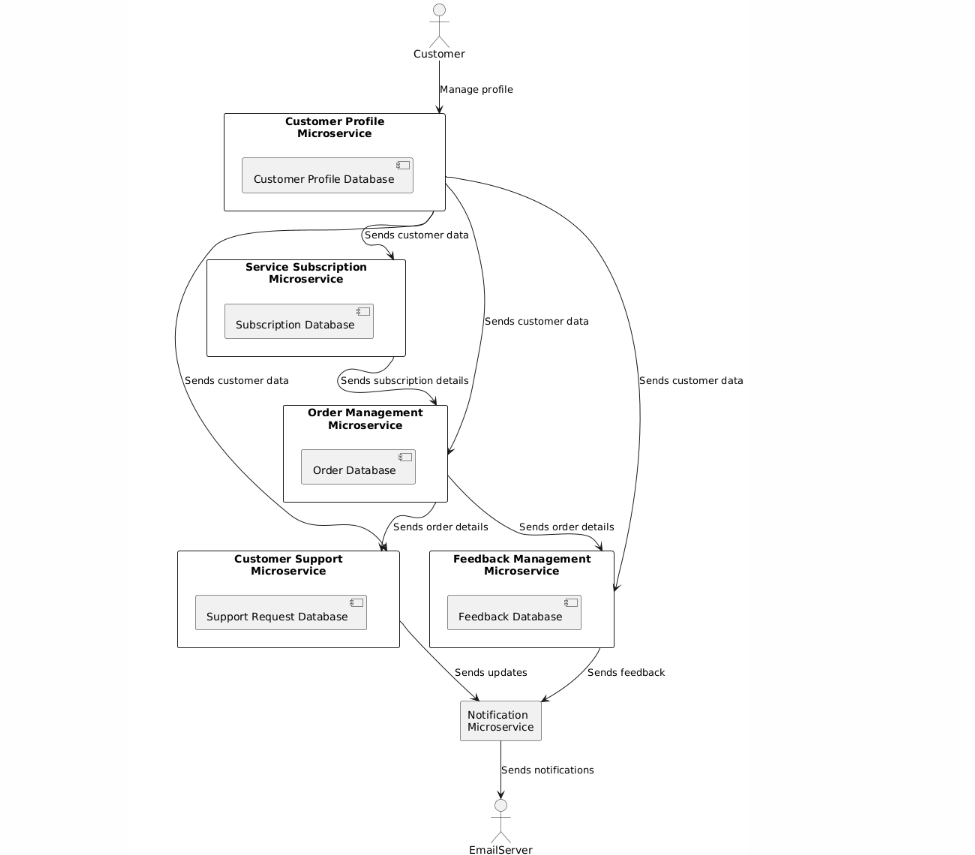
* 1. Summary
  2. Future Work

#### ****Overview****

The customer management system is designed to handle various aspects of customer interaction and service management. This system is built using a microservice architecture to ensure scalability, maintainability, and modularity. The following sections detail each microservice, its database schema, service endpoints, and integration points.

#### ****Technologies Used****

* **Spring Boot**: Framework for building microservices with ease.
* **Spring Data JPA**: Provides data access and manipulation using JPA.
* **PostgreSQL**: Relational database for persistent data storage.



### **1. Customer Profile Microservice**

#### ****Definition****

#### The Customer Profile Microservice is a central component of the Customer Management System, responsible for managing and maintaining detailed information about customers. As the foundational service, it plays a critical role in providing consistent and reliable customer data to other microservices within the system.

#### ****Database Schema****

sql

Copy code

CREATE TABLE customer\_profile (

customer\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone\_number VARCHAR(20),

address TEXT

);

#### ****Service Endpoints****

* **GET /customers**: Retrieve all customer profiles.
* **GET /customers/{id}**: Retrieve a specific customer profile.
* **POST /customers**: Create a new customer profile.
* **PUT /customers/{id}**: Update an existing customer profile.
* **DELETE /customers/{id}**: Delete a customer profile.

#### ****Integration Points****

* **Service Subscription Microservice**: Customer ID is used as a foreign key.
* **Order Management Microservice**: Customer ID is used as a foreign key.
* **Customer Support Microservice**: Customer ID is used as a foreign key.
* **Feedback Management Microservice**: Customer ID is used as a foreign key.
* **Notification Microservice**: Notifies based on changes in customer profiles.

### **2. Service Subscription Microservice**

#### ****Definition****

#### The Service Subscription Microservice is a key component of the Customer Management System, responsible for managing customer subscriptions to various service plans. This microservice handles the selection of service plans, subscription management, and subscription periods, ensuring that customers have access to the services they require.

#### ****Database Schema****

sql

Copy code

CREATE TABLE service\_subscription (

subscription\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customer\_profile(customer\_id),

plan VARCHAR(100) NOT NULL,

time\_period INTERVAL,

FOREIGN KEY (customer\_id) REFERENCES customer\_profile(customer\_id)

);

#### ****Service Endpoints****

* **GET /subscriptions**: Retrieve all subscriptions.
* **GET /subscriptions/{id}**: Retrieve a specific subscription.
* **POST /subscriptions**: Create a new subscription.
* **PUT /subscriptions/{id}**: Update an existing subscription.
* **DELETE /subscriptions/{id}**: Delete a subscription.

#### ****Integration Points****

* **Customer Profile Microservice**: References customer profile data.
* **Order Management Microservice**: Uses subscription plan information.
* **Feedback Management Microservice**: Indirectly related through subscription impact.

### **3. Order Management Microservice**

#### ****Definition****

Manages customer orders, including order creation and tracking. Each order is linked to a customer and their service subscription.The Order Management Microservice is a crucial element of the Customer Management System, dedicated to handling customer orders. It is responsible for creating, tracking, and managing orders throughout their lifecycle. This microservice links each order to a customer and their associated service subscription, ensuring a comprehensive view of order-related information.

#### ****Database Schema****

sql

Copy code

CREATE TABLE order\_management (

order\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customer\_profile(customer\_id),

plan VARCHAR(100) REFERENCES service\_subscription(plan),

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES customer\_profile(customer\_id),

FOREIGN KEY (plan) REFERENCES service\_subscription(plan)

);

#### ****Service Endpoints****

* **GET /orders**: Retrieve all orders.
* **GET /orders/{id}**: Retrieve a specific order.
* **POST /orders**: Create a new order.
* **PUT /orders/{id}**: Update an existing order.
* **DELETE /orders/{id}**: Cancel an order.

#### ****Integration Points****

* **Customer Profile Microservice**: Links orders to customer profiles.
* **Service Subscription Microservice**: Associates orders with subscription plans.
* **Feedback Management Microservice**: Collects feedback related to orders.\

### **4. Customer Support Microservice**

#### ****Definition****

Handles customer support requests and tickets. Tracks issues and their resolution status.The Customer Support Microservice is a critical component of the Customer Management System, designed to handle and manage customer support requests and tickets. This microservice is responsible for tracking customer issues, monitoring their resolution status, and ensuring effective resolution of support queries.

#### ****Database Schema****

sql

Copy code

CREATE TABLE customer\_support (

request\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customer\_profile(customer\_id),

issue TEXT NOT NULL,

status VARCHAR(50) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES customer\_profile(customer\_id)

);

#### ****Service Endpoints****

* **GET /support/requests**: Retrieve all support requests.
* **GET /support/requests/{id}**: Retrieve a specific support request.
* **POST /support/requests**: Create a new support request.
* **PUT /support/requests/{id}**: Update an existing support request.
* **DELETE /support/requests/{id}**: Resolve or delete a support request.

#### ****Integration Points****

* **Customer Profile Microservice**: Links support requests to customer profiles.
* **Notification Microservice**: Sends notifications about request status changes.
* **Order Management Microservice**: May reference orders related to support requests.

### **5. Feedback Management Microservice**

#### ****Definition****

Collects and manages feedback from customers. Each feedback entry is linked to a customer profile.The Feedback Management Microservice is an essential component of the Customer Management System, tasked with collecting, managing, and analyzing feedback from customers. This microservice ensures that each feedback entry is linked to a customer profile, allowing for effective tracking and resolution of customer concerns and suggestions.

#### ****Database Schema****

sql

Copy code

CREATE TABLE feedback\_management (

feedback\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customer\_profile(customer\_id),

feedback\_text TEXT NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES customer\_profile(customer\_id)

);

#### ****Service Endpoints****

* **GET /feedback**: Retrieve all feedback.
* **GET /feedback/{id}**: Retrieve specific feedback.
* **POST /feedback**: Submit new feedback.
* **PUT /feedback/{id}**: Update feedback.
* **DELETE /feedback/{id}**: Delete feedback.

#### ****Integration Points****

* **Customer Profile Microservice**: Associates feedback with customer profiles.
* **Order Management Microservice**: Collects feedback related to orders.
* **Customer Support Microservice**: May use feedback for support resolution.

### **6. Notification Microservice**

#### ****Definition****

Handles notifications related to various events across the system. Utilizes JMS for email notifications. The Notification Microservice is a key component of the Customer Management System, designed to handle and manage notifications related to various events across the system. It leverages Java Messaging Service (JMS) for sending email notifications, ensuring timely and efficient communication with customers and internal stakeholders.

#### ****Database Schema****

(No dedicated database schema as notifications are primarily managed through JMS and other messaging systems.)

#### ****Service Endpoints****

* **POST /notifications**: Send a new notification.
* **GET /notifications**: Retrieve notifications (if applicable).

#### ****Integration Points****

* **Customer Profile Microservice**: Sends notifications for profile changes.
* **Order Management Microservice**: Notifies order status updates.
* **Customer Support Microservice**: Sends updates on support request statuses.
* **Feedback Management Microservice**: Notifies feedback submission.

### **Integration and Communication**

The integration and communication strategy for the Customer Management System is crucial for ensuring seamless interaction between various microservices and maintaining system security and efficiency. This section elaborates on the components involved in managing integration and communication, including the API Gateway, inter-service communication methods, and security considerations.

#### ****API Gateway****

An API Gateway is a critical component in the microservice architecture that acts as a single entry point for all client requests. Its primary functions include:

**Routing**: The API Gateway directs incoming requests to the appropriate microservice based on the request path and parameters. This routing mechanism abstracts the complexity of service locations and allows clients to interact with the system through a unified interface.

**Load Balancing**: The API Gateway distributes incoming requests across multiple instances of a microservice. This ensures that no single instance becomes a bottleneck, improving the system’s scalability and reliability. Load balancing also helps in maintaining high availability and optimal performance.

**Security**: The API Gateway enforces security policies such as authentication and authorization before routing requests to microservices. It can integrate with security services to validate tokens, manage user access, and enforce security protocols.

**Rate Limiting and Throttling**: To prevent abuse and ensure fair usage, the API Gateway can limit the number of requests a client can make within a specific time frame. This protects microservices from being overwhelmed by excessive requests.

**Caching**: The API Gateway can cache responses from microservices to reduce load and latency, improving the overall performance of the system.

**Monitoring and Logging**: The API Gateway collects metrics and logs related to incoming requests, response times, and error rates. This data is essential for monitoring system health, diagnosing issues, and optimizing performance.

### **Database Design**

#### ****PostgreSQL Schema****

All microservices utilize PostgreSQL for persistent storage. Relationships are defined through foreign keys to maintain referential integrity.

#### ****Entity Relationships****

* **Customer Profile** is central, with other microservices referencing it via customer\_id.
* **Service Subscription** and **Order Management** use subscription plans and customer IDs to link their data.
* **Customer Support** and **Feedback Management** reference customer profiles to track and manage requests and feedback.

### **Conclusion**

The Customer Management System designed in this document is a comprehensive solution to handle various aspects of customer interactions and service management using a microservice architecture. This architecture provides flexibility, scalability, and maintainability, ensuring that the system can grow and adapt to changing requirements. Below is an expanded conclusion highlighting the key aspects and future considerations for the system.

#### ****Summary****

The customer management system is composed of the following microservices:

1. **Customer Profile Microservice**: Manages customer information such as personal details and contact information. This microservice serves as the central data source for other services, ensuring consistency and integrity across the system.
2. **Service Subscription Microservice**: Handles customer subscriptions to different service plans, including subscription management and time periods. This service interacts with the Customer Profile Microservice to link subscriptions with customer profiles.
3. **Order Management Microservice**: Manages customer orders, tracking order details and ensuring that orders are linked to both customer profiles and service subscriptions. This service also manages order durations and related data.
4. **Customer Support Microservice**: Handles customer support requests and tracks issues and their resolutions. This microservice ensures that customer queries and support needs are addressed efficiently.
5. **Feedback Management Microservice**: Collects and manages customer feedback, linking feedback entries to customer profiles. This service is critical for understanding customer satisfaction and areas for improvement.
6. **Notification Microservice**: Manages notifications and communications with customers, using JMS to send email notifications about various system events (e.g., profile updates, order status changes, support request updates).

Each microservice is designed to perform a specific function and interact with other services to provide a seamless user experience. The system uses PostgreSQL for data persistence, with each microservice having its dedicated database schema to ensure data isolation and integrity.

#### ****Integration and Communication****

Integration between microservices is achieved through RESTful APIs, which allows for clear and well-defined communication channels. Each microservice is responsible for its own data and operations but relies on the Customer Profile Microservice as the core reference point for customer-related data. Notifications are handled by a dedicated Notification Microservice that utilizes JMS for asynchronous communication, ensuring that customers receive timely updates about their interactions with the system.

#### ****Database Design****

The use of PostgreSQL for database management ensures robust and scalable data storage. Each microservice has its own database schema, allowing for clear separation of concerns and reducing the risk of data conflicts. Foreign key relationships are established where necessary to maintain referential integrity, particularly linking customer IDs across different services.

#### ****Scalability and Maintainability****

The microservice architecture is designed with scalability in mind. Each microservice can be scaled independently based on demand, which ensures that the system can handle increasing loads and accommodate new features without impacting existing functionality. The modular nature of microservices also enhances maintainability, as each service can be developed, deployed, and updated independently.

#### ****Future Considerations****

While the current design addresses the core requirements, there are several areas to consider for future enhancements:

1. **Security**: Implementing robust security measures such as OAuth2 for authentication and authorization, and ensuring data protection through encryption and secure communication protocols.
2. **Monitoring and Logging**: Integrating monitoring and logging solutions to track system performance, identify issues, and ensure that the system operates smoothly.
3. **API Gateway**: Introducing an API Gateway to manage routing, load balancing, and security across microservices, which can simplify integration and improve system reliability.
4. **Data Analytics**: Adding analytics capabilities to gain insights from customer interactions, subscription data, and feedback. This can help in making data-driven decisions and enhancing the overall customer experience.
5. **Extensibility**: Designing the system with extensibility in mind to accommodate new features and services as the business grows and evolves.

By addressing these considerations, the customer management system can continue to provide value and adapt to future needs, ensuring a positive and efficient experience for customers and administrators alike.