**Vilnius University**

**Faculty of Mathematics and Informatics**

# Algorithm Brothers and Sister

*Software architecture*

**Dominykas Dobkevičius**
**Gintarė Keraitė**
**Joris Lisas**
**Kajus Kutelis**
**Lukas Šimonėlis**

2024

# Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to provide a detailed technical specification and design for a Software as a Service (SaaS) solution catering to the hospitality and beauty industry, focusing on managing orders, reservations, services, products, and user management. This system will facilitate the automation of operations within small and medium businesses, ensuring streamlined workflows for employees and enhanced management for business owners.

## 1.2 Scope

The scope of the system includes order management, reservations, services, and products for businesses in the catering and beauty sectors. The system supports multiple roles, including employees, business owners, and super admins, ensuring that only authorized users can access and perform operations. The functionality also extends to handling tax calculations, discounts, and inventory management. Future iterations will involve the integration of card payment providers such as Stripe or Elavon.

## 1.3 Definitions, Acronyms, and Abbreviations

- **SaaS:** Software as a Service
- **API:** Application Programming Interface
- **UI:** User Interface
- **CRUD:** Create, Read, Update, Delete (basic operations for managing data)
- **Stripe/Elavon:** Payment service providers for online card payments
- **Reservation:** A booking for a service made by the customer
- **Order:** A record of a transaction created by an employee
- **Product:** An item available for purchase in the business
- **Service:** A specific task or activity provided to customers (e.g., haircut, dining)

## 1.4 Overview

This document presents the design and technical specification of a Software as a Service (SaaS) system intended for small and medium businesses in the catering and beauty sectors. The system supports core business functions, such as order and reservation management, inventory control, and user role-based access. It aims to streamline operations, improve efficiency, and enhance service delivery by providing a centralized platform for managing day-to-day activities and ensuring secure user access.

# 2 Data Model

## 2.1 Class Diagram

The class diagram represents the core components of the SaaS solution designed for the hospitality and beauty industry. It includes entities that are essential for managing orders, reservations, products, services, and user roles. Each entity in the diagram is carefully structured to facilitate streamlined operations, efficient data handling, and secure access control, aligning with the overall goal of providing a centralized platform for small and medium businesses.

- **Entity Structure:** The primary entities in the class diagram include objects such as Order, Reservation, Product, Service, and User. Each entity is designed with attributes that define its properties, including serialized primary keys for unique identification. This approach simplifies data retrieval, ensures integrity, and enables easy management of relationships between different entities.

- **Archiving Strategy:** The archiving strategy leverages the isDeleted: bool field to manage soft deletions. Instead of physically removing records from the database, setting this flag to true marks the entity as deleted while preserving historical data for auditing or future reference. This approach ensures that data integrity is maintained, and the system can efficiently handle inactive or obsolete records. By using soft deletion, the system can track changes and restore records when necessary, which is crucial for maintaining comprehensive data logs and reducing the risk of accidental data loss.

- **Relationship Management:** The class diagram employs enums such as UserRoles and Status to categorize user types and state information, respectively. Proper use of these enums helps maintain clear and structured relationships between entities, improving query performance and data handling. Enums offer a standardized way to represent fixed sets of values, reducing the chance of inconsistencies and simplifying data validation within the system. This approach ensures that data remains consistent across all entities, facilitating easier data management and enforcing business rules at the application level.

- **Business Logic Integration:** All calculations and business logic operations are kept within the application's code rather than directly within the database. This decision enhances system flexibility by allowing updates to logic without requiring structural changes to the database. The attributes and relationships defined in the class diagram are leveraged by the application's logic to execute operations like order processing, inventory adjustments, and reservation management.

- **Design Considerations:** The class diagram emphasizes modularity and scalability, making it easier to extend the system in future iterations. The structure supports core

functions such as handling user roles, processing payments, managing services, and conducting transactions efficiently. Additionally, the indexing strategy, particularly for archived entities, ensures that data retrieval remains swift even as the dataset grows.

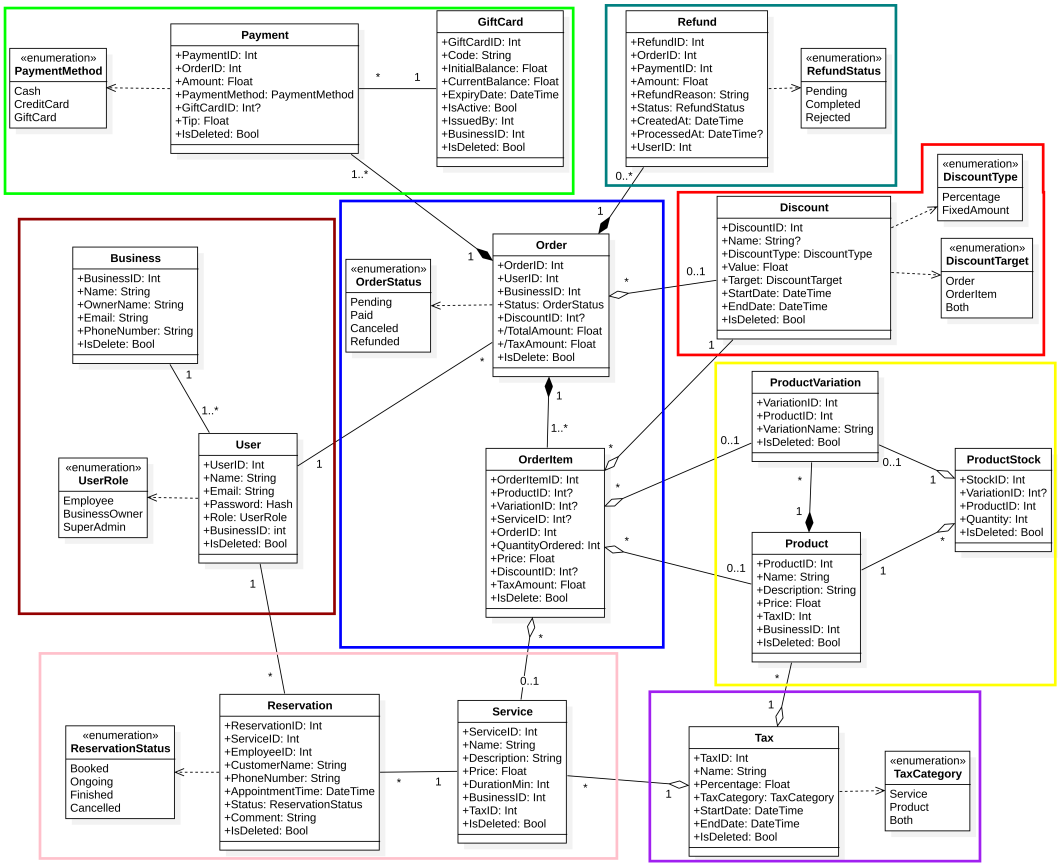


Figure 1: Class diagram

## 2.2 Clarification of Sections

### 2.2.1 Discounts (Red)

Discounts in this system can be applied at two levels: to an entire order or to individual items within the order. The `Discount` entity contains fields that specify the type of discount (percentage or fixed amount) and the target for the discount, which is controlled by the `DiscountTarget` enum. This enum allows a discount to be applied either to the whole order (`Order`) or to specific items (`OrderItem`).

- **Order-level discounts** affect the entire total of the order and are linked directly to the `Order` entity.
- **OrderItem-level discounts** are applied to individual items and are associated with `OrderItem` entities, allowing for more granular control.

The DiscountType field defines whether the discount is a percentage or a fixed amount, further refining how the discount is calculated.

### 2.2.2 Products and Product Variations (Yellow)

The system supports products both with and without variations. A Product can have multiple ProductVariations, such as different sizes or colors, which are tracked through the ProductVariation entity. Each ProductVariation can have its own stock levels, managed by the ProductStock entity.

- If a product has no variations, ProductStock will reference only the ProductID, and VariationID will remain null.
- If a product has variations, ProductStock will reference both ProductID and VariationID to track the quantity of each variation separately.

This system allows flexibility for managing products with and without variations while keeping inventory management consistent.

### 2.2.3 Orders and OrderItems (Blue)

The Order is the central entity for tracking transactions in the system. Each Order is associated with one or more OrderItem entities, which represent the specific products or services that were part of the order.

- OrderItem can reference both Product and Service, meaning the same OrderItem entity is used for tracking sales of physical products as well as services rendered.
- OrderItem also allows for the application of specific discounts (through a DiscountID) at the item level, making it possible to apply targeted discounts to specific items without affecting the entire order.

### 2.2.4 Reservations and Services (Pink)

Reservation is designed for booking services that a business offers. Each Reservation is linked to a Service and represents a specific time slot when that service will be rendered.

- Reservation tracks information such as the customer's name, the scheduled time, and the status of the reservation (e.g., booked, ongoing, finished, canceled).
- Reservations don't include totals or discounts since services can be tied directly to an Order, which handles all payment and discount information.

### 2.2.5 Payments and GiftCards (Green)

Payments in the system can be made through various methods, including cash, credit cards, and gift cards. The `PaymentMethod` enum defines the different types of payment methods available.

- `GiftCard` is treated as a form of payment and is linked to `Payment`. A gift card has its own attributes such as initial balance, current balance, and expiration date. When a gift card is used, the corresponding balance is deducted from the `GiftCard` entity.
- Payments are linked to orders via the `Payment` entity, which tracks the total amount paid, any tips, and the payment method used. This makes it easy to track and reconcile payments against specific orders.

### 2.2.6 Refunds (Teal)

`Refund` records are created when an order is refunded, either partially or fully. Each `Refund` is associated with both an `Order` and a `Payment` (the payment being refunded).

- The `RefundStatus` enum helps to track the progress of a refund, with possible statuses including pending, completed, or rejected.
- The system supports multiple refunds per order if needed, and the `Refund` entity tracks how much of the original payment has been refunded.

### 2.2.7 Taxes and Tax Categories (Purple)

Taxes are managed through the `Tax` entity, which allows businesses to define different tax rates for services and products.

- The `TaxCategory` enum helps classify whether the tax applies to services, products, or both. Each tax entity tracks the percentage of tax applied and the validity period (start and end dates), allowing businesses to update tax rates as needed.
- `TaxID` is referenced in both the `Product` and `Service` entities to specify which tax category applies to them. This ensures the correct tax is applied when an order is created.

# 3 Business Flows

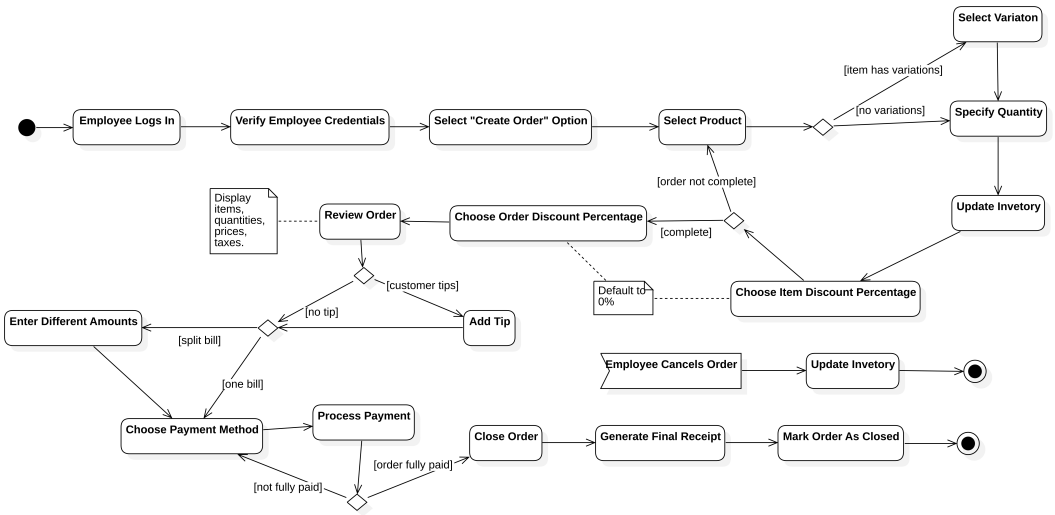


Figure 2: Order (Item) Creation

1. **Employee Logs In**: The process starts when the employee logs into the POS system.
2. **Verify Employee Credentials**: The system verifies the employee's credentials to ensure they are authorized to use the system.
3. **Select "Create Order" Option**: After a successful login, the employee selects the option to create a new order.
4. **Select Product**: The employee begins the order by selecting the product. At this stage, if the item has variations (such as size or color), the system prompts the employee to select a specific variation.
    - **Specify Quantity**: After selecting the product or variation, the employee specifies the quantity of the item.
    - **Update Inventory**: The inventory is updated automatically based on the items and quantities selected.
5. **Choose Item Discount Percentage**: If the item is eligible for a discount, the employee can choose a specific discount percentage. If no discount is applied, the system defaults to a 0% discount.
6. **Review Order**: Once all items are selected, the system displays a summary of the order, including item details, quantities, prices, and taxes.
7. **Add Tip (Optional)**: At this stage, if the customer chooses to tip, the employee can add a tip to the order. If no tip is given, the process continues without adding one.

8. **Choose Order Discount Percentage (Optional)**: The employee may also apply a discount to the entire order. If no discount is applied, the system defaults to a 0% discount.

9. **Choose Payment Method**: After reviewing the order, the employee proceeds to the payment step, where they choose the payment method.

    - **Enter Different Amounts (Optional)**: If the customer wants to split the bill, the employee can enter different amounts based on the split. Otherwise, they proceed with one payment.

10. **Process Payment**: The system processes the payment. If the order is not fully paid, the process may need to be repeated until full payment is received.

11. **Close Order**: Once the payment is completed, the order is closed. The system marks the order as fully paid and completed.

12. **Generate Final Receipt**: A final receipt is generated, summarizing all items, payments, tips, and discounts.

13. **Mark Order As Closed**: Finally, the order is marked as closed in the system, concluding the transaction.

14. **Employee Cancels Order (Optional)**: If necessary, the employee can cancel the order at any point before closing it. When an order is canceled, the inventory is updated to reflect the change.

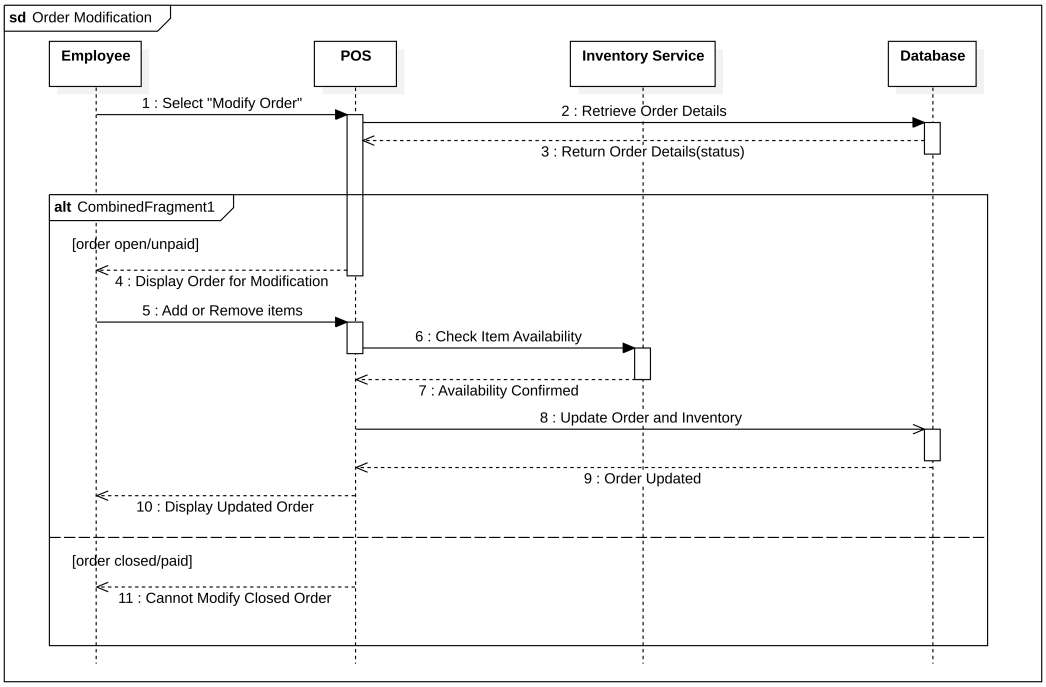


Figure 3: Order Modification

1. **Select "Modify Order"**: The employee initiates the modification process by selecting the “Modify Order" option in the POS system.
2. **Retrieve Order Details**: The POS system sends a request to the inventory service to retrieve the order details, including the current status (open/unpaid or closed/paid).
3. **Return Order Details (Status)**: The inventory service returns the requested order details, including the order’s current status.
4. **Display Order for Modification**: If the order is open and unpaid, the POS system displays the order to the employee for modification.
5. **Add or Remove Items**: The employee makes changes to the order, such as adding or removing items.
6. **Check Item Availability**: Once the changes are made, the POS system checks the availability of the items by communicating with the inventory service.
7. **Availability Confirmed**: The inventory service confirms the availability of the requested items.
8. **Update Order and Inventory**: The POS system updates the order based on the modifications, and the inventory is adjusted accordingly.
9. **Order Updated**: The order is successfully updated in the system, and the inventory levels are adjusted.

10. **Display Updated Order**: The updated order is displayed to the employee.

11. **Cannot Modify Closed Order**: If the order is closed and fully paid, the system prevents any further modifications, notifying the employee that the order cannot be modified.

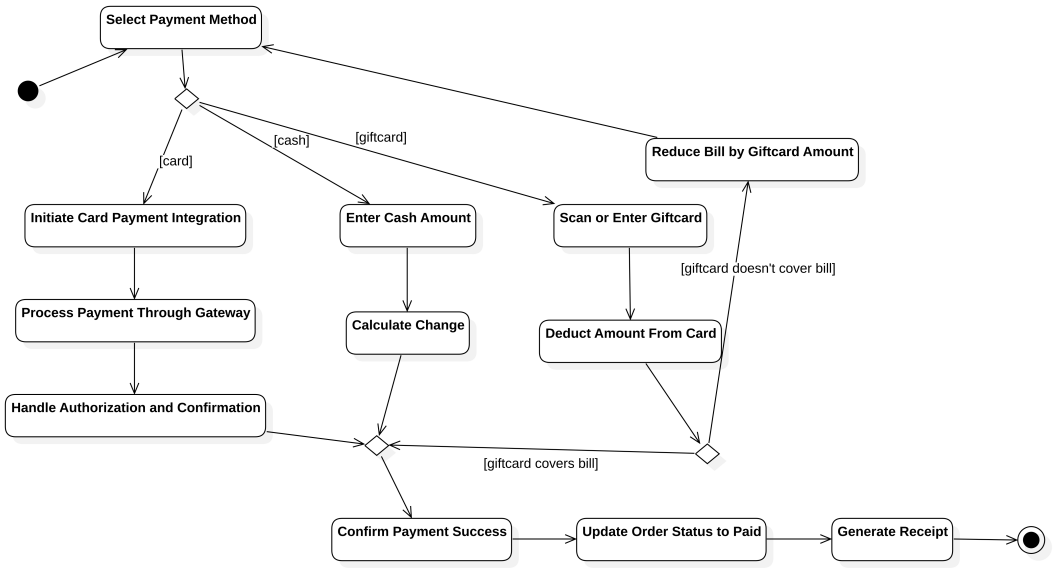


Figure 4: Payment Flow

1. **Select Payment Method**: The employee selects a payment method to begin the transaction. The options include cash, card, or gift card.

2. **Gift Card Payment**:

   - If the customer is paying with a gift card, the employee can either scan or manually enter the gift card details.
   - **Deduct Amount From Card**: The system deducts the applicable amount from the gift card.
     - **Gift Card Covers Bill**: If the gift card fully covers the bill, the system marks the payment as successful.
     - **Gift Card Doesn't Cover Bill**: If the gift card does not cover the full amount, the remaining balance needs to be paid using an alternative method (e.g., cash or card).
   - **Reduce Bill by Gift Card Amount**: The bill is adjusted to reflect any payment made using the gift card.

3. **Cash Payment**:

   - If the customer is paying with cash, the employee enters the amount of cash provided.

- **Calculate Change**: The system calculates the change due to the customer and displays this information.
- **Confirm Payment Success**: The payment is marked as successful once the cash has been accepted, and any necessary change has been calculated.

4. **Card Payment**:
    - If the customer is paying by card, the system initiates the card payment integration.
    - **Process Payment Through Gateway**: The card payment is processed through a payment gateway.
    - **Handle Authorization and Confirmation**: The system handles the authorization process, and once confirmed, the payment is marked as successful.
5. **Confirm Payment Success**: Once the payment has been processed and authorized, the system confirms the success of the payment.
6. **Update Order Status to Paid**: After a successful payment, the system updates the status of the order to “Paid."
7. **Generate Receipt**: The final step is the generation of a receipt, summarizing the completed payment and transaction details.

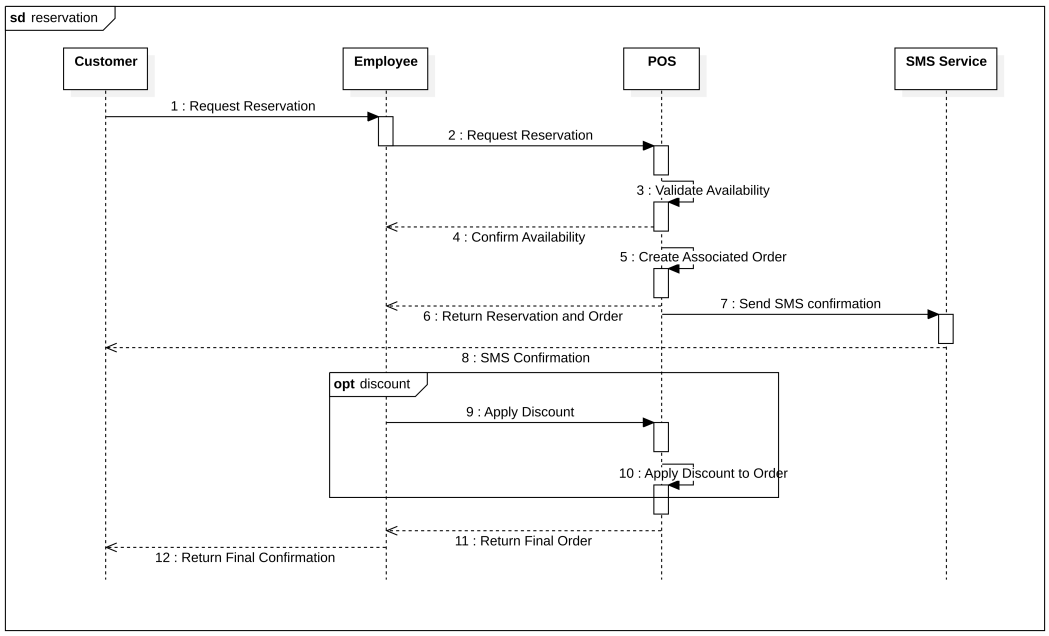


Figure 5: Reservation Creation

1. **Customer Requests Reservation:** The customer initiates the process by requesting a reservation through the employee, either via phone, in-person, or an online interface.

2. **Employee Requests Reservation from POS:** The employee inputs the reservation request into the POS system, specifying the service, time, and other relevant details.

3. **POS Validates Service Availability:** The POS system checks the availability of the requested service at the desired time (assignable employee availability).

4. **POS Confirms Availability:** If the service is available, the POS system confirms the availability to the employee, allowing the process to continue.

5. **POS Creates Associated Order:** The POS system generates an associated order for the reservation. The order tracks the payment information and will eventually include any discounts, taxes, or additional services.

6. **Return Reservation and Order:** The system returns the newly created `ReservationID` and `OrderID` to the employee, who can now proceed with the reservation confirmation.

7. **Send SMS Confirmation:** The POS system communicates with the SMS service, sending a confirmation message to the customer that includes the reservation details such as the time, service, and contact information.

8. **SMS Confirmation Sent:** The SMS service acknowledges that the confirmation message has been successfully sent to the customer.

9. **Optional: Apply Discount:** If the customer provides a discount code or if a promotion is active, the employee can apply the discount at this point. The POS system updates the order to reflect the discounted price.

10. **Apply Discount to Order:** The POS system applies the discount to the appropriate items in the order, ensuring that the final total reflects the discount.

11. **Return Final Order:** Once the discount is applied, the system finalizes the order, returning the updated total to the employee.

12. **Return Final Confirmation:** The employee provides the final reservation and order confirmation to the customer, including details such as the service booked, the discounted total (if applicable), and the reservation time.

# 4 System Architecture

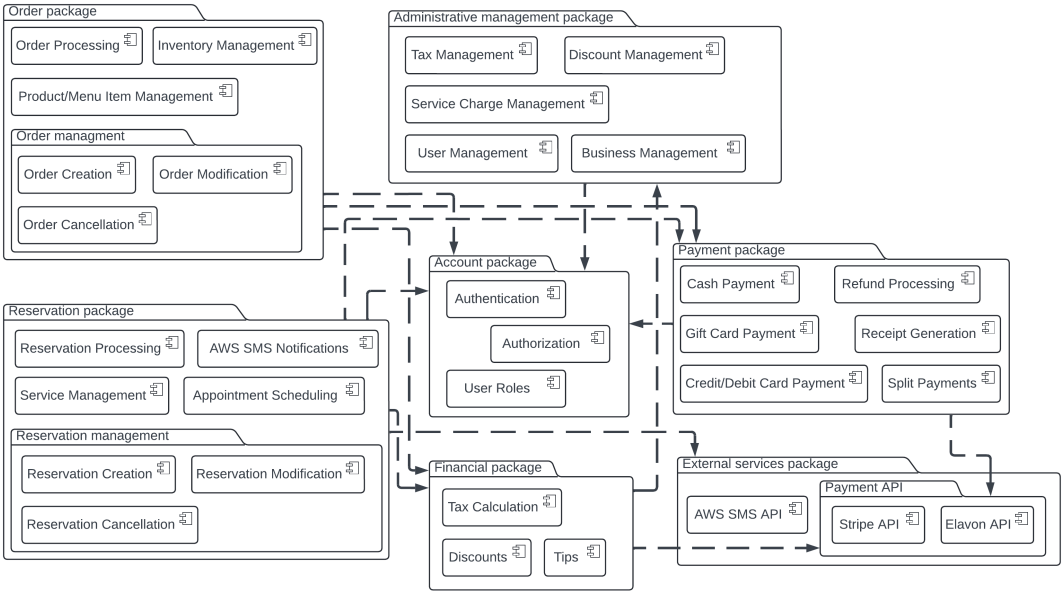


Figure 6: Back end package diagram

This package diagram breaks down the back end system into several main areas, grouping related functions, along with their dependency relationships.

The order package takes care of all the features needed for managing orders. It includes components for creating, updating, and canceling orders, as well as managing products, menu items, and inventory. This package works with the financial package to handle tips, discount and tax calculations, and with the payment package to process payments like cash, gift cards, and credit/debit cards. It also connects to the account package to ensure that only logged-in and authorized users can do these tasks.

The reservation package lets employees create, modify, and cancel customer appointments. It integrates with external services for SMS notifications (via AWS) and, similar to the order package, it links to the payment, financial and account packages to manage payments and user access related to reservations.

The account package handles authentication, authorization and user roles, ensuring that only appropriate people have access to the various system management tools.

The financial package deals with business calculations like taxes, discounts and tips for various services and products. Order and reservation packages rely on it for such calculations. It also links to the administrative management package, which modifies what categories of items get taxed and discounted, and by how much.

The payment package handles payment processing, supporting various methods and features like split payments, refunds and receipt generation. It uses external payment gateways like Stripe and Elavon through the payment API.

The administrative management package provides tools for managing broader business settings, including user roles, taxes, service charges, etc. By connecting to the account package, it ensures that only authorized users, like business owners or super admins, can make critical changes.

# 5 API Contracts

## 5.1 Overview

The API Contracts section outlines the backend endpoints necessary for the system's functionality. It provides detailed information on each endpoint, including HTTP methods, request parameters, request bodies, responses, and possible error codes. Additionally, Swagger/OpenAPI documentation is provided to facilitate seamless integration and implementation.

## 5.2 Backend Endpoints

### 5.2.1 Endpoint #1: `/api/businesses`

**Description** Handles business-related operations, including creating, modifying, retrieving, and deleting businesses. Each business has an owner, contact details, and other metadata.

#### HTTP Methods and Paths

- `POST /api/businesses` - Create a new business.
- `GET /api/businesses` - Retrieve a list of businesses.
- `GET /api/businesses/{businessId}` - Retrieve a specific business by ID.
- `PUT /api/businesses/{businessId}` - Modify an existing business.
- `DELETE /api/businesses/{businessId}` - Delete a business.

#### Request Parameters POST /api/businesses

- `Authorization` (Header, required): Bearer token for authentication.

#### GET /api/businesses

- `Authorization` (Header, required): Bearer token for authentication.
- `owner` (Query, optional): Filter businesses by owner ID.
- `name` (Query, optional): Filter businesses by business name.

#### GET /api/businesses/{businessId}

- `Authorization` (Header, required): Bearer token for authentication.
- `businessId` (Path, required): ID of the business to retrieve.

**PUT /api/businesses/{businessId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `businessId` (Path, required): ID of the business to modify.

**DELETE /api/businesses/{businessId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `businessId` (Path, required): ID of the business to delete.

**Request Bodies POST /api/businesses**

```
{
    "name": string,
    "owner": string,
    "email": string,
    "phoneNumber": string
}
```

**PUT /api/businesses/{businessId}**

```
{
    "name": string,
    "email": string,
    "phoneNumber": string
}
```

**Responses POST /api/businesses**

- `201 Created`: Business successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/businesses**

- `200 OK`: Returns a list of businesses.
- `401 Unauthorized`: Authentication failed.

- `500 Internal Server Error`: Server error.

**GET /api/businesses/{businessId}**

- `200 OK`: Returns the specified business.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Business not found.
- `500 Internal Server Error`: Server error.

**PUT /api/businesses/{businessId}**

- `200 OK`: Business successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Business not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/businesses/{businessId}**

- `200 OK`: Business successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Business not found.
- `500 Internal Server Error`: Server error.

### 5.2.2 Endpoint #2: `/api/users`

**Description** Handles user-related operations, including creating, modifying, retrieving, and deleting users. Users can have different roles, such as Employee, Business Owner, or SuperAdmin.

**HTTP Methods and Paths**

- `POST /api/users` - Create a new user.
- `GET /api/users` - Retrieve a list of users.
- `GET /api/users/{userId}` - Retrieve a specific user by ID.
- `PUT /api/users/{userId}` - Modify an existing user.
- `DELETE /api/users/{userId}` - Delete a user.

**Request Parameters POST /api/users**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/users**

- `Authorization` (Header, required): Bearer token for authentication.
- `role` (Query, optional): Filter users by their role (e.g., Employee, BusinessOwner, SuperAdmin).
- `businessId` (Query, optional): Filter users by business ID.

**GET /api/users/{userId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `userId` (Path, required): ID of the user to retrieve.

**PUT /api/users/{userId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `userId` (Path, required): ID of the user to modify.

**DELETE /api/users/{userId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `userId` (Path, required): ID of the user to delete.

**Request Bodies POST /api/users**

```
{
    "name": string,
    "role": "Employee" | "BusinessOwner" | "SuperAdmin",
    "password": string,
    "businessId": "int" (optional)
}
```

**PUT /api/users/{userId}**

```
{
    "name": string,
    "role": "Employee" | "BusinessOwner" | "SuperAdmin",
    "password": string (optional),
    "businessId": "int" (optional)
}
```

**Responses POST /api/users**

- `201 Created`: User successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/users**

- `200 OK`: Returns a list of users.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/users/{userId}**

- `200 OK`: Returns the specified user.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: User not found.
- `500 Internal Server Error`: Server error.

**PUT /api/users/{userId}**

- `200 OK`: User successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: User not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/users/{userId}**

- `200 OK`: User successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: User not found.
- `500 Internal Server Error`: Server error.

### 5.2.3 Endpoint #3: `/api/orders`

**Description** Handles all operations related to order management, including creating, modifying, retrieving, and deleting orders. Each order can have multiple items, payment details, and can be associated with a customer.

**HTTP Methods and Paths**

- `POST /api/orders` - Create a new order.
- `GET /api/orders` - Retrieve a list of orders.
- `GET /api/orders/{orderId}` - Retrieve a specific order by ID.
- `PUT /api/orders/{orderId}` - Modify an existing order.
- `DELETE /api/orders/{orderId}` - Cancel an order.

**Request Parameters POST /api/orders**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/orders**

- `Authorization` (Header, required): Bearer token for authentication.
- `status` (Query, optional): Filter orders by status (e.g., open, closed).
- `customerId` (Query, optional): Filter orders by customer ID.

**GET /api/orders/{orderId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `orderId` (Path, required): ID of the order to retrieve.

**PUT /api/orders/{orderId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `orderId` (Path, required): ID of the order to modify.

**DELETE /api/orders/{orderId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `orderId` (Path, required): ID of the order to cancel.

**Request Bodies POST /api/orders**

```
{
    "customerId": string,
    "items": [
        {
            "productId": string,
            "quantity": integer,
            "price": float
        }
    ],
    "paymentMethod": "cash" | "gift_card" | "credit_card",
    "tip": number,
    "discountId": string (optional)
}
```

**PUT /api/orders/{orderId}**

```
{
    "items": [
        {
            "productId": string,
            "quantity": integer,
            "price": float
        }
    ],
    "paymentMethod": "cash" | "gift_card" | "credit_card",
    "tip": number,
    "discountId": string (optional)
}
```

**Responses POST /api/orders**

- `201 Created`: Order successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/orders**

- `200 OK`: Returns a list of orders.

- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/orders/{orderId}**

- `200 OK`: Returns the specified order.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Order not found.
- `500 Internal Server Error`: Server error.

**PUT /api/orders/{orderId}**

- `200 OK`: Order successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `403 Forbidden`: Modification not allowed (e.g., order is closed).
- `404 Not Found`: Order not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/orders/{orderId}**

- `200 OK`: Order successfully canceled.
- `401 Unauthorized`: Authentication failed.
- `403 Forbidden`: Cancellation not allowed (e.g., order is closed).
- `404 Not Found`: Order not found.
- `500 Internal Server Error`: Server error.

#### 5.2.4 Endpoint #4: `/api/products`

**Description** Handles product-related operations, including creating, modifying, retrieving, and deleting products. Products can include items for sale, services, and can have variations (e.g., sizes or types).

**HTTP Methods and Paths**

- `POST /api/products` - Create a new product.
- `GET /api/products` - Retrieve a list of products.
- `GET /api/products/{productId}` - Retrieve a specific product by ID.
- `PUT /api/products/{productId}` - Modify an existing product.
- `DELETE /api/products/{productId}` - Delete a product.

**Request Parameters POST /api/products**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/products**

- `Authorization` (Header, required): Bearer token for authentication.
- `category` (Query, optional): Filter products by category (e.g., food, services).
- `name` (Query, optional): Filter products by name.

**GET /api/products/{productId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `productId` (Path, required): ID of the product to retrieve.

**PUT /api/products/{productId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `productId` (Path, required): ID of the product to modify.

**DELETE /api/products/{productId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `productId` (Path, required): ID of the product to delete.

**Request Bodies POST /api/products**

```
{
    "name": string,
    "category": string,
    "price": float,
    "quantityInStock": integer,
    "description": string,
```

```
    "variations": [
        {
            "name": string,
            "price": float (optional)
        }
    ]
}
```

**PUT /api/products/{productId}**

```
{
    "name": string,
    "category": string,
    "price": float,
    "quantityInStock": integer,
    "description": string,
    "variations": [
        {
            "name": string,
            "price": float (optional)
        }
    ]
}
```

**Responses POST /api/products**

- `201 Created`: Product successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/products**

- `200 OK`: Returns a list of products.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/products/{productId}**

- `200 OK`: Returns the specified product.
- `401 Unauthorized`: Authentication failed.

- `404 Not Found`: Product not found.
- `500 Internal Server Error`: Server error.

**PUT /api/products/{productId}**

- `200 OK`: Product successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Product not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/products/{productId}**

- `200 OK`: Product successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Product not found.
- `500 Internal Server Error`: Server error.

#### 5.2.5 Endpoint #5: `/api/orderitems`

**Description** Handles all operations related to managing items within an order. This includes adding, modifying, and removing items from an order. Each order item represents a specific product within the order.

**HTTP Methods and Paths**

- `POST /api/orderitems` - Add an item to an order.
- `PUT /api/orderitems/{itemId}` - Modify an existing order item.
- `DELETE /api/orderitems/{itemId}` - Remove an item from an order.

**Request Parameters** **POST /api/orderitems**

- `Authorization` (Header, required): Bearer token for authentication.

**PUT /api/orderitems/{itemId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `itemId` (Path, required): ID of the order item to modify.

**DELETE /api/orderitems/{itemId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `itemId` (Path, required): ID of the order item to delete.

**Request Bodies POST /api/orderitems**

```
{
    "orderId": string,
    "productId": string,
    "quantity": integer,
    "price": number
}
```

**PUT /api/orderitems/{itemId}**

```
{
    "quantity": integer,
    "price": number
}
```

**Responses POST /api/orderitems**

- `201 Created`: Order item successfully added to the order.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Order or product not found.
- `500 Internal Server Error`: Server error.

**PUT /api/orderitems/{itemId}**

- `200 OK`: Order item successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Order item not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/orderitems/{itemId}**

- `200 OK`: Order item successfully removed.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Order item not found.
- `500 Internal Server Error`: Server error.

### 5.2.6 Endpoint #6: `/api/taxes`

**Description** Handles all operations related to tax management, including creating, modifying, retrieving, and deleting taxes. Taxes can be applied to products and services, and they can vary by percentage and category (e.g., VAT, service tax).

**HTTP Methods and Paths**

- `POST /api/taxes` - Create a new tax.
- `GET /api/taxes` - Retrieve a list of taxes.
- `GET /api/taxes/{taxId}` - Retrieve a specific tax by ID.
- `PUT /api/taxes/{taxId}` - Modify an existing tax.
- `DELETE /api/taxes/{taxId}` - Delete a tax.

**Request Parameters POST /api/taxes**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/taxes**

- `Authorization` (Header, required): Bearer token for authentication.
- `category` (Query, optional): Filter taxes by category (e.g., VAT, service).

**GET /api/taxes/{taxId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `taxId` (Path, required): ID of the tax to retrieve.

**PUT /api/taxes/{taxId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `taxId` (Path, required): ID of the tax to modify.

**DELETE /api/taxes/{taxId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `taxId` (Path, required): ID of the tax to delete.

**Request Bodies POST /api/taxes**

```
{
    "percentage": float,
    "category": string,
    "startDate": datetime,
    "endDate": datetime (optional)
}
```

**PUT /api/taxes/{taxId}**

```
{
    "percentage": float,
    "category": string,
    "startDate": datetime,
    "endDate": datetime (optional)
}
```

**Responses POST /api/taxes**

- `201 Created`: Tax successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/taxes**

- `200 OK`: Returns a list of taxes.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/taxes/{taxId}**

- `200 OK`: Returns the specified tax.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Tax not found.
- `500 Internal Server Error`: Server error.

**PUT /api/taxes/{taxId}**

- `200 OK`: Tax successfully updated.

- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Tax not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/taxes/{taxId}**

- `200 OK`: Tax successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Tax not found.
- `500 Internal Server Error`: Server error.

#### 5.2.7 Endpoint #7: `/api/discounts`

**Description** Handles all operations related to discount management, including creating, modifying, retrieving, and deleting discounts. Discounts can be applied to specific products or entire orders and can have different valid periods.

**HTTP Methods and Paths**

- `POST /api/discounts` - Create a new discount.
- `GET /api/discounts` - Retrieve a list of discounts.
- `GET /api/discounts/{discountId}` - Retrieve a specific discount by ID.
- `PUT /api/discounts/{discountId}` - Modify an existing discount.
- `DELETE /api/discounts/{discountId}` - Delete a discount.

**Request Parameters** **POST /api/discounts**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/discounts**

- `Authorization` (Header, required): Bearer token for authentication.
- `isForOrder` (Query, optional): Filter discounts by their application (specific product or entire order).
- `name` (Query, optional): Filter discounts by name.

**GET /api/discounts/{discountId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `discountId` (Path, required): ID of the discount to retrieve.

**PUT /api/discounts/{discountId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `discountId` (Path, required): ID of the discount to modify.

**DELETE /api/discounts/{discountId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `discountId` (Path, required): ID of the discount to delete.

**Request Bodies POST /api/discounts**

```
{
    "name": string,
    "value": float,
    "validFrom": datetime,
    "validUntil": datetime,
    "isForOrder": "boolean",
    "description": string (optional)
}
```

**PUT /api/discounts/{discountId}**

```
{
    "name": string,
    "value": float,
    "validFrom": datetime,
    "validUntil": datetime,
    "isForOrder": "boolean",
    "description": string (optional)
}
```

**Responses POST /api/discounts**

- `201 Created`: Discount successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/discounts**

- `200 OK`: Returns a list of discounts.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/discounts/{discountId}**

- `200 OK`: Returns the specified discount.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Discount not found.
- `500 Internal Server Error`: Server error.

**PUT /api/discounts/{discountId}**

- `200 OK`: Discount successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Discount not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/discounts/{discountId}**

- `200 OK`: Discount successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Discount not found.
- `500 Internal Server Error`: Server error.

### 5.2.8 Endpoint #8: `/api/reservations`

**Description** Handles all operations related to reservation management, including creating, modifying, retrieving, and canceling reservations. Each reservation can be linked to a customer, a service, an employee, and an appointment time.

**HTTP Methods and Paths**

- `POST /api/reservations` - Create a new reservation.
- `GET /api/reservations` - Retrieve a list of reservations.
- `GET /api/reservations/{reservationId}` - Retrieve a specific reservation by ID.
- `PUT /api/reservations/{reservationId}` - Modify an existing reservation.
- `DELETE /api/reservations/{reservationId}` - Cancel a reservation.

**Request Parameters POST /api/reservations**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/reservations**

- `Authorization` (Header, required): Bearer token for authentication.
- `date` (Query, optional): Filter reservations by appointment date.
- `employeeId` (Query, optional): Filter reservations by employee ID.

**GET /api/reservations/{reservationId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `reservationId` (Path, required): ID of the reservation to retrieve.

**PUT /api/reservations/{reservationId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `reservationId` (Path, required): ID of the reservation to modify.

**DELETE /api/reservations/{reservationId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `reservationId` (Path, required): ID of the reservation to cancel.

**Request Bodies POST /api/reservations**

```
{
    "customerId": string,
    "appointmentTime": "YYYY-MM-DDTHH:MM:SSZ",
    "serviceId": string,
    "employeeId": string (optional),
    "contactInfo": {
        "phone": string,
        "email": string
    }
}
```

**PUT /api/reservations/{reservationId}**

```
{
    "appointmentTime": "YYYY-MM-DDTHH:MM:SSZ",
    "serviceId": string,
    "employeeId": string (optional),
    "contactInfo": {
        "phone": string,
        "email": string
    }
}
```

**Responses POST /api/reservations**

- `201 Created`: Reservation successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/reservations**

- `200 OK`: Returns a list of reservations.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/reservations/{reservationId}**

- `200 OK`: Returns the specified reservation.

- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Reservation not found.
- `500 Internal Server Error`: Server error.

**PUT /api/reservations/{reservationId}**

- `200 OK`: Reservation successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Reservation not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/reservations/{reservationId}**

- `200 OK`: Reservation successfully canceled.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Reservation not found.
- `500 Internal Server Error`: Server error.

#### 5.2.9 Endpoint #9: `/api/services`

**Description** Handles all operations related to managing services, including creating, modifying, retrieving, and deleting services. Services are activities or tasks that businesses offer to customers (e.g., haircut, massage), and each service can have an associated duration and price.

**HTTP Methods and Paths**

- `POST /api/services` - Create a new service.
- `GET /api/services` - Retrieve a list of services.
- `GET /api/services/{serviceId}` - Retrieve a specific service by ID.
- `PUT /api/services/{serviceId}` - Modify an existing service.
- `DELETE /api/services/{serviceId}` - Delete a service.

**Request Parameters POST /api/services**

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/services**

- `Authorization` (Header, required): Bearer token for authentication.
- `name` (Query, optional): Filter services by name.
- `category` (Query, optional): Filter services by category.

**GET /api/services/{serviceId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `serviceId` (Path, required): ID of the service to retrieve.

**PUT /api/services/{serviceId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `serviceId` (Path, required): ID of the service to modify.

**DELETE /api/services/{serviceId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `serviceId` (Path, required): ID of the service to delete.

**Request Bodies POST /api/services**

```
{
    "name": string,
    "category": string,
    "durationMinutes": integer,
    "price": float,
    "description": string (optional)
}
```

**PUT /api/services/{serviceId}**

```
{
    "name": string,
    "category": string,
    "durationMinutes": integer,
    "price": float,
    "description": string (optional)
}
```

**Responses POST /api/services**

- `201 Created`: Service successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/services**

- `200 OK`: Returns a list of services.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/services/{serviceId}**

- `200 OK`: Returns the specified service.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Service not found.
- `500 Internal Server Error`: Server error.

**PUT /api/services/{serviceId}**

- `200 OK`: Service successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Service not found.
- `500 Internal Server Error`: Server error.

**DELETE /api/services/{serviceId}**

- `200 OK`: Service successfully deleted.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Service not found.
- `500 Internal Server Error`: Server error.

### 5.2.10 Endpoint #10: `/api/payments`

**Description** Handles all operations related to payment management, including creating, retrieving, and modifying payments. Payments are linked to orders and can be made using various methods such as cash, credit card, or gift card.

#### HTTP Methods and Paths

- `POST /api/payments` - Create a new payment.
- `GET /api/payments` - Retrieve a list of payments.
- `GET /api/payments/{paymentId}` - Retrieve a specific payment by ID.
- `PUT /api/payments/{paymentId}` - Modify an existing payment.

#### Request Parameters POST /api/payments

- `Authorization` (Header, required): Bearer token for authentication.

**GET /api/payments**

- `Authorization` (Header, required): Bearer token for authentication.
- `orderId` (Query, optional): Filter payments by the associated order ID.
- `paymentMethod` (Query, optional): Filter payments by the payment method (e.g., cash, credit card).

**GET /api/payments/{paymentId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `paymentId` (Path, required): ID of the payment to retrieve.

**PUT /api/payments/{paymentId}**

- `Authorization` (Header, required): Bearer token for authentication.
- `paymentId` (Path, required): ID of the payment to modify.

#### Request Bodies POST /api/payments

```
{
    "orderId": string,
    "amount": float,
    "tip": float,
    "paymentMethod": "cash" | "credit_card" | "gift_card"
}
```

**PUT /api/payments/{paymentId}**

```
{
    "amount": float,
    "tip": float,
    "paymentMethod": "cash" | "credit_card" | "gift_card"
}
```

**Responses POST /api/payments**

- `201 Created`: Payment successfully created.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Order not found.
- `500 Internal Server Error`: Server error.

**GET /api/payments**

- `200 OK`: Returns a list of payments.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

**GET /api/payments/{paymentId}**

- `200 OK`: Returns the specified payment.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Payment not found.
- `500 Internal Server Error`: Server error.

**PUT /api/payments/{paymentId}**

- `200 OK`: Payment successfully updated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `404 Not Found`: Payment not found.
- `500 Internal Server Error`: Server error.

### 5.2.11 Endpoint #11: `/api/auth`

**Description** Handles authentication operations tied to user management, including generating and validating bearer tokens for users. Tokens are based on user credentials.

#### HTTP Methods and Paths

- `POST /api/auth/generate-token` - Generate a bearer token for a specific user.
- `POST /api/auth/validate-token` - Validate an existing bearer token.

#### Request Parameters POST /api/auth/generate-token

- `Request Body` (Header, required): User credentials for generating a token.

#### POST /api/auth/validate-token

- `Authorization` (Header, required): Bearer token for authentication validation.

#### Request Bodies POST /api/auth/generate-token

```
{
    "name": "string",
    "password": "string"
}
```

#### Responses POST /api/auth/generate-token

- `200 OK`: Token successfully generated.
- `400 Bad Request`: Invalid input data.
- `401 Unauthorized`: Authentication failed.
- `500 Internal Server Error`: Server error.

#### POST /api/auth/validate-token

- `200 OK`: Token is valid.
- `401 Unauthorized`: Invalid token.
- `500 Internal Server Error`: Server error.