VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
SOFTWARE ENGINEERING

Point of Sale SaaS

Technical design

Author: Bražėnas Arminas, Cicėnaitė Vesta
Kisieliūtė Rūta, Kaziulis Deividas, Stoškus Karolis

Vilnius – 2024

Contents

1 INTRODUCTION 3

2 BUSINESS FLOWS 3
- 2.1 Business Creation Flow 3
- 2.2 Business Modification Flow 4
- 2.3 User Creation Flow 5
- 2.4 User Modification Flow 7
- 2.5 Tax creation flow 9
- 2.6 Service charge creation flow 10
- 2.7 Order creation flow 11
- 2.8 Order price calculation flow 12
- 2.9 Pay for order by cash flow 13
- 2.10 Pay for order by card flow 14
- 2.11 Reservation creation flow 15
- 2.12 Reservation creation flow (2) 16
- 2.13 Reservation modification flow 18

3 ENTITY DATA MODEL 19
- 3.1 User entity data model 19
- 3.2 Order entity data model 21
- 3.3 Payments entity data model 22

4 AUTHENTICATION AND AUTHORIZATION 23
- 4.1 Authentication and Authorization Flow 23
- 4.2 User Roles 24

5 MODULE DIAGRAM 25

6 API ENDPOINT CONTRACTS 27

1 Introduction

This document provides an overview of the Point of Sale SaaS designed for businesses in catering and beauty sectors. The system aims to offer a flexible solution for managing orders, payments, and user interactions. The document includes details on business flows through sequence and flow diagrams, high-level system architecture with module diagram, and the data model represented by class diagrams. This information serves as a guide for developers and stakeholders involved in the software's design and implementation.

2 Business flows

2.1 Business Creation Flow

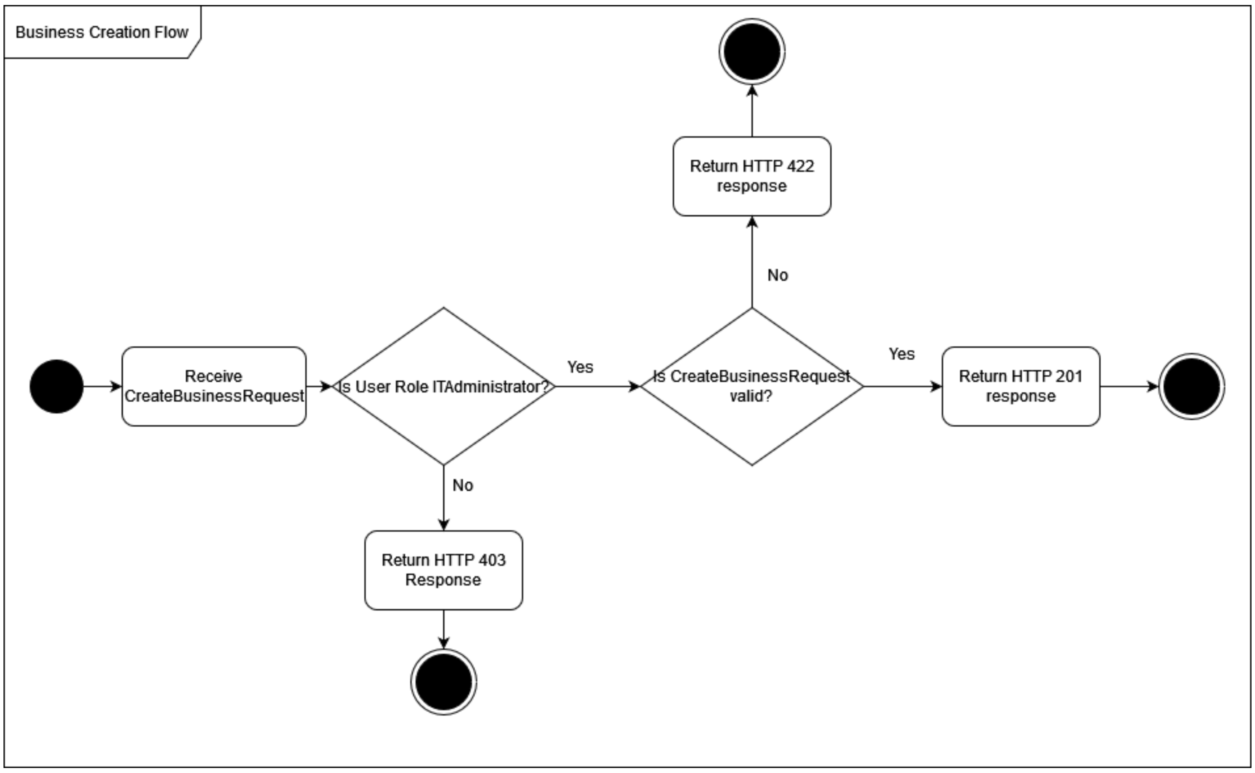

Figure 1. Business creation flow.

Figure 1 illustrates the process of creating business. The steps are as follows:

1. Receive CreateBusinessRequest:
   - The flow begins when a CreateBusinessRequest is received by the system. This request contains the necessary data to create a new business.
2. Is User Role ITAdministrator?
   - The system checks the role of the user making the request.
   - Condition:
     – If the user's role is ITAdministrator, the flow continues to the next step.

    - If the user's role is not ITAdministrator, the system returns an HTTP 403 (Forbidden) response and ends the flow.
3. Is CreateBusinessRequest valid?
    - The system then validates the CreateBusinessRequest to ensure all the required data and conditions are met.
    - Condition:
        - If the request is valid, the system moves on to the next step.
        - If the request is invalid, the system returns an HTTP 422 (Unprocessable Entity) response, indicating that there are validation errors in the request.
4. Return HTTP 201 Response:
    - If the request passes validation, the system successfully creates the business and returns an HTTP 201 (Created) response, indicating that the new business has been created.

2.2 Business Modification Flow

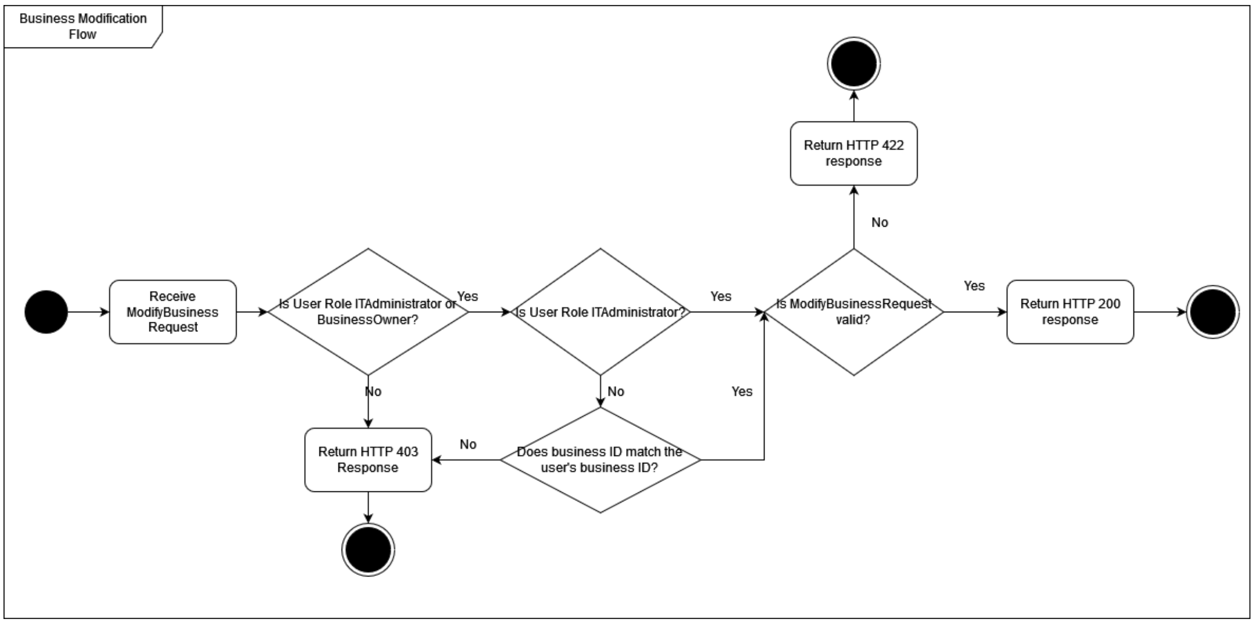
Business Modification Flow
Receive ModifyBusiness Request
Is User Role ITAdministrator or BusinessOwner?
Yes
No
Is User Role ITAdministrator?
Yes
No
Is ModifyBusinessRequest valid?
Yes
No
Return HTTP 422 response
Return HTTP 200 response
Does business ID match the user's business ID?
Yes
No
Return HTTP 403 Response

Figure 2. Business modification flow.

Figure 2 illustrates the process of modifying business. The steps are as follows:

1. Receive ModifyBusiness Request:
    - The flow begins when the system receives a request to modify business details via the ModifyBusinessRequest.
2. Is User Role ITAdministrator or BusinessOwner?
    - The system checks the role of the user making the modification request.
    - Condition:
        - If the user's role is either ITAdministrator or BusinessOwner, the flow proceeds to the next step.

    - If the user's role is neither, the system returns an HTTP 403 (Forbidden) response and the flow ends.
3. Is User Role ITAdministrator?
    - The system checks if the user is an ITAdministrator.
    - Condition:
        - If the user is an ITAdministrator, the flow continues to check the request's validity.
        - If the user is not an ITAdministrator, the system checks if the business ID provided in the request matches the user's associated business ID.
4. Does Business ID Match the User's Business ID?
    - If the user is not an ITAdministrator, the system checks whether the business ID in the request matches the business ID associated with the user.
    - Condition:
        - If the business ID matches, the flow proceeds to validate the request.
        - If the business ID does not match, the system returns an HTTP 403 (Forbidden) response and the flow ends.
5. Is ModifyBusinessRequest valid?
    - The system then validates the ModifyBusinessRequest to ensure all the required data and conditions are met.
    - Condition:
        - If the request is valid, the system moves on to the next step.
        - If the request is invalid, the system returns an HTTP 422 (Unprocessable Entity) response, indicating that there are validation errors in the request.
6. Return HTTP 200 Response:
    - If the request passes validation, the system successfully modifies the business and returns an HTTP 200 (OK) response, indicating that the business has been modified.

2.3 User Creation Flow

Figure 3 illustrates the process of creating user. The steps are as follows:

1. Receive CreateUserRequest:
    - The flow begins when the system receives a request to create a new user via the CreateUserRequest.
2. Is User Role ITAdministrator or BusinessOwner?
    - The system checks the role of the user making the request.
    - Condition:
        - If the user's role is either ITAdministrator or BusinessOwner, the flow proceeds to the next step.
        - If the user's role is neither, the system returns an HTTP 403 (Forbidden) response, and the flow ends.
3. Is User Role ITAdministrator?
    - The system checks if the user making the request is an ITAdministrator.

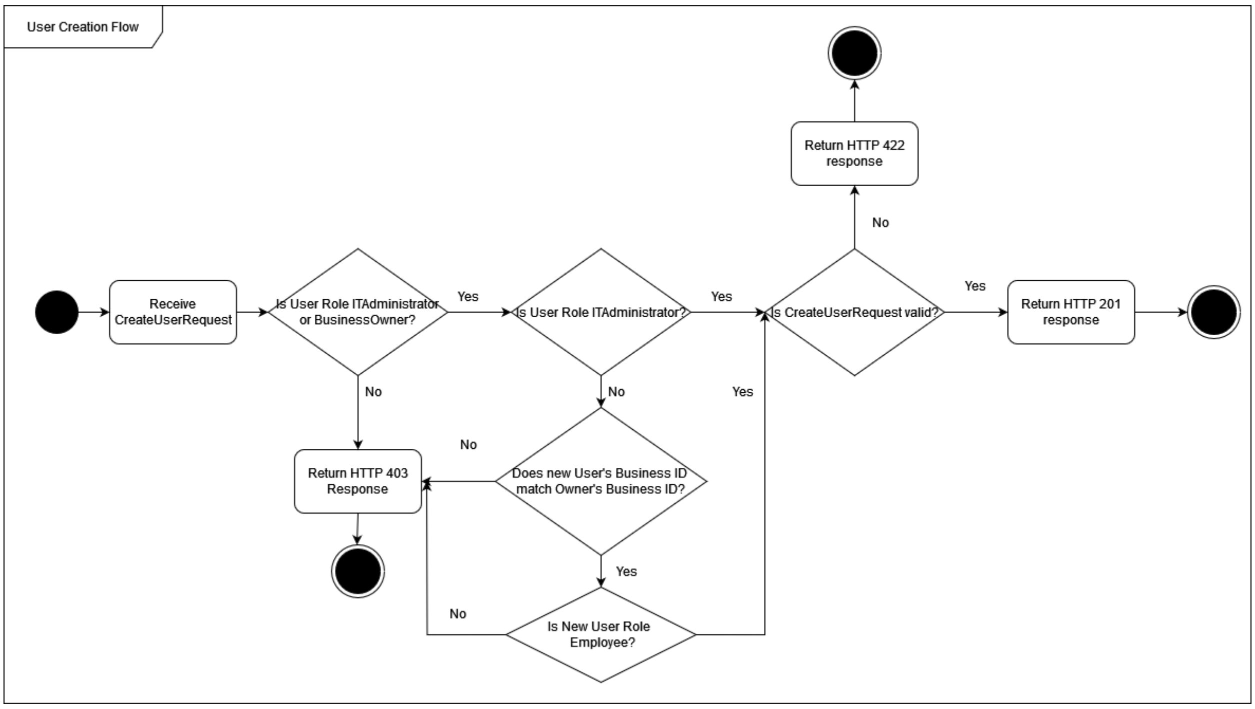
User Creation Flow
Receive CreateUserRequest
Is User Role ITAdministrator or BusinessOwner?
Yes
No
Is User Role ITAdministrator?
Yes
No
Is CreateUserRequest valid?
Yes
No
Return HTTP 422 response
Return HTTP 201 response
Return HTTP 403 Response
Does new User's Business ID match Owner's Business ID?
No
Yes
Is New User Role Employee?
No
Yes

Figure 3. User creation flow.

- Condition:
  - If the user is an ITAdministrator, the flow skips further business ID checks and proceeds to validate the request.
  - If the user is not an ITAdministrator, the system checks whether the business ID provided in the request matches the business ID associated with the business owner.

4. Does new User's Business ID match Owner's Business ID?
   - If the user is not an ITAdministrator, the system checks whether the new user's business ID matches the business owner's business ID.
   - Condition:
     - If the business IDs match, the flow continues to check if the role of the new user to be created is an Employee.
     - If the business IDs do not match, the system returns an HTTP 403 (Forbidden) response, and the flow ends.
5. Is New User Role Employee?
   - The system checks if the new user role to be created is an Employee.
   - Condition:
     - If the new user's role is Employee, the system proceeds to validate the request.
     - If the new user's role is not Employee, the system returns an HTTP 403 (Forbidden) response, and the flow ends.
6. Is CreateUserRequest valid?
   - The system then validates the CreateUserRequest to ensure all the required data and conditions are met.
   - Condition:

  - If the request is valid, the system moves on to the next step.
  - If the request is invalid, the system returns an HTTP 422 (Unprocessable Entity) response, indicating that there are validation errors in the request, and the flow ends.
7. Return HTTP 201 Response:
   - If the request passes validation, the system successfully creates the user and returns an HTTP 201 (Created) response, indicating that the new user has been created.

2.4 User Modification Flow

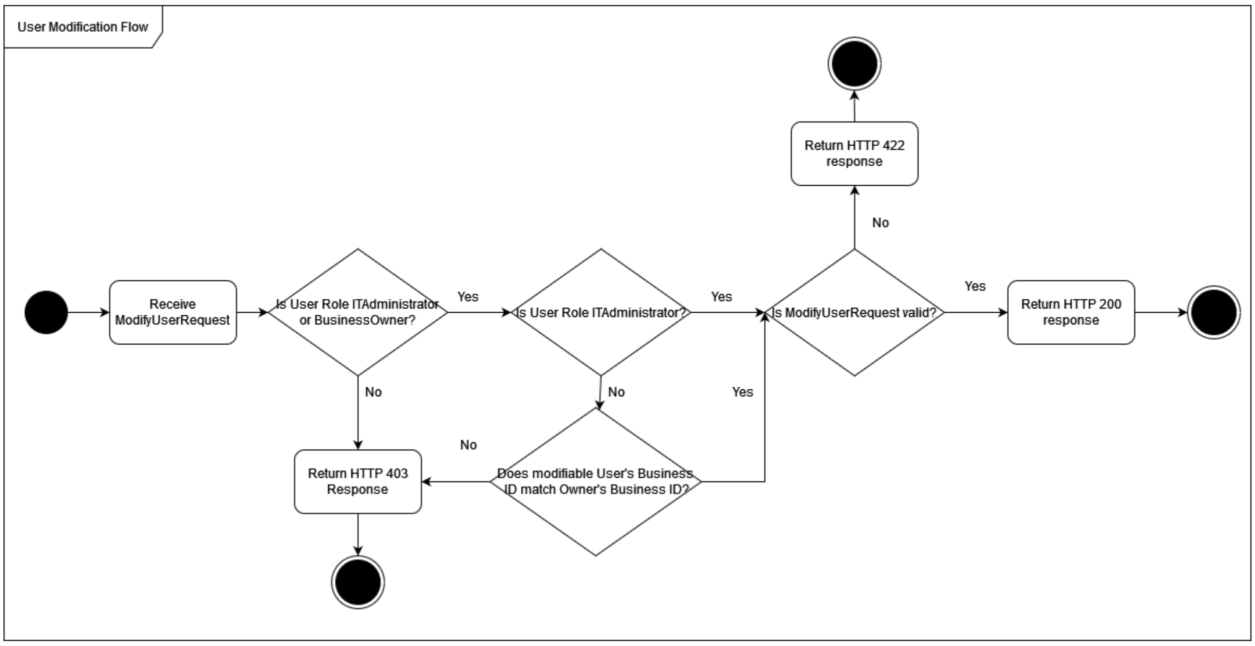
User Modification Flow
Receive ModifyUserRequest
Is User Role ITAdministrator or BusinessOwner?
Yes
No
Is User Role ITAdministrator?
Yes
No
Does modifiable User's Business ID match Owner's Business ID?
Yes
No
Return HTTP 403 Response
Is ModifyUserRequest valid?
Yes
No
Return HTTP 422 response
Return HTTP 200 response

Figure 4. User modification flow.

Figure 4 illustrates the process of modifying user. The steps are as follows:

1. Receive ModifyUserRequest:
   - The flow begins when the system receives a request to modify an existing user via the ModifyUserRequest.
2. Is User Role ITAdministrator or BusinessOwner?
   - The system checks the role of the user making the modification request.
   - Condition:
     - If the user's role is either ITAdministrator or BusinessOwner, the flow proceeds to the next step.
     - If the user's role is neither, the system returns an HTTP 403 (Forbidden) response, and the flow ends.
3. Is User Role ITAdministrator?
   - The system checks if the user making the request is an ITAdministrator.
   - Condition:
     - If the user is an ITAdministrator, the flow skips further business ID checks and proceeds to validate the request.

    - If the user is not an ITAdministrator, the system checks whether the modifiable user's business ID matches the business owner's business ID.

4. Does modifiable User's Business ID match Owner's Business ID?
    - If the user is not an ITAdministrator, the system checks whether the modifiable user's business ID matches the business ID associated with the business owner.
    - Condition:
        - If the business IDs match, the flow continues to validate the modification request.
        - If the business IDs do not match, the system returns an HTTP 403 (Forbidden) response, and the flow ends.
5. Is ModifyUserRequest valid?
    - The system then validates the ModifyUserRequest to ensure all the required data and conditions are met.
    - Condition:
        - If the request is valid, the system moves on to the next step.
        - If the request is invalid, the system returns an HTTP 422 (Unprocessable Entity) response, indicating that there are validation errors in the request, and the flow ends.
6. Return HTTP 200 Response:
    - If the request passes validation, the system successfully modifies the user and returns an HTTP 200 (OK) response, indicating that the user has been modified.

2.5 Tax creation flow

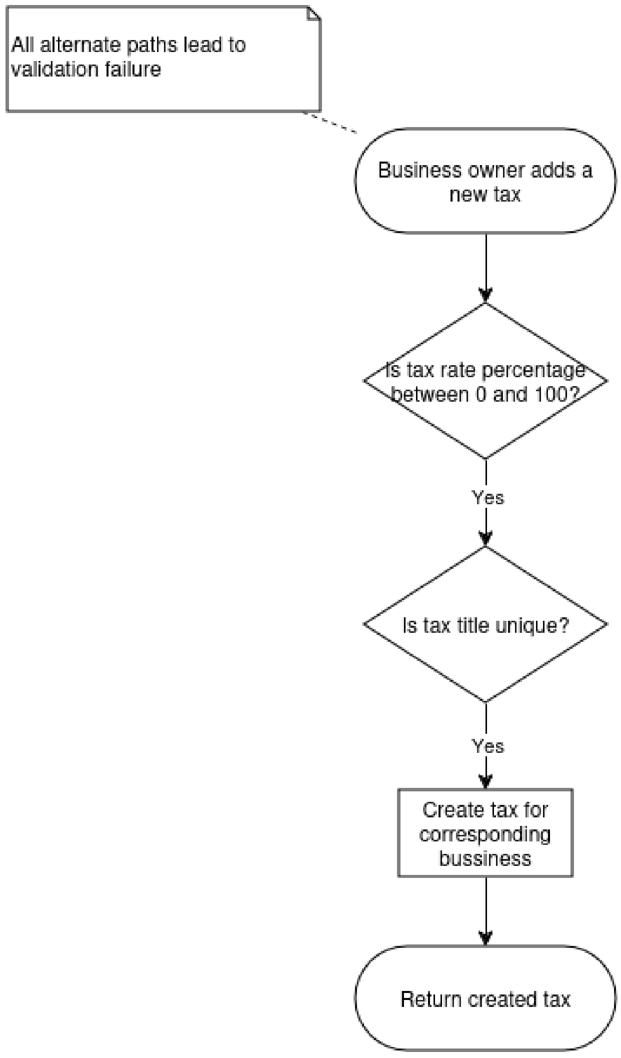
All alternate paths lead to validation failure
Business owner adds a new tax
Is tax rate percentage between 0 and 100?
Yes
Is tax title unique?
Yes
Create tax for corresponding bussiness
Return created tax

Figure 5. Tax creation flow.

Figure 5 illustrates the process of creating a tax. Business owners can define custom taxes, which can be applied to their business products. Tax must have a rate percentage ranging from 0 to 100, and the tax title must be unique within the business.

Taxes can be modified by the business owner, following the same validation rules. Any changes made to it will not affect historical order items where the tax has already been applied.

2.6 Service charge creation flow

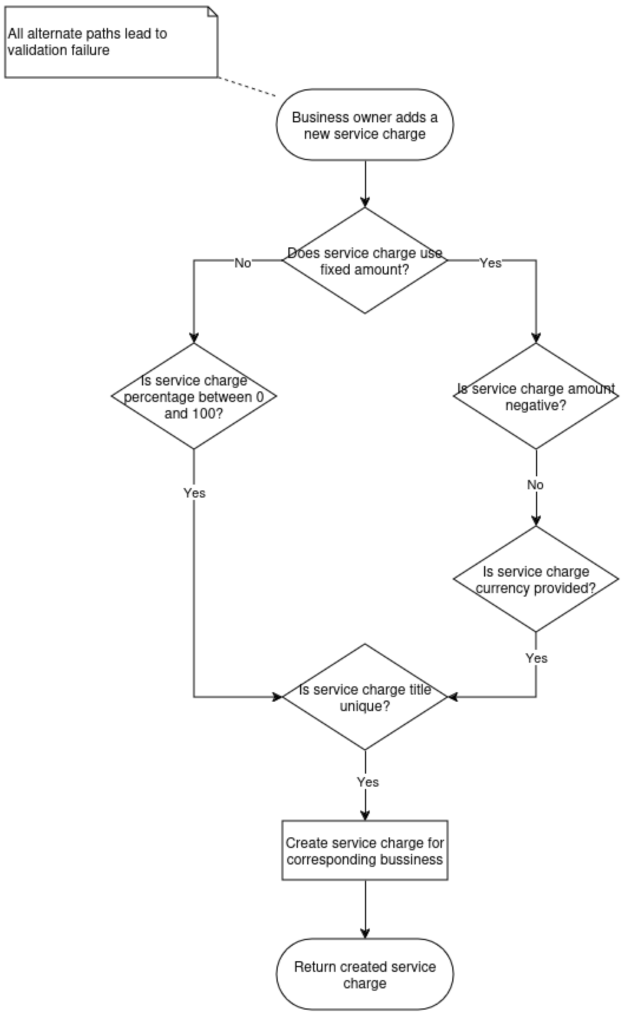
All alternate paths lead to validation failure
Business owner adds a new service charge
Does service charge use fixed amount?
No
Yes
Is service charge percentage between 0 and 100?
Is service charge amount negative?
Yes
No
Is service charge currency provided?
Yes
Is service charge title unique?
Yes
Create service charge for corresponding bussiness
Return created service charge

Figure 6. Service charge creation flow.

Figure 6 illustrates the process of creating a service charge. Business owners can define custom service charges, which can be applied by the employee when taking an order or reservation. A service charge can be of fixed amount or a percentage from the order's price.

Service charges can be modified by the business owner, following the same validation rules. Any changes made to it will not affect historical orders where the service charge has already been applied.

2.7 Order creation flow

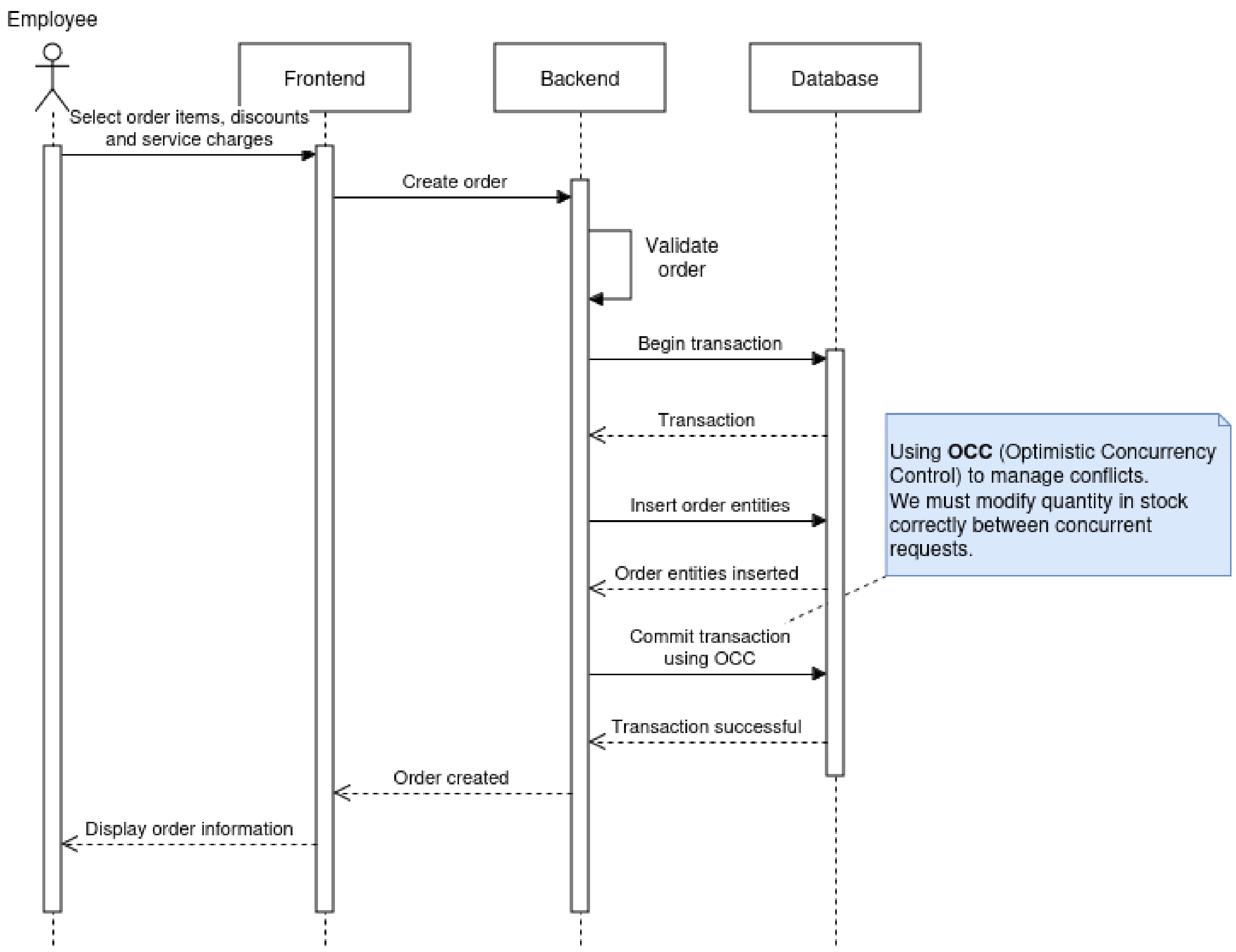
Employee
Frontend
Backend
Database
Select order items, discounts and service charges
Create order
Validate order
Begin transaction
Transaction
Insert order entities
Order entities inserted
Commit transaction using OCC
Transaction successful
Order created
Display order information
Using OCC (Optimistic Concurrency Control) to manage conflicts. We must modify quantity in stock correctly between concurrent requests.

Figure 7. Order creation flow.

Figure 7 illustrates the process of creating an order by an employee. The steps are as follows:

1. The employee:
   1.1. Selects products to order, along with any modifiers;
   1.2. Optionally applies discounts to the entire order or specific items;
   1.3. Optionally applies service charges;
   1.4. Optionally associates the order with a reservation.
2. The frontend sends order details to the backend via an HTTP request.
3. The backend validates the order:
   3.1. Ensures that the ordered items are in stock;
   3.2. Verifies that any applied discounts are valid and not expired.
4. The backend initiates a transaction with the database.
5. Backend inserts the order to the database and reduces the product stock accordingly.
6. Using Optimistic Concurrency Control, the backend commits the transaction. If a concurrent transaction modifies the product stock, the transaction is rolled back, stock levels are recalculated, and the commit is retried.

7. Once the transaction is successfully committed, the backend returns the order information, including the calculated prices, to the frontend.
8. The frontend displays order details to the employee.

2.8 Order price calculation flow

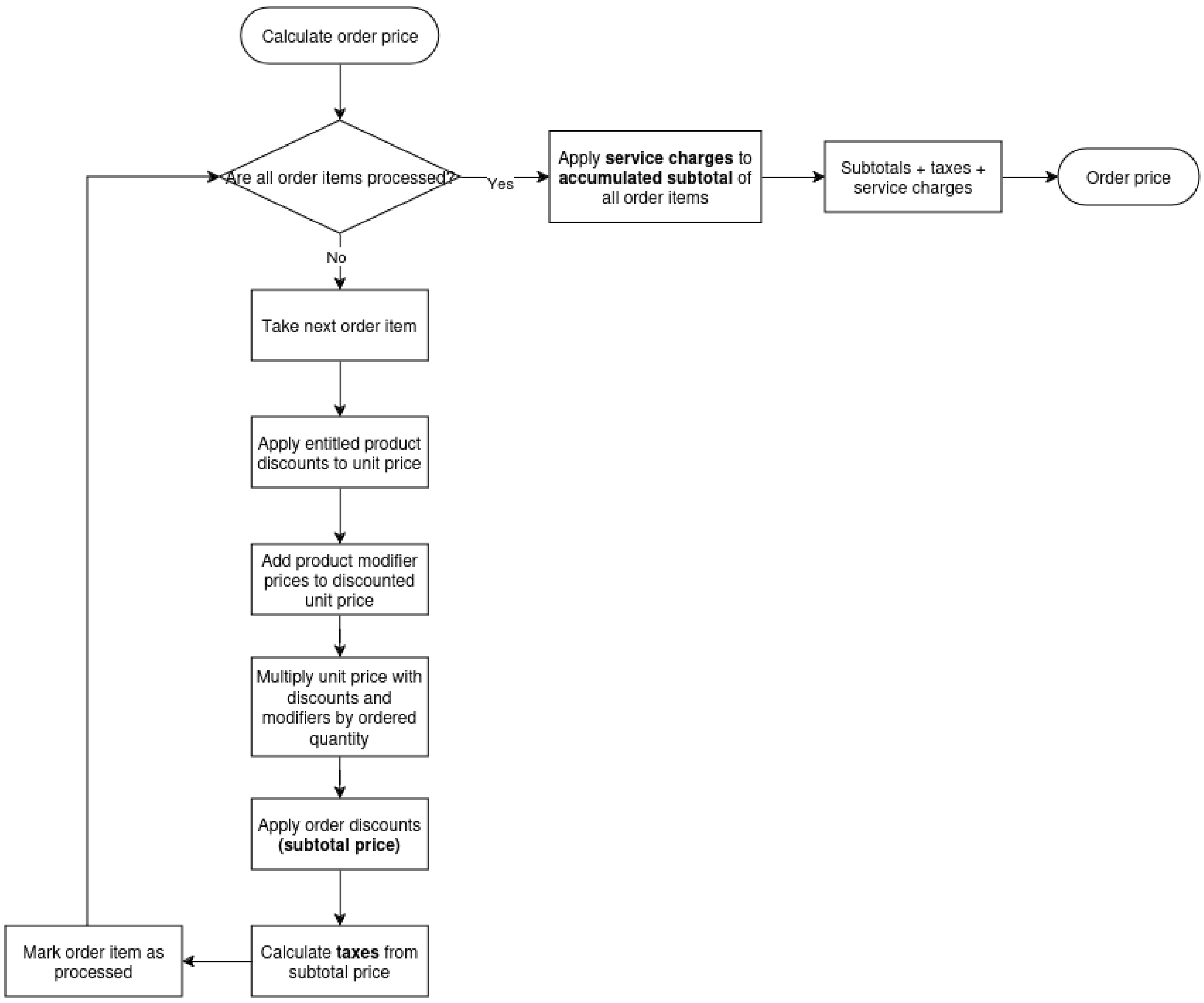
Calculate order price
Are all order items processed?
Yes
Apply service charges to accumulated subtotal of all order items
Subtotals + taxes + service charges
Order price
No
Take next order item
Apply entitled product discounts to unit price
Add product modifier prices to discounted unit price
Multiply unit price with discounts and modifiers by ordered quantity
Apply order discounts (subtotal price)
Calculate taxes from subtotal price
Mark order item as processed

Figure 8. Order price calculation flow.

Figure 8 illustrates the process of calculating order price. The steps are as follows:

1. For each order item:
   1.1. Start with the base product unit price and apply any discounts.
   1.2. Apply any product modifier prices to the discounted unit price.
   1.3. Multiply the resulting price by the ordered quantity to obtain the accumulated price for the item.
   1.4. Apply any order-level discounts to the accumulated price to get the item's subtotal.
   1.5. Calculate taxes based on the item's subtotal price.
2. Apply service charges to the accumulated subtotal of all order items.
3. Sum the item subtotals, taxes, and service charges to get the final order price.

2.9 Pay for order by cash flow

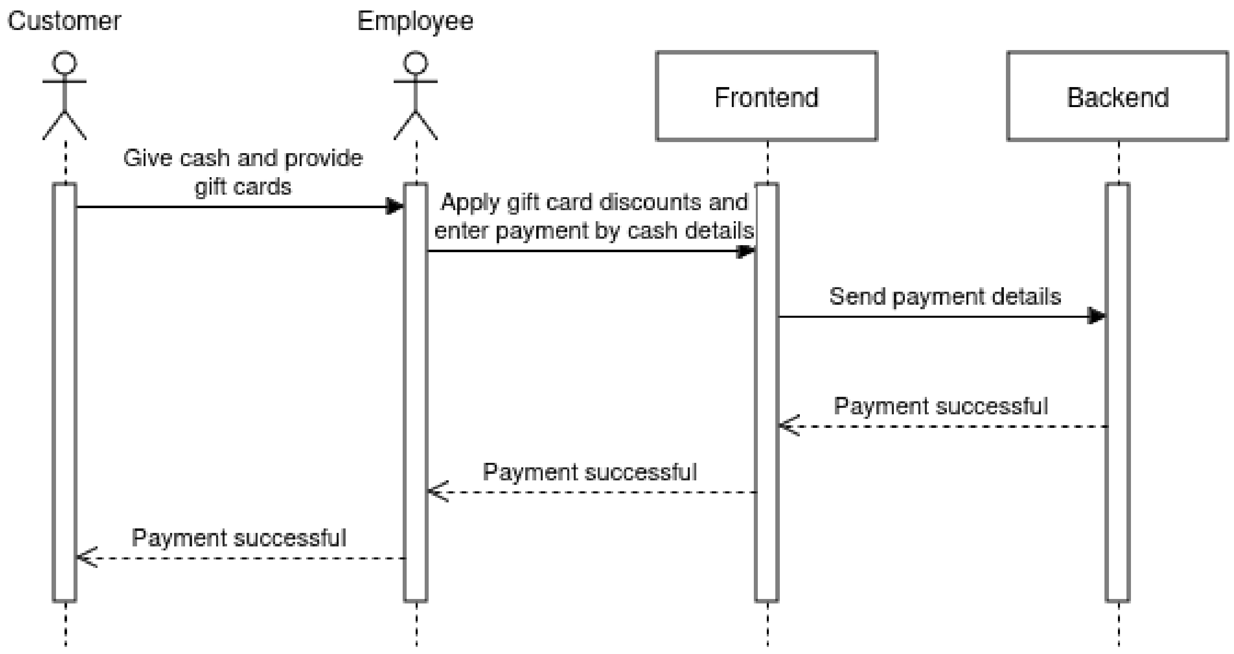

Figure 9. Pay for order by cash flow.

Figure 9 illustrates the process for completing a cash payment for an order. The steps are as follows:

1. The customer informs the employee of the items they wish to pay for.
2. The employee enters the payment details into the application's interface and confirms the payment.
3. The frontend sends a request to the backend, including the payment details.
4. The backend validates the payment information and saves it.
5. The backend returns to the frontend that the payment was successfully processed.

2.10 Pay for order by card flow

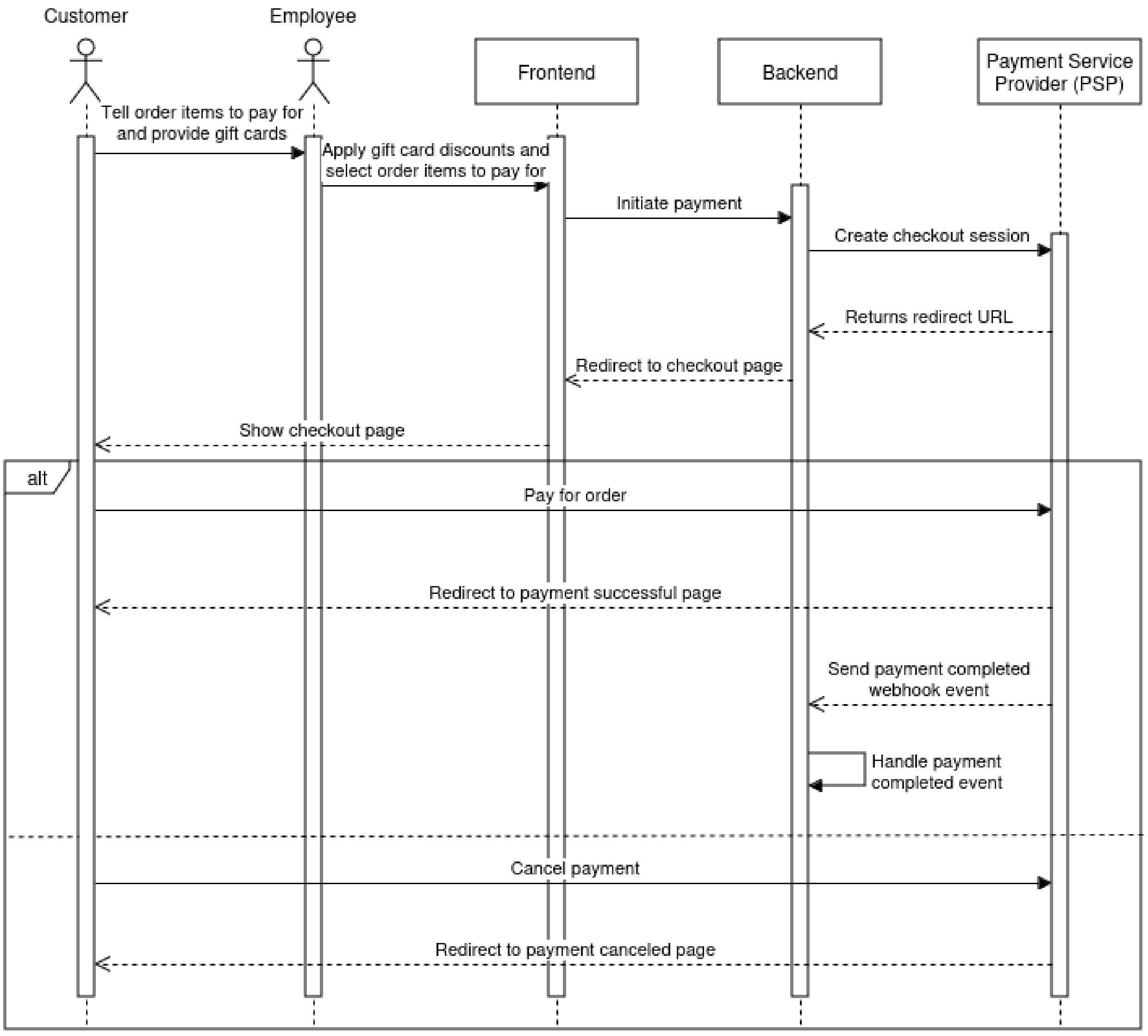
Customer
Employee
Frontend
Backend
Payment Service Provider (PSP)
Tell order items to pay for and provide gift cards
Apply gift card discounts and select order items to pay for
Initiate payment
Create checkout session
Returns redirect URL
Redirect to checkout page
Show checkout page
alt
Pay for order
Redirect to payment successful page
Send payment completed webhook event
Handle payment completed event
Cancel payment
Redirect to payment canceled page

Figure 10. Pay for order by card flow.

Figure 10 illustrates the process of paying for an order by card. The steps are as follows:

1. The customer selects the items to pay for.
2. The frontend sends a request to the backend to initiate the payment process for the selected items.
3. The backend sends a request to the Payment Service Provider (PSP) to create a checkout session for the selected items.
4. The PSP responds with a redirect URL, which the customer uses to complete the payment.
5. The backend redirects the customer to the PSP's payment page.
   - If the customer completes the payment, the PSP redirects him to a confirmation page indicating that the payment was successful. The PSP also sends a webhook event to the backend with the details of the completed payment.

- If the customer cancels the payment, the PSP redirects them to a page, displaying that the payment was canceled.

2.11 Reservation creation flow

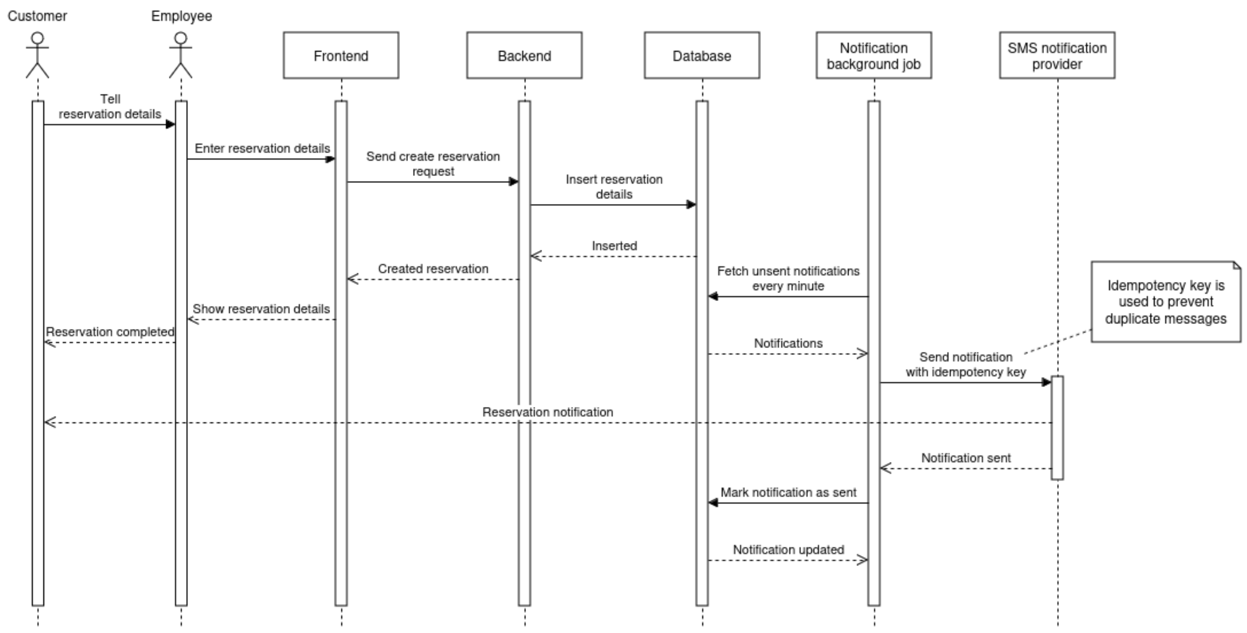
Customer
Employee
Frontend
Backend
Database
Notification background job
SMS notification provider
Tell reservation details
Enter reservation details
Send create reservation request
Insert reservation details
Inserted
Created reservation
Fetch unsent notifications every minute
Show reservation details
Reservation completed
Notifications
Idempotency key is used to prevent duplicate messages
Send notification with idempotency key
Reservation notification
Notification sent
Mark notification as sent
Notification updated

Figure 11. Reservation creation flow.

Figure 11 illustrates the process of making a reservation. The steps are as follows:

1. The customer provides their reservation details to the employee.
2. The employee enters the reservation details into the application's interface.
3. The frontend sends a request to the backend to create the reservation.
4. The backend validates the reservation details and stores them in the database.
5. The backend sends a response to the frontend confirming the successful creation of the reservation.
6. A background job responsible for notifications runs every minute, fetching unsent notifications from the database.
7. For each notification, the background job performs the following:
   7.1. Sends a request to the SMS Notification Provider to dispatch the notification. The request includes an idempotency key (Notification ID) to ensure that the notification is delivered only once.
   7.2. Marks the notification as sent in the database by updating the SentAt timestamp.

2.12 Reservation creation flow (2)

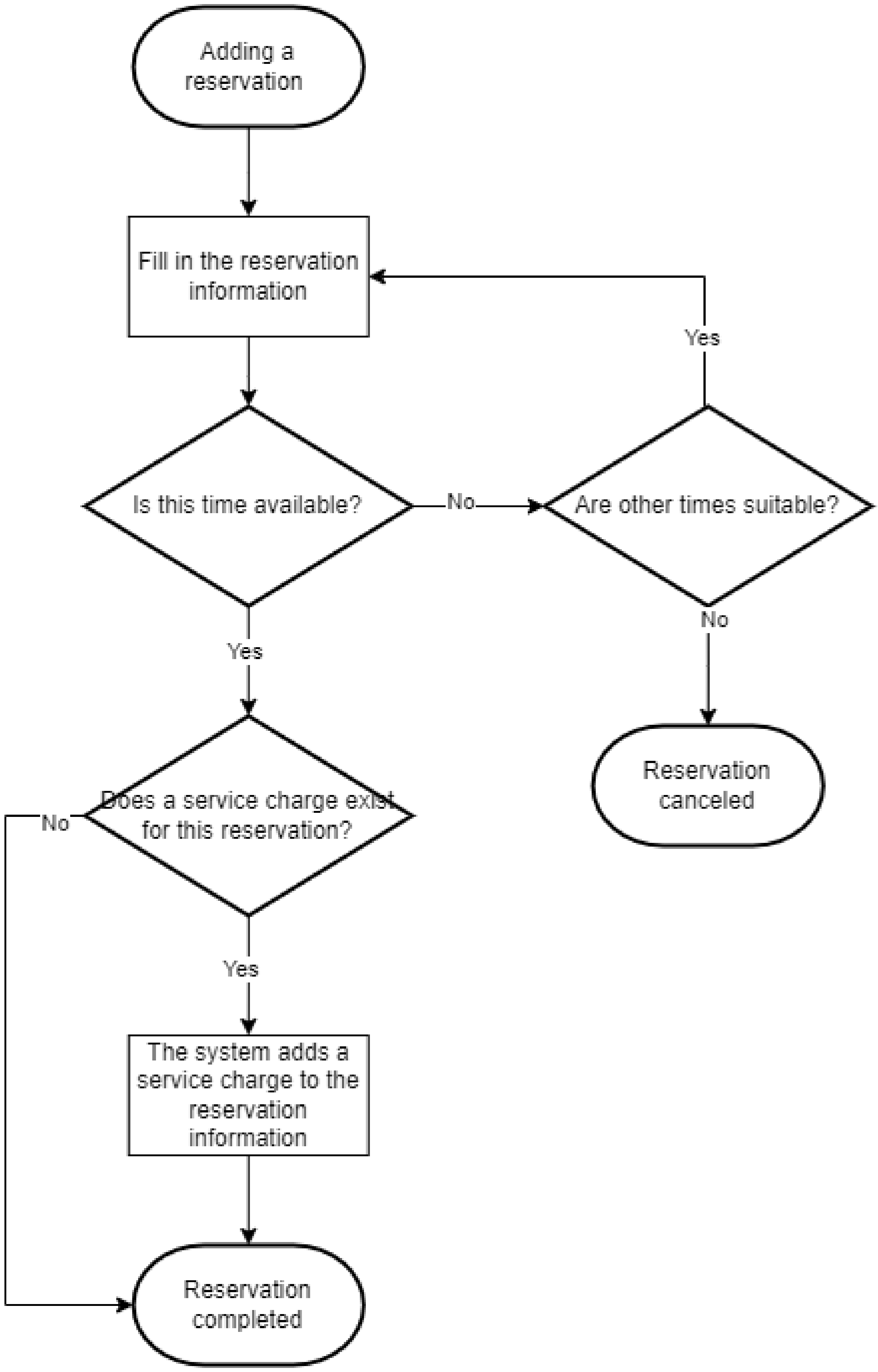
Adding a reservation
Fill in the reservation information
Is this time available?
No
Are other times suitable?
Yes
No
Reservation canceled
Yes
Does a service charge exist for this reservation?
No
Yes
The system adds a service charge to the reservation information
Reservation completed

Figure 12. Reservation creation flow (2).

Figure 12 illustrates the process of creating a reservation. The steps are as follows:

1. The employee fills in the reservation information.
2. The system checks if the selected time is available for the reservation.
3. If time is not available: the employee checks if other times are suitable for the customer.
   - If other times are suitable: refill the reservation information.
   - If other times are not suitable: The reservation is canceled due to unsuitable times.
4. If time is available: the system checks if the selected service has a service charge.
   - If the charge does not exist: the reservation is completed.
   - If a charge exists: the system adds service charge information to the reservation so that it can be paid together with the order.
     - The reservation is completed.

2.13 Reservation modification flow

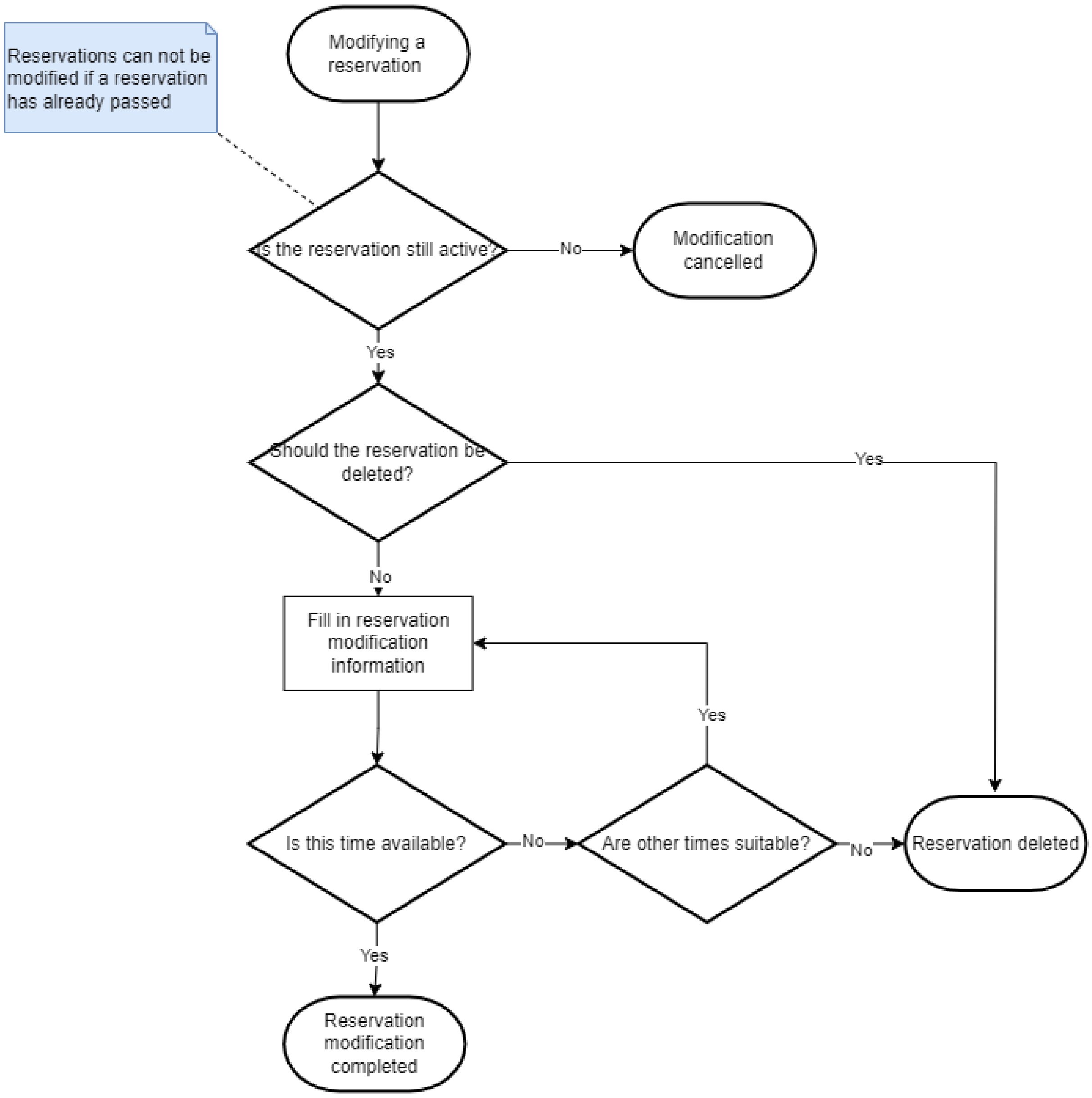
Reservations can not be modified if a reservation has already passed
Modifying a reservation
Is the reservation still active?
No
Modification cancelled
Yes
Should the reservation be deleted?
Yes
No
Fill in reservation modification information
Yes
Is this time available?
No
Are other times suitable?
No
Reservation deleted
Yes
Reservation modification completed

Figure 13. Reservation modification flow.

Figure 13 illustrates the process of modifying a reservation. The steps are as follows:

1. The employee checks if a reservation has not yet passed and if it is still in a suitable time interval to be modified.
2. If the reservation is not suitable for modifying: the modification is canceled.
3. If the reservation is suitable for modification: Employee checks if deletion was issued.
   - If the deletion was issued: the reservation is deleted.
   - If deletion was not issued, then modification is needed: the employee fills in new reservation information. The system checks if a new time is available.

- If time is not available: the employee must check if other times are suitable for the client.
  - If other times are not suitable: The reservation is deleted.
  - If other times are suitable: the employee must refill the reservation modification form.
- If time is available: the reservation modification is completed.

3 Entity data model

3.1 User entity data model

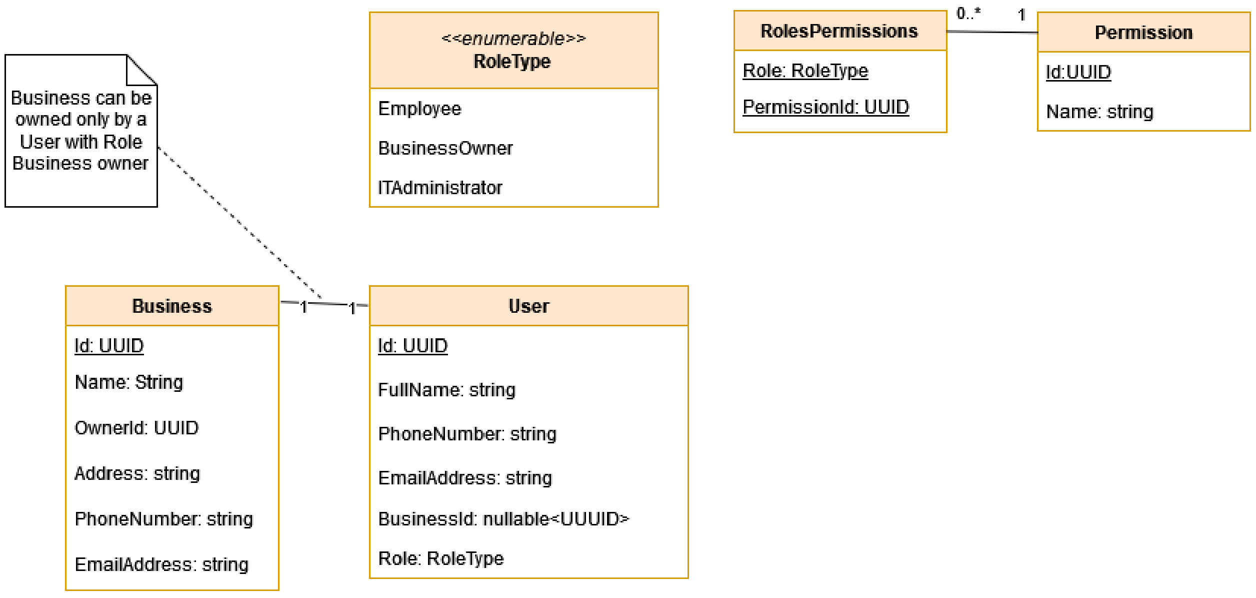

Figure 14. User entity data model.

Figure 14 illustrates user entity data model.

1. Business
   1.1. Attributes
      i. Id: UUID – Unique identifier for the business.
      ii. Name: String – The name of the business.
      iii. OwnerId: UUID – A reference to the UUID of a user who owns the business.
      iv. Address: String – The physical address of the business.
      v. PhoneNumber: String – The contact phone number for the business.
      vi. EmailAddress: String – The contact email address for the business.
   1.2. Relationships
      i. A business can only be owned by a user with the role of BusinessOwner.
2. User
   2.1. Attributes
      i. A business can only be owned by a user with the role of BusinessOwner.

   ii. FullName: String – The full name of the user.
   iii. PhoneNumber: String – The contact phone number of the user.
   iv. EmailAddress: String – The email address of the user.
   v. BusinessId: nullable<UUID> – Nullable field representing the business the user is associated with (can be empty if the user is not tied to any business).
   vi. Role: RoleType – Defines the role of the user in the system.

2.2. Relationships
   i. The BusinessId links the user to a business, but only if they are associated with one.

3. RoleType (Enumerated): Represents different roles a user can have
   3.1. Employee – Regular employee in the business.
   3.2. BusinessOwner – The owner of a business. Has all the permissions that Employee role has and more.
   3.3. ITAdministrator – User responsible for and or administrative functions. Has all the permissions that BusinessOwner role has and more.
4. RolesPermissions
   4.1. Attributes
      i. Role: RoleType – Defines the role of the user.
      ii. PermissionId: UUID – The permission identifier linked to a role.
   4.2. Relationships
      i. A many-to-many relationship exists between roles and permissions.
5. Permission
   5.1. Attributes
      i. Id: UUID – Unique identifier for the permission.
      ii. Name: String – The name of the permission.
   5.2. Relationships
      i. Each permission is linked to multiple roles through the RolesPermissions entity - one permission can be linked to many roles.

3.2 Order entity data model

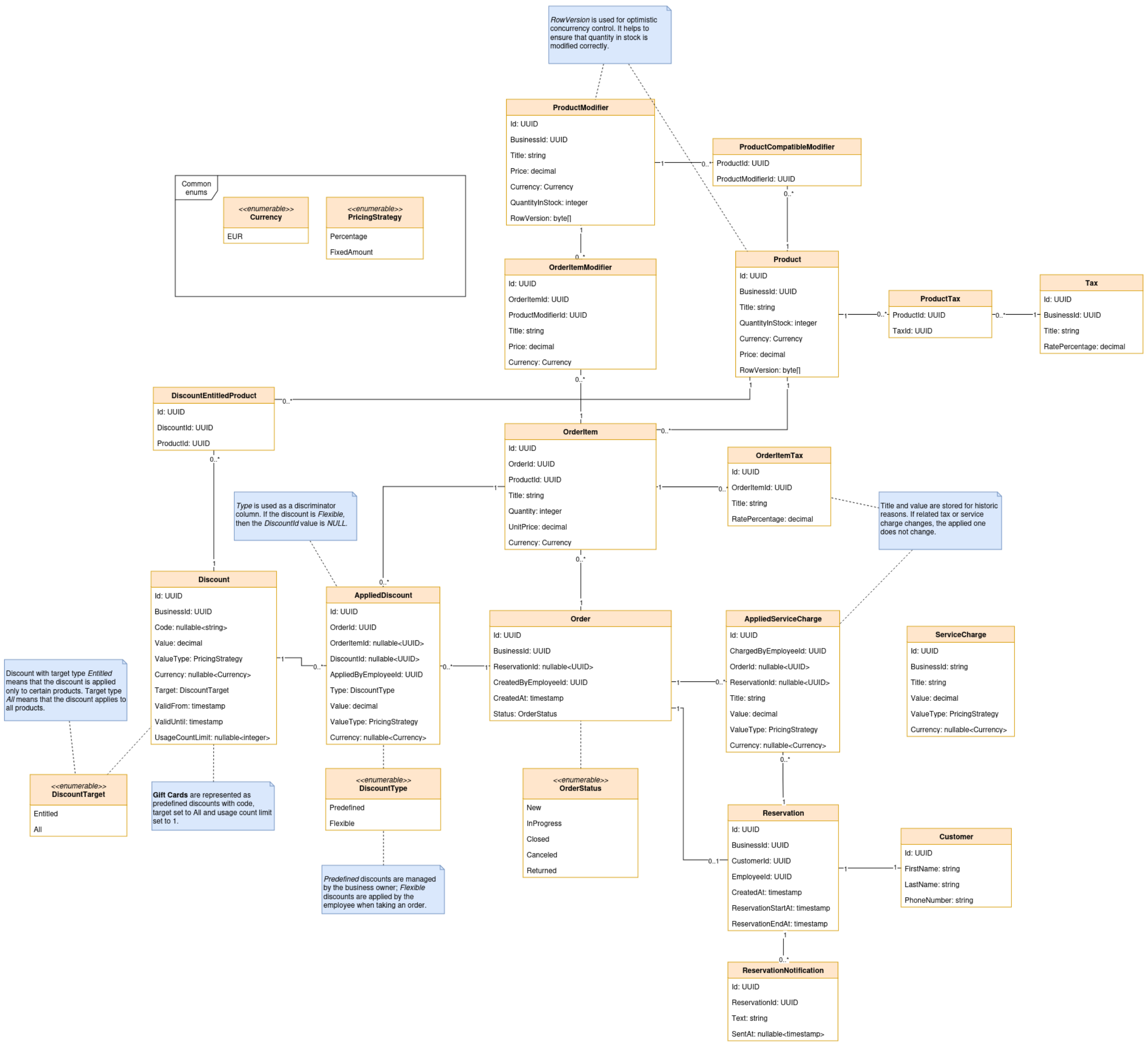

Figure 15. Order entity data model.

Product management is integrated together with order management. A business owner can create products and product modifiers for their business A product modifier is compatible with multiple products. A product modifier can be applied across multiple products. For example, "Coffee" or "Tea" could have "Medium Cup Size" and "Large Cup Size" product modifiers applied.

Additionally, different tax rates can be applied to products, such as groceries and alcohol. When an order is placed, the applied tax title and rate are stored in a separate table, linked to the order item. This ensures that any tax adjustments do not impact past orders, preserving the integrity of historical data.

Employees can apply service charges to both orders and reservations, either as a percentage of the order subtotal or as a fixed amount. Similar to taxes, the service charge title and the applied amount are stored in a separate table, ensuring that historical data is preserved and unaffected by future changes.

Business owners can create predefined discounts that can be applied either to specific (entitled) products or the entire order. Discounts may be activated using a code or automatically applied to all customers without the need for a code. These discounts can be structured as either a fixed amount or a percentage and can be set to work within a defined time range. In addition to predefined discounts, employees have the ability to apply flexible discounts directly to the order or individual order items when taking the order.

Gift cards are created by the business owner and represented as predefined discounts with some code and usage count limit set to 1. Usage count is calculated by counting related applied discounts.

3.3 Payments entity data model

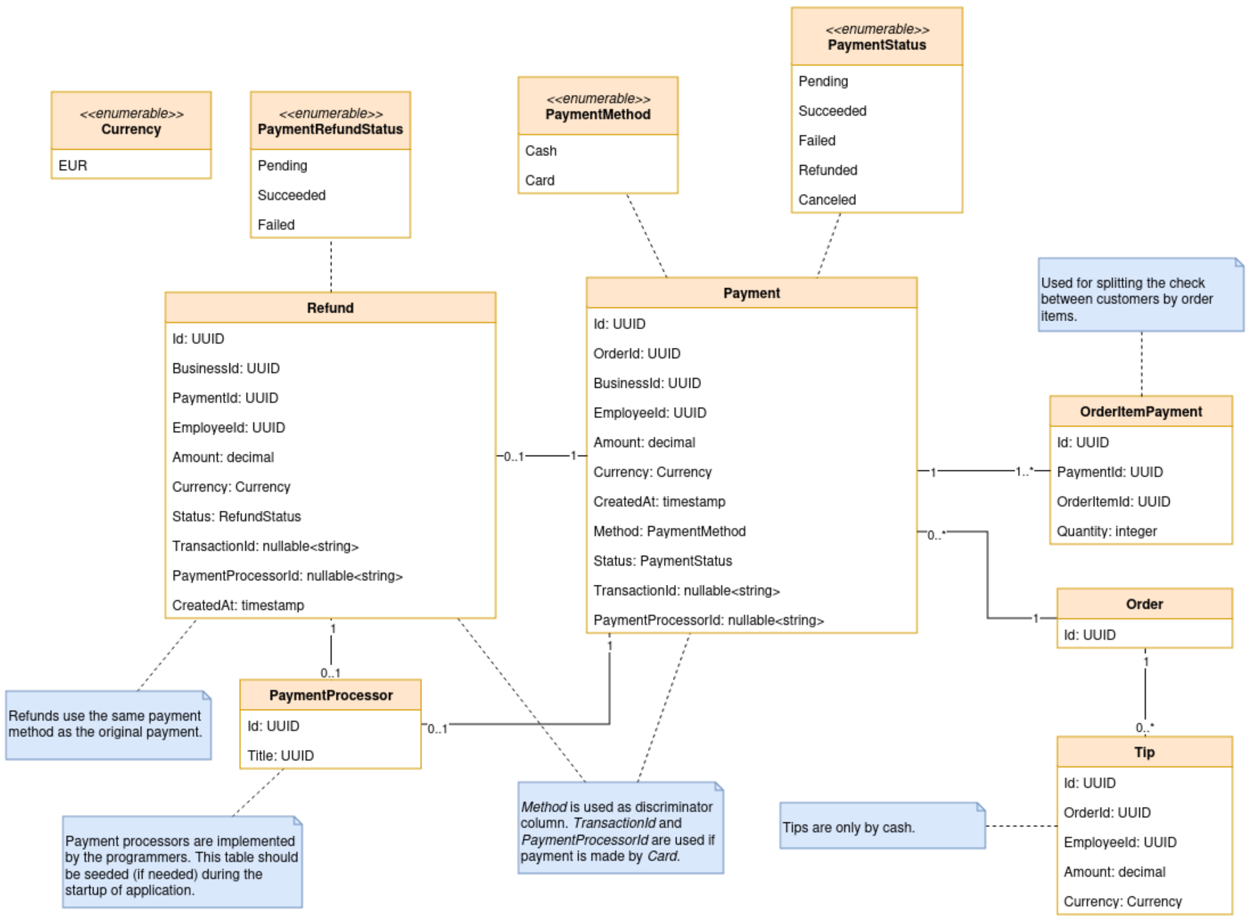

Figure 16. Payments entity data model.

Customer can pay for their order using cash or card. Customers can split the check by order items. After paying for the order, the customer can tip the employee by cash.

Payments can be refunded upon customer's request. Refunds use the same payment method as the original payment for the order. For example, if a payment was made by cash, it can only be refunded by cash.

4 Authentication and Authorization

4.1 Authentication and Authorization Flow

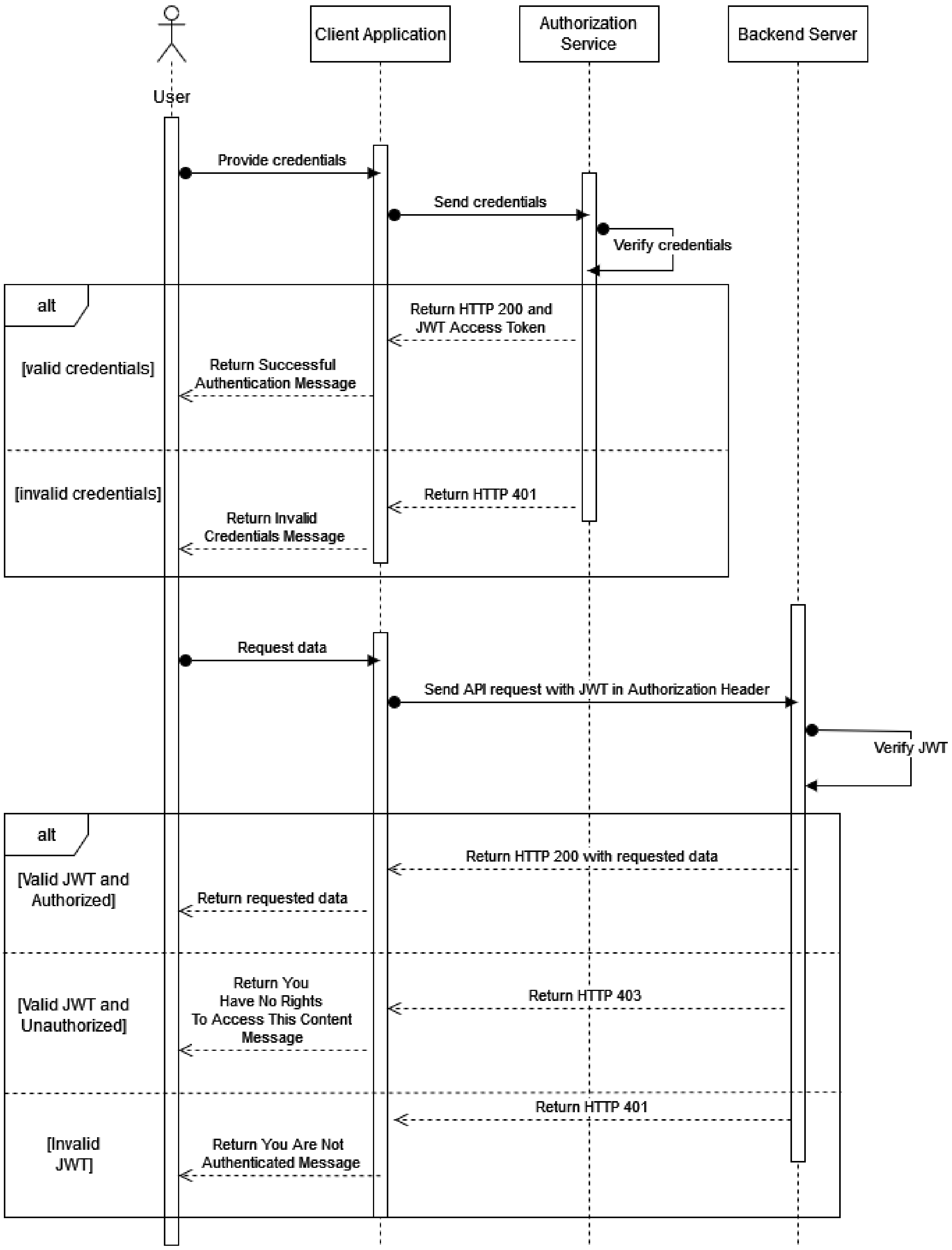
User
Client Application
Authorization Service
Backend Server
Provide credentials
Send credentials
Verify credentials
alt
[valid credentials]
Return HTTP 200 and JWT Access Token
Return Successful Authentication Message
[invalid credentials]
Return HTTP 401
Return Invalid Credentials Message
Request data
Send API request with JWT in Authorization Header
Verify JWT
alt
[Valid JWT and Authorized]
Return HTTP 200 with requested data
Return requested data
[Valid JWT and Unauthorized]
Return HTTP 403
Return You Have No Rights To Access This Content Message
[Invalid JWT]
Return HTTP 401
Return You Are Not Authenticated Message

Figure 17. Authorization flow.

Figure 17 The sequence diagram illustrates the OAuth2 Password flow using JWT for authorization. The JWT issued by external authorization service includes claims such as businessId and the user's role to indicate their specific business association and access level within the system. The user provides credentials to the client application, which sends them to the authorization service. If the credentials are valid, a JWT token is returned; otherwise, an error (HTTP 401) is returned. The client then uses the JWT to request data from the backend, where the server verifies the JWT and responds based on whether the user is authorized (HTTP 200 for success, 403 for unauthorized, or 401 for invalid token)

4.2 User Roles

In the system, there are three user roles with varying levels of access:

1. Employee: Has basic access rights.
2. BusinessOwner: Inherits all the rights of an Employee and has additional permissions to create, delete, and edit employees within their own business.
3. ITAdministrator (SuperAdmin): Has the highest level of access, inheriting all permissions of a BusinessOwner, and can create, delete, and edit any user in the system, regardless of business.

All actions (create, delete, edit) are restricted to logged-in and authorized users, ensuring that only those with the appropriate permissions can manage other users.

5 Module Diagram

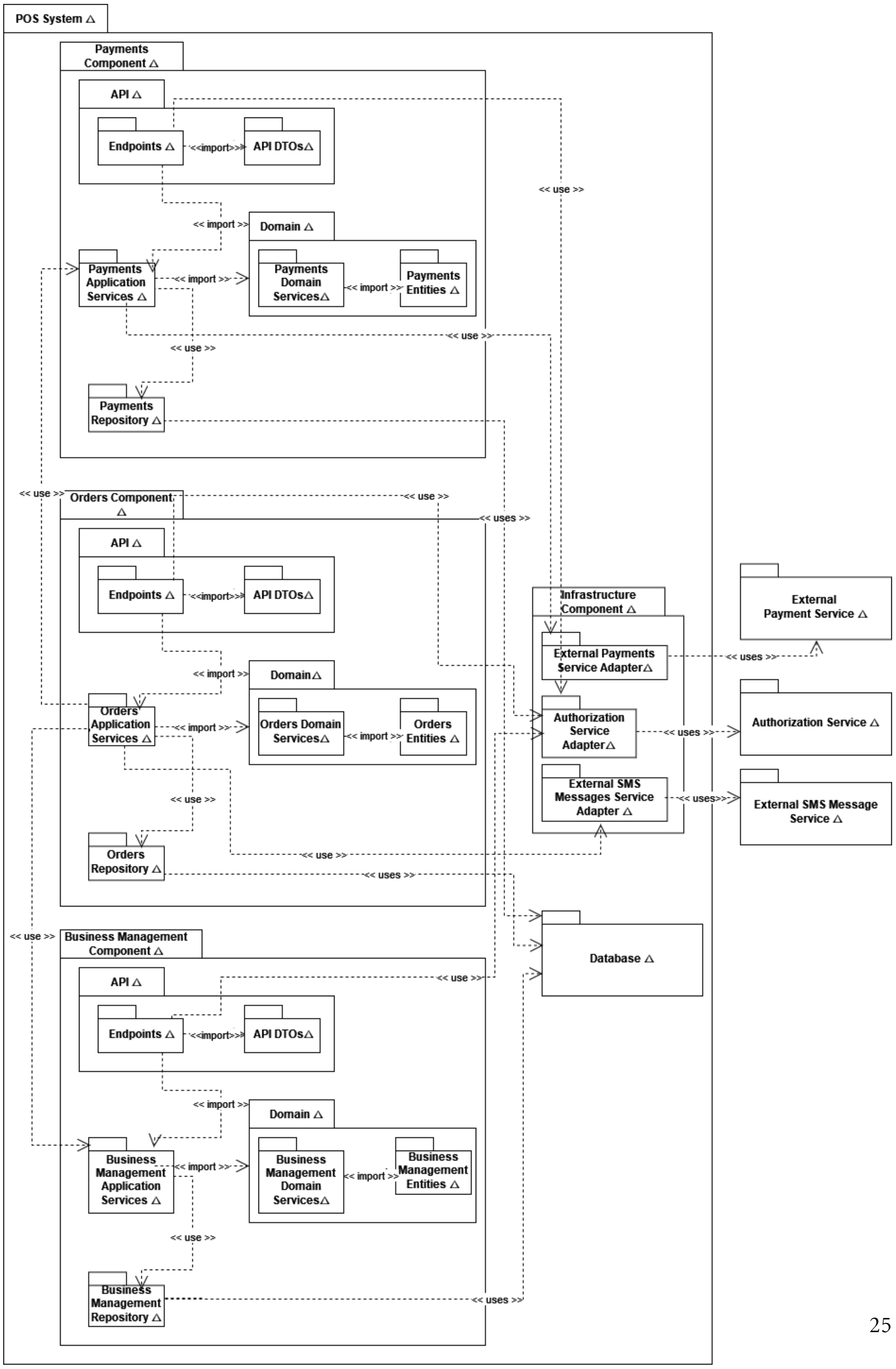

Figure 18. Module diagram.

Figure 18 represents a system architecture as a monolith that consists of Payments, Orders and Business Management components.

1. Payments Component:
    1.1. API: Contains DTOs and Endpoints that handle requests related to payments and use the Authorization Service Adapter for access control and authorization validation.
    1.2. Payments Application Services handle application flow for payments. It imports and utilizes Payments Domain Services and Entities from the Domain Layer.
    1.3. Domain: Contains Payments Domain Services and Payments Entities to handle the business rules, domain logic and data structures.
    1.4. Payments Repository manages database access for payment-related data.
2. Orders Component:
    2.1. API: Contains DTOs and Endpoints that handle requests related to orders and use the Authorization Service Adapter for access control and authorization validation.
    2.2. Orders Application Services manage the application flow related to orders. Uses the Payments Services from the Payments Component's Application Layer to handle interactions between orders and payments. It imports the Orders Domain Services from the Domain Layer. Uses External SMS Service Adapter to send notifications related to order statuses. Orders Services use the Business Management Services to determine which employee is responsible for managing or fulfilling the order.
    2.3. Domain: Manages Orders Domain Services and Orders Entities to handle domain logic, business rules and order entities.
    2.4. Orders Repository manages database access for orders data.
3. Business Management Component:
    3.1. API: Contains DTOs and Endpoints that handle requests related to business management and use the Authorization Service Adapter for access control and authorization validation.
    3.2. Business Management Application Services orchestrate application flow related to business management.
    3.3. Domain: Contains Business Management Domain Services and Business Management Entities to handle business rules, domain logic.
    3.4. Business Management Repository manages database access.
4. Infrastructure Component - This component consists of adapters that interact with external services:
    4.1. External Payments Service Adapter: Interacts with the external payment service.
    4.2. Authorization Service Adapter: Handles communication with the authorization service.
    4.3. External SMS Messages Service Adapter: Interfaces with an external SMS messaging service.
5. External Services:
    5.1. External Payment Service: An external system for handling payments.

    5.2. Authorization Service: Manages authorization for user access and roles.
    5.3. External SMS Message Service: Used for sending SMS notifications.
6. Database - A centralized database being used across the various components through their respective repositories.

6 API endpoint contracts

The API contracts are available in the included *Champignons_API.yaml* file, which can be viewed using the Swagger editor. For optimal performance, all endpoints that return a list of items are required to implement pagination.