Vilnius University
Faculty of Mathematics and Informatics
Software Engineering

PoS System Technical Design

Software Design and Architecture 1st Lab

Vitas Eliseenko
Daniil Švager
Ernestas Karalius
Patricija Katinaitė
Vytenis Normantas
Pijus Sadauskas

Supervisor: Lekt. Vasilij Savin

2024

Contents

1. Introduction

This document outlines the design and architecture of a Software-as-a-Service (SaaS) platform tailored for small and medium-sized businesses in the catering (bars, cafes, restaurants) and beauty (barbers, hairdressers, spas) sectors. The platform is designed to streamline daily operations by providing a unified ***Point of Sale (PoS) system*** for managing orders, services, reservations, payments, and business management tasks.

Our system is exclusively employee-facing, meaning it is intended to be used by the business's staff to manage bookings, handle orders, process payments, and perform administrative tasks. All the interactions with customers are performed by employees. The platform ensures flexibility by enabling businesses to access different features based on their specific plan.

The core features of the system are:

- **Order Management**: Employees can create and modify orders, apply discounts, split checks, and handle multiple payment methods including cash, gift cards, and card payments.
- **Service Management**: The platform allows employees to manage service reservations, assign employees, and send customer notifications. Modifications and cancellations can be made upon customer request.

Throughout the document, we will illustrate the system's components with activity diagrams, class diagrams, and package diagrams, providing a structured view of how the system operates. The architecture ensures scalability, ease of use, and security, making it a reliable tool for businesses seeking to optimize their day-to-day operations.

2. Business flows

Below are provided activity diagrams describing the most important business flows of a system. In these diagrams you can see columns that correspond to different actors:

- **Customer** – describes customer needs
- **Employee** – describes employee's actions upon a customer request
- **UI** – describes what interaction with user interface needs to be performed
- **API** – describes which endpoint is used on request and what response is returned

- **System** – describes what actions are performed by the system itself
- **Database** – describes what queries are executed in a database
- **Payment provider** – describes what transactions are performed by a third-party payment provider.

2.1 Authentication flow

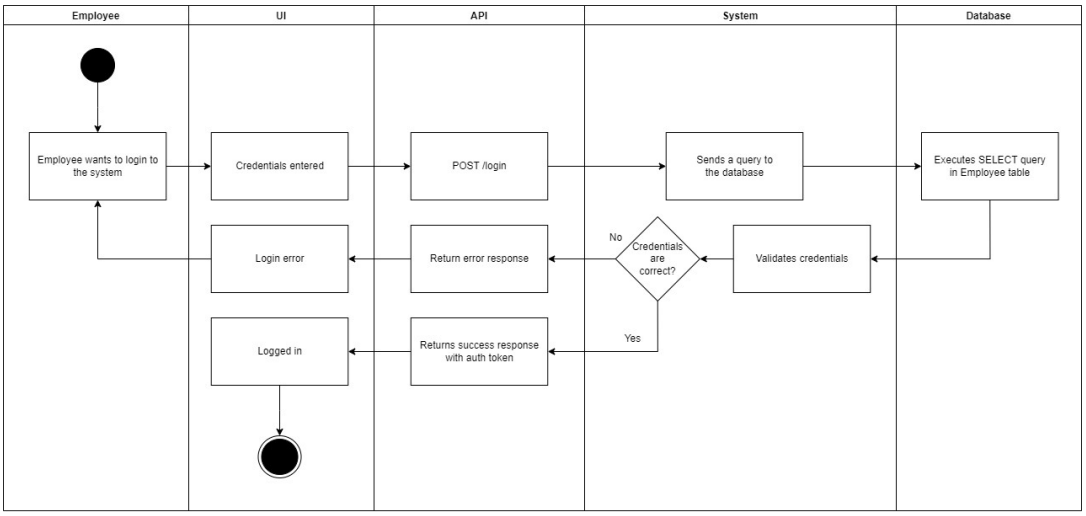

Figure 1: Authentication activity diagram

2.2 Create reservation flow

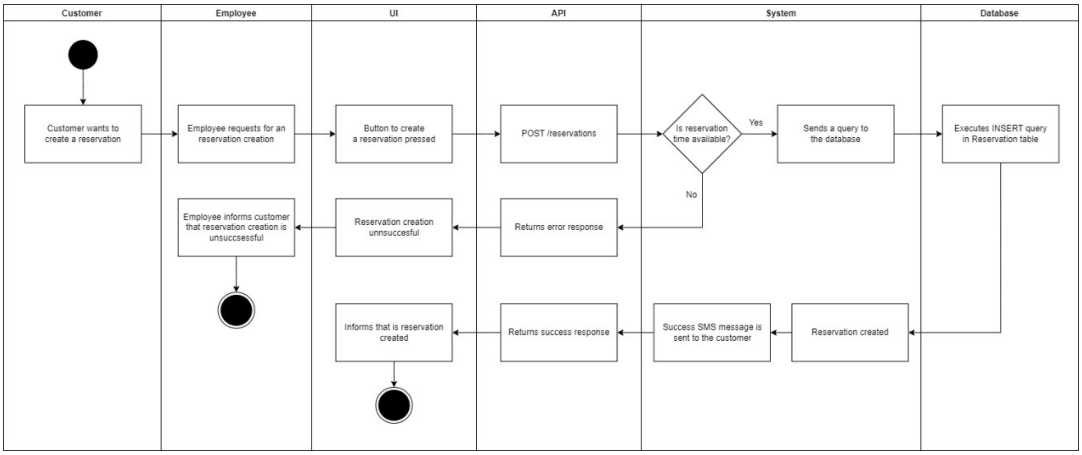

Figure 2: Create reservation activity diagram

2.3 Cancel reservation flow

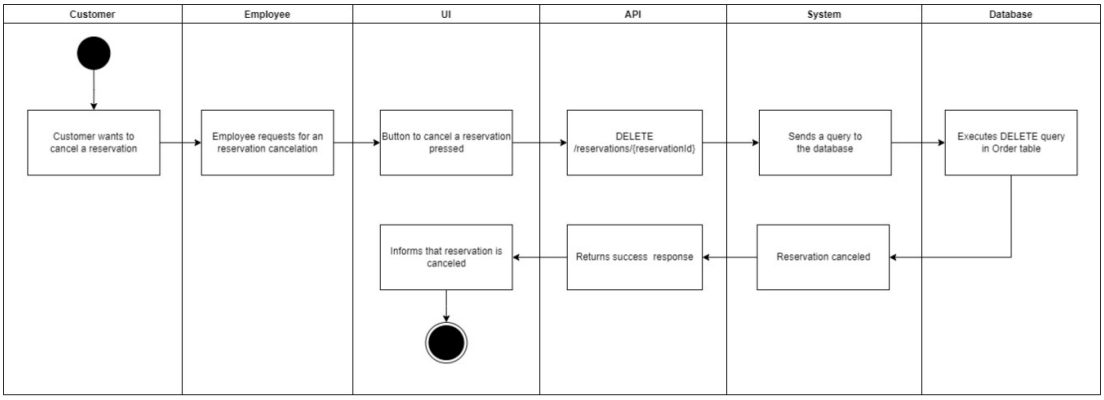

Figure 3: Cancel reservation activity diagram

2.4 Create order flow

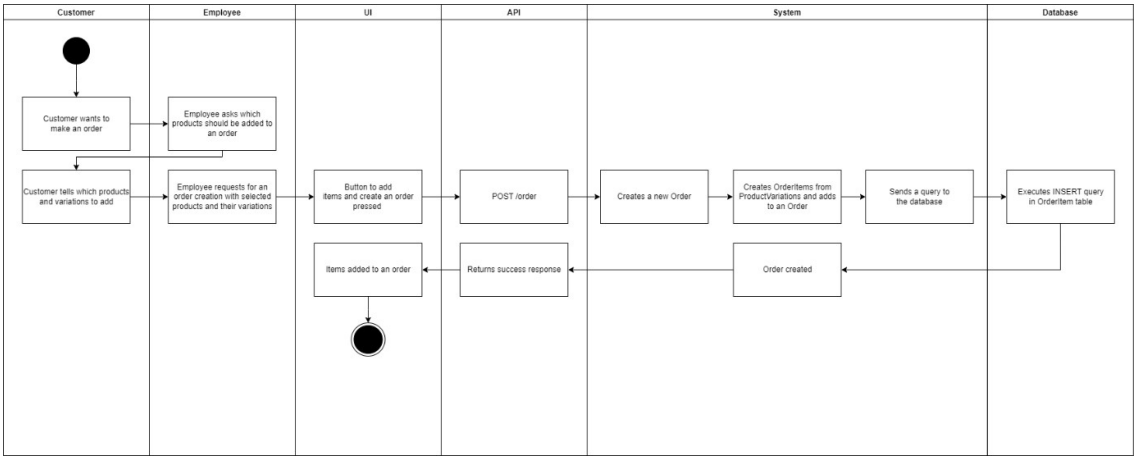

Figure 4: Create order activity diagram

2.5 Cancel order flow

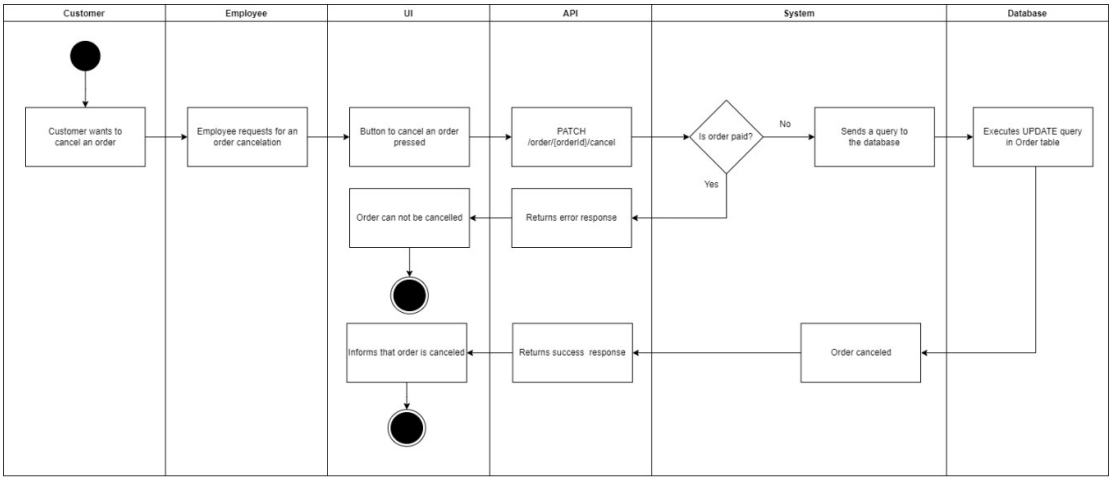

Figure 5: Cancel Order activity diagram

2.6 Pay for order flow

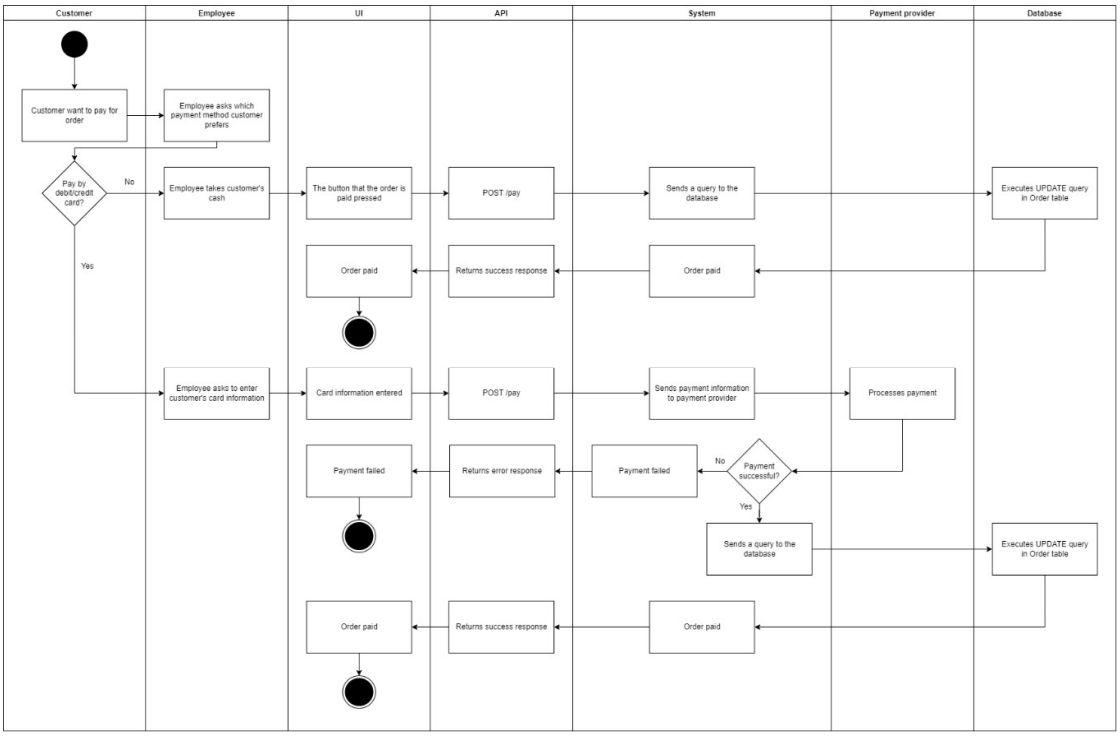

Figure 6: Pay for order activity diagram

2.7 Order refund flow

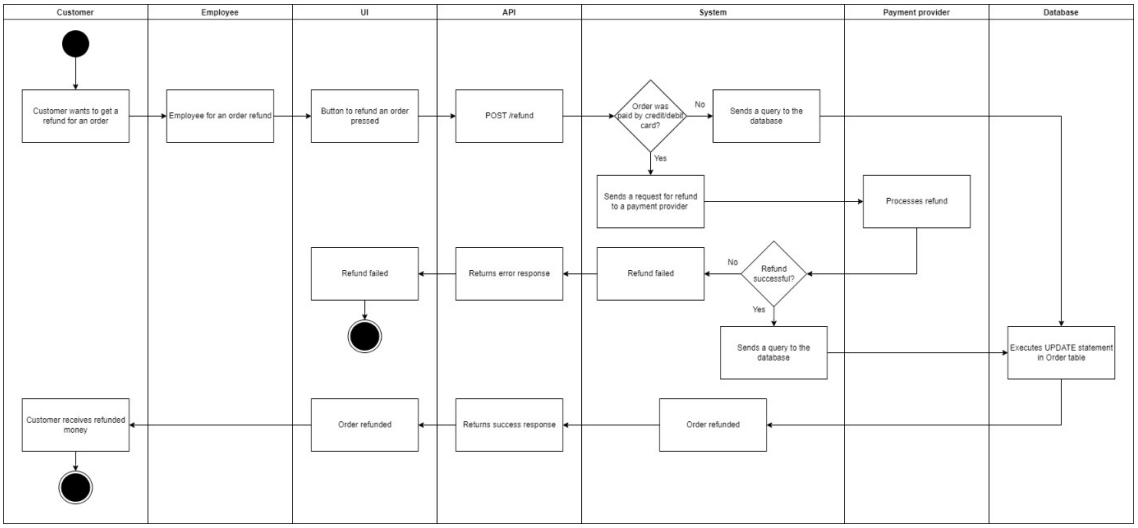

Figure 7: Order refund activity diagram

3. System high level architecture

We decided to split the whole PoS system into several components so that, if necessary, it would be possible to separate certain components into standalone microservices. This architecture implies better organization and code maintainability because each component can be created or updated independently.

Here we have following components:

- **Merchant** – this is the central component of a system. Since we are trying to create a system that could be used by many businesses, it is important to have an ability to manage merchants. Almost all other components depend on it, since most of the entities that exist in the system belong to a certain merchant.
- **Customer** – this is a component that is responsible for creation, modification and deletion of customers.
- **Order –** this is a component responsible for creation, modification and cancellation of orders. It also includes products management that lets to create/edit/delete products and their variations and discount management that allows to apply discounts to a specific product or the whole order.
- **Reservation** – this is a component that lets employees create reservations of services for customers. Also, it allows to cancel or modify a reservation upon a customer

request. It integrates with Customer component since when creating a reservation, customer's contact information should be provided, and with Employee component since each reservation has an assigned employee.

- **Employee** – this is a component that is responsible for creation, modification and deletion of employees. It is also responsible for employee authentication as a security measure.
- **Payment** – this is a component that integrates with some kind of payment provider and is responsible for accepting and refunding payments.

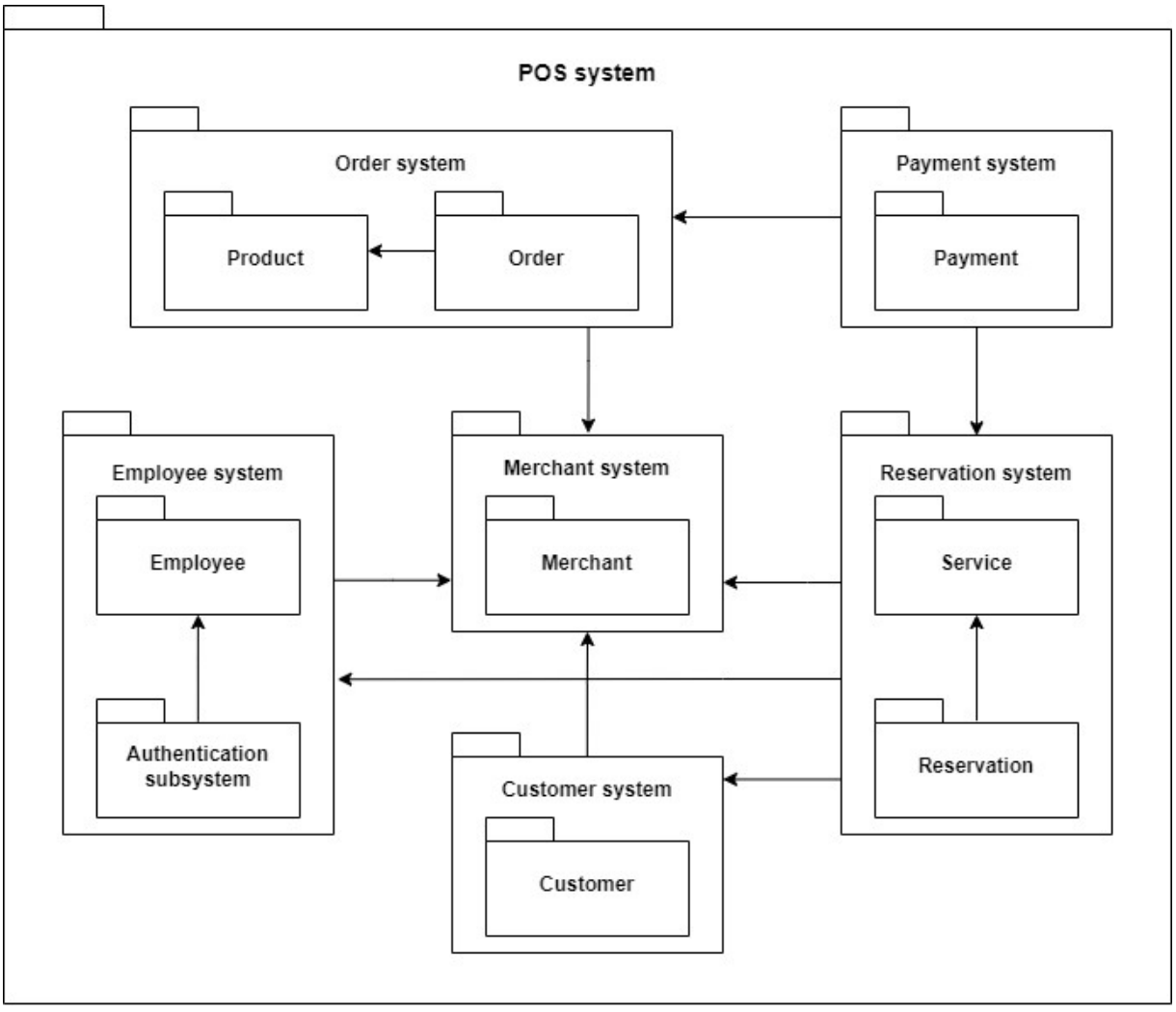

Figure 8: Package diagram

4. Data model

4.1 Class Diagram Overview

This class diagram (see Figure 9) represents the core structure of our Point of Sale (PoS) system. It defines how different entities (classes) in the system interact with one another and encapsulates both the data and relationships within the system. Each class represents a key component in the PoS, such as orders, payments, services, and customers, while the relationships between them illustrate how data is connected.

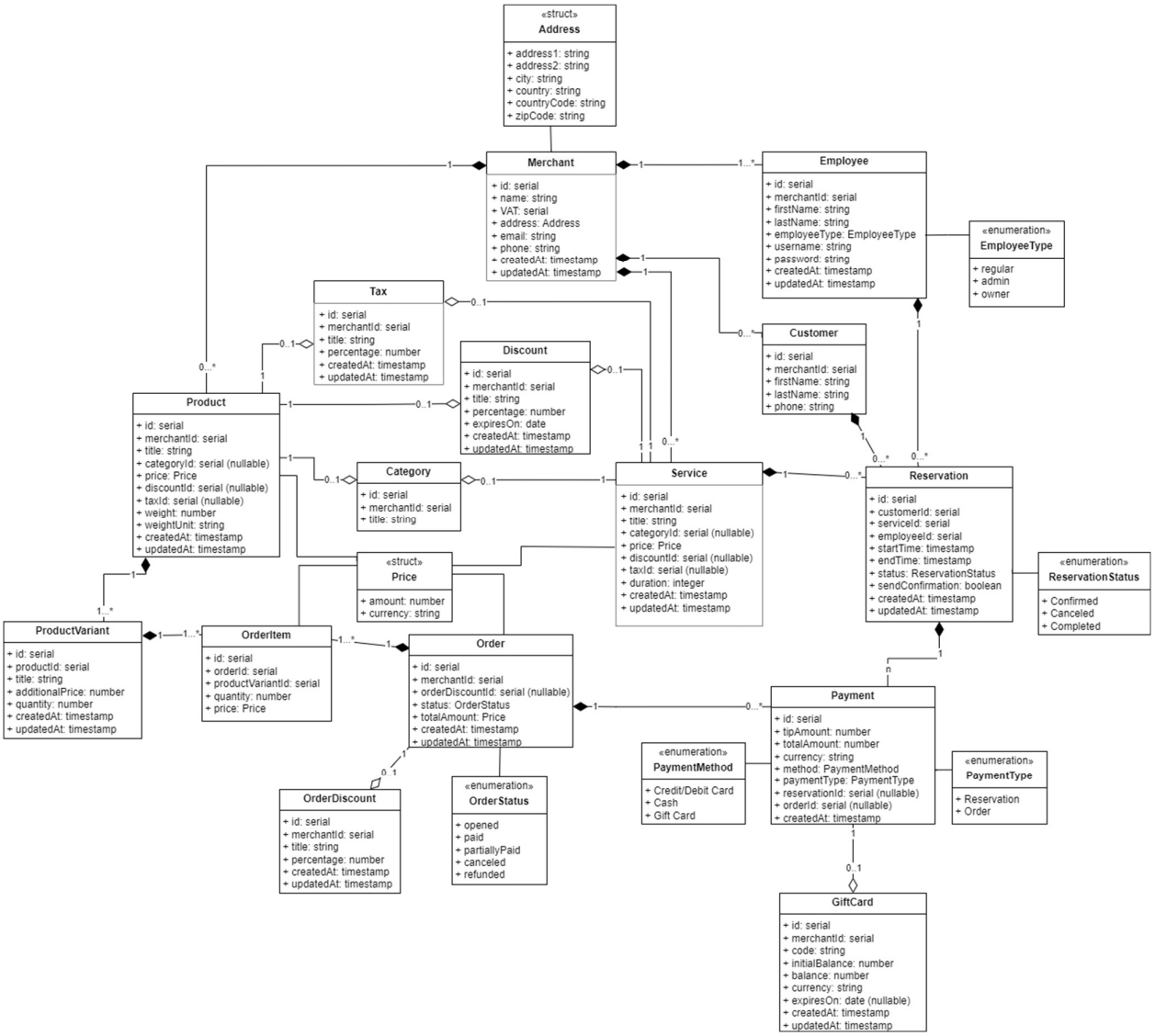

Figure 9: Class diagram

4.2 Relationships and Cardinalities

In our Point of Sale (PoS) system class diagram, the relationships between entities (classes) define how data is connected and related to each other.

Cardinality and Relationship descriptions in our Data model:

- **1 (one)**
  Example: 1 between Employee and Merchant means that each employee is tied to exactly one merchant.
- **0...1 (zero or one)**
  Example: 0...1 between Order and OrderDiscount means an order might have zero or one discount.
- **0...* (zero or many)**
  Example: 0...* between Customer and Reservation means a customer may have zero or many reservations.
- **1...* (one or many)**
  Example: 1...* between Product and ProductVariant means a Product can have one or more product variant.

- **Composition –** when a child object cannot exist without its parent object.
  Example: Customer (parent) and Reservation (child). A Reservation cannot exist without a customer who booked it, meaning if the Customer is deleted, the associated Reservation is also deleted (see Figure 10).

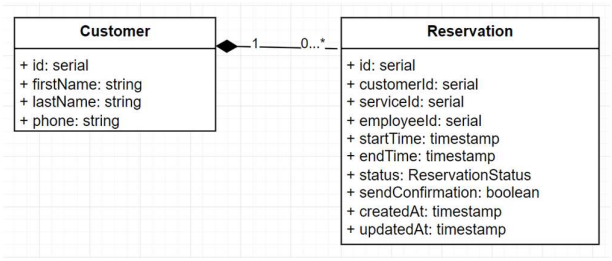

Figure 10: Composition example

- **Aggregation** - implies a relationship where the child can exist independently of the parent.
  Example: Payment (parent) and GiftCard (child). Delete the Payment and the GiftCard still exist (see Figure 11).

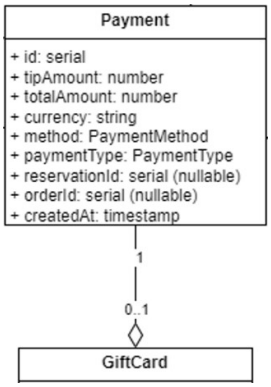

Figure 11: Aggregation example

4.3 Product entity

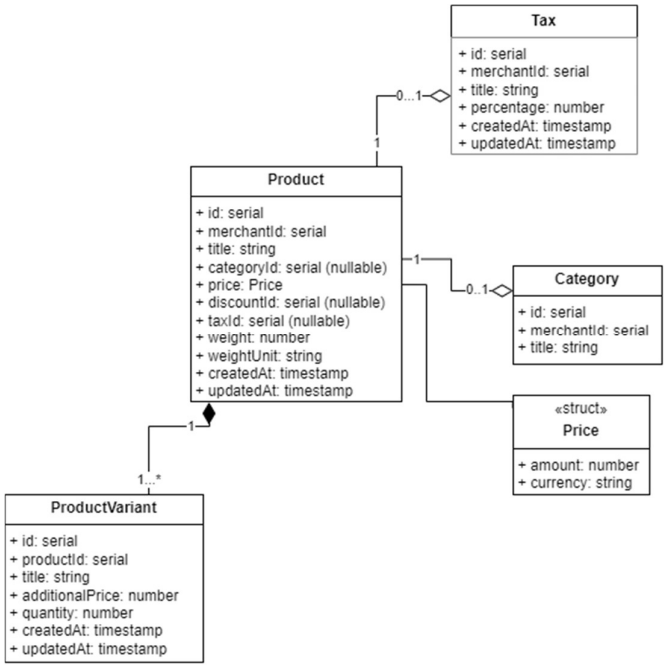

Figure 12: Product entity

Here you can see the *Product* entity with other related entities (see Figure 12).

Product is the base entity from which many variants can be derived from. A product has a base price, and all variants have some additionalPrice that is added to the product price when ordering.

Explanations of some of the Product fields:

- taxId. Businesses can choose a tax from a custom list to ensure they are automatically calculated.

- categoryId. Businesses can put their products in custom categories for organisational purposes.
- discountId. A product can have only **one** discount active.

When a Product is created, the user of the system will be asked to create variants. If they choose to create none, a default *ProductVariant* will be created with *additionalPrice* set to 0, and some title, like 'Default'.

When a *Product* has only **one** *ProductVariant*, the user of the system won't be able to select any variants when processing an order since it will be **automatically** selected.

If there are more than 1, an **indicator** will appear. After clicking it, the user of the system will be able to select a *ProductVariant*.

In the image below (see Figure 13) you can see a wireframe of how a product selection screen could look like.

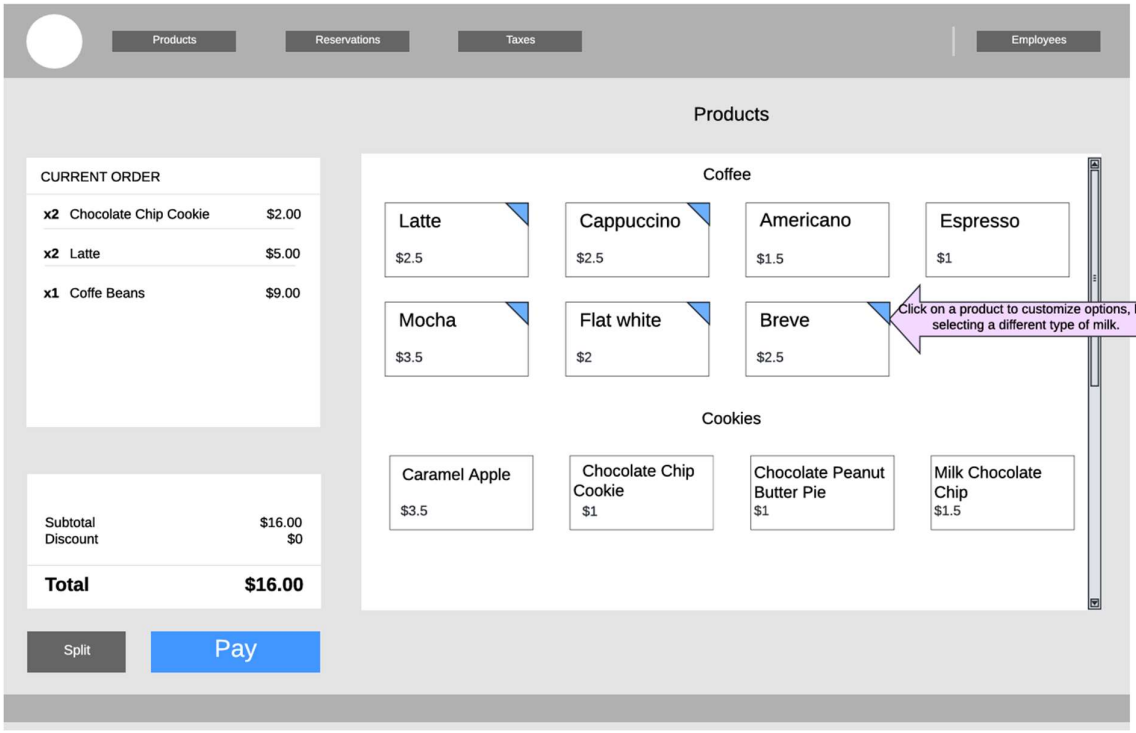

Figure 13: Main system page wireframe

Note: The wireframes which will be presented in this document are rough sketches meant to give an overall idea of the system's layout and functionality. They aren't meant to represent the final design or implementation but serve as a guide to understand the basic

structure and flow of the system. These wireframes provide a clear picture of key features, though specific details and design elements may change during development.

4.4 Order entity

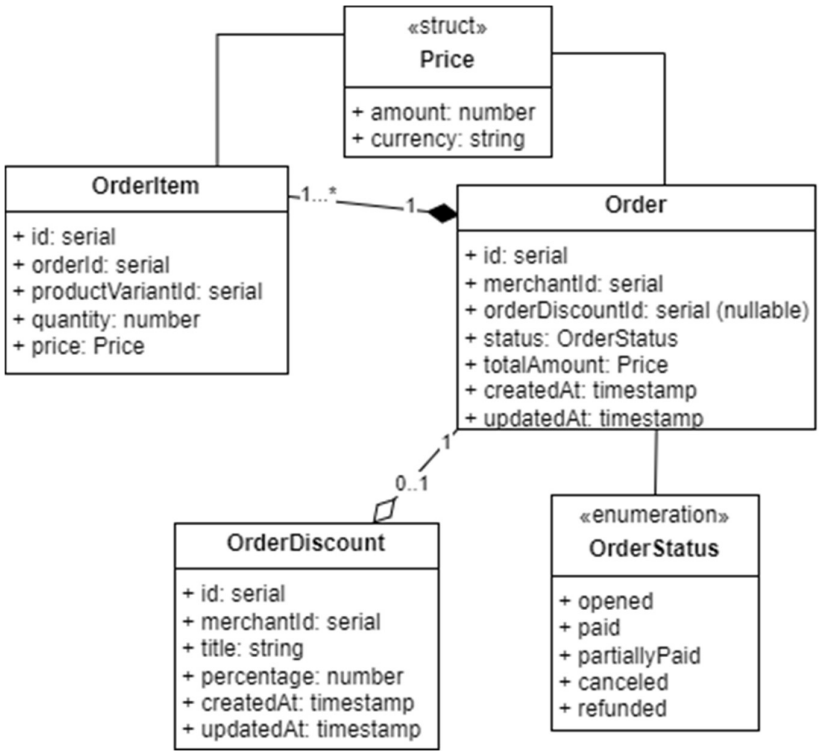

Figure 14: Order entity

Order it assembled by selecting *OrderItem* on the screen. OrderItem has a price that is calculated by adding the base price of the *Product* and the *additionalPrice* of the *ProductVariant* and multiplying by the product *Discount* percentage (see Figure 14).

After adding an *OrderItem*, the *totalAmount* field of *Order* is automatically recalculated by adding *price* * *quantity* for each *OrderItem*. If an *OrderDiscount* has been already applied, everything is multiplied by the *percentage* of *OrderDiscount* as well as the tax rate.

OrderDiscount do not stack, meaning only **one** can be applied.

The default *status* of the Order is ***opened***.

4.5 Service entity

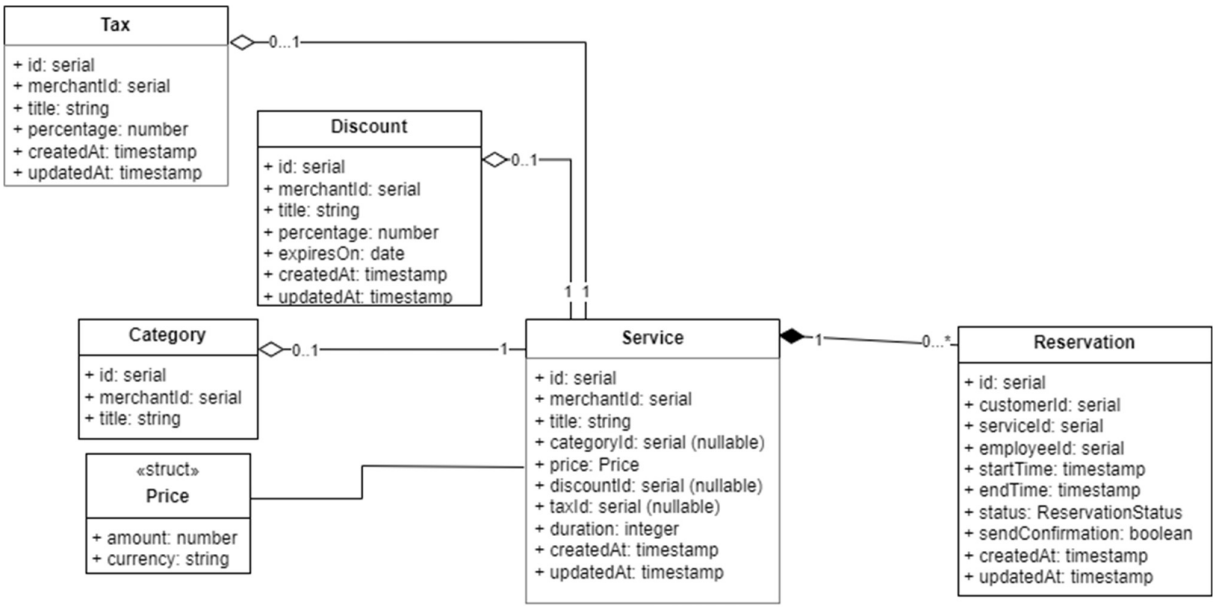

Figure 15: Service entity

The Service entity (see Figure 15) represents the services/ appointments that businesses offer to their customers. These services are essential since they can later be booked by customers.

Each service has:

- unique id
- merchantId linking it to a specific business
- title describing the service
- price, defined using a structured format to ensure the amount and currency are stored correctly
- duration field, which tells how long the service takes so it helps with scheduling and avoiding overlaps
- createdAt and updatedAt timestamps, that are used to track when the service was first created and last updated, useful for monitoring any changes.

Some fields are optional, like categoryId, which links the service to a category (e.g., "Haircuts" or "Spa Services"). If the service doesn't fit into a category, this field can be left empty, and the service will show up in a general "Other" section. Similarly, discountId and taxId are also optional because not every service will have a discount or be taxed.

4.6 Payment entity

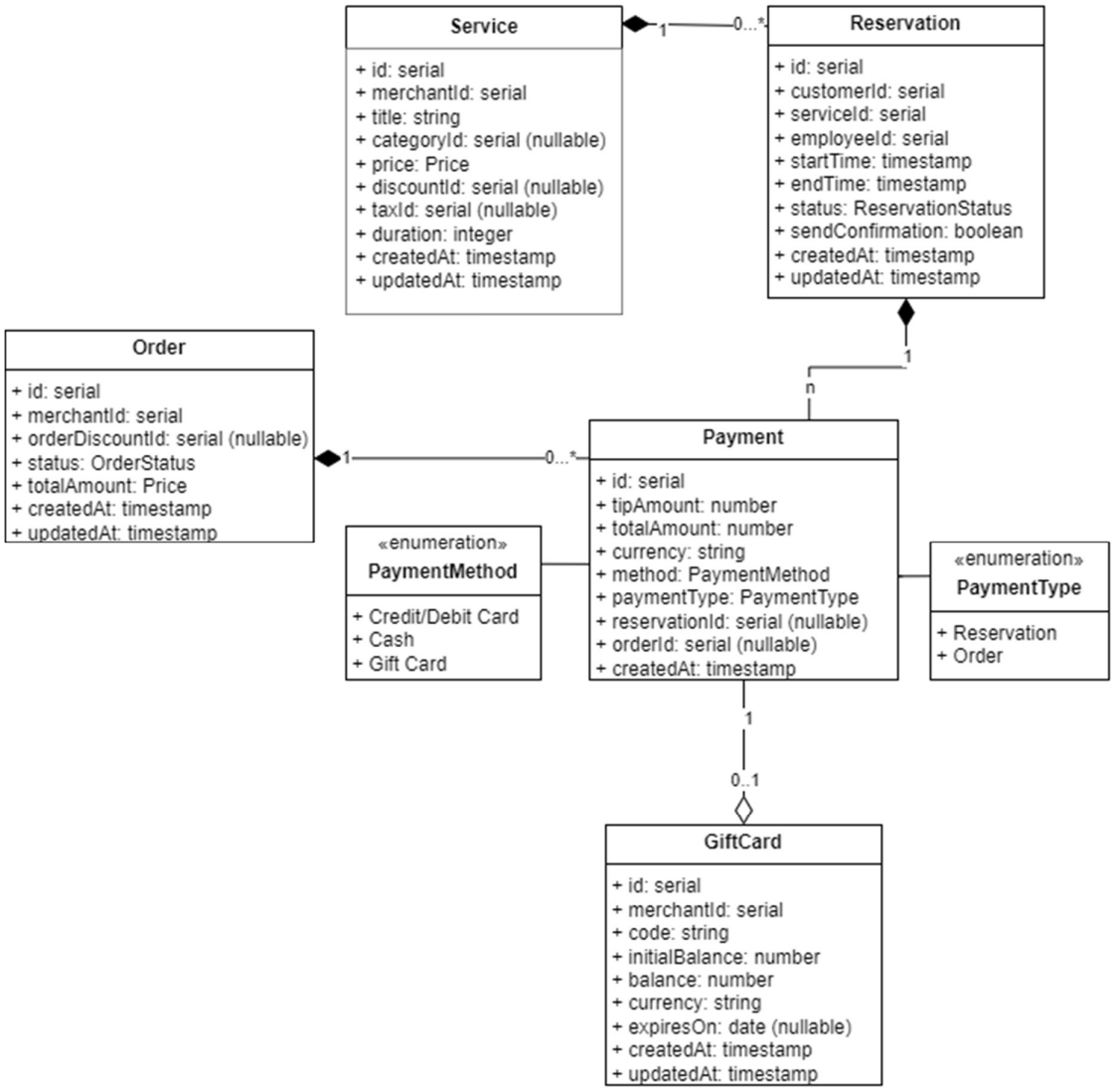

Figure 16: Payment entity

The Payment entity (see Figure 16) is designed to capture all payment transactions made within the system, whether for reservations (services) or product orders.

Each payment has:

- unique id
- tipAmount field, which can be used if the customer decides to leave a tip. If ***no tip*** is provided, this will **automatically** be set to **0**
- totalAmount - representing the total value of the payment
- currency indicating the type of currency used for the payment (e.g., USD, EUR)

- method field from the *PaymentMethod* enum, which shows how the payment was made (Credit/Debit Card, Cash or Gift Card).
- createdAt timestamp that records when the payment was made, which is useful for tracking and reporting purposes.
- paymentType field distinguishes whether the payment is for a reservation or an order. It stores either a reservationId (if the payment is for a service reservation) or an orderId (if it's for a product order). Only one of these fields will be filled based on the type of transaction.

It's important to note that ***services can only be paid for if a reservation has been made***. This ensures that every service-related payment is tied directly to a reservation, preventing payments from being made without a booking and easier time managing.

Also, if the payment method is a Gift card, the Payment entity is linked to the GiftCard entity, storing the gift card details used in the transaction.

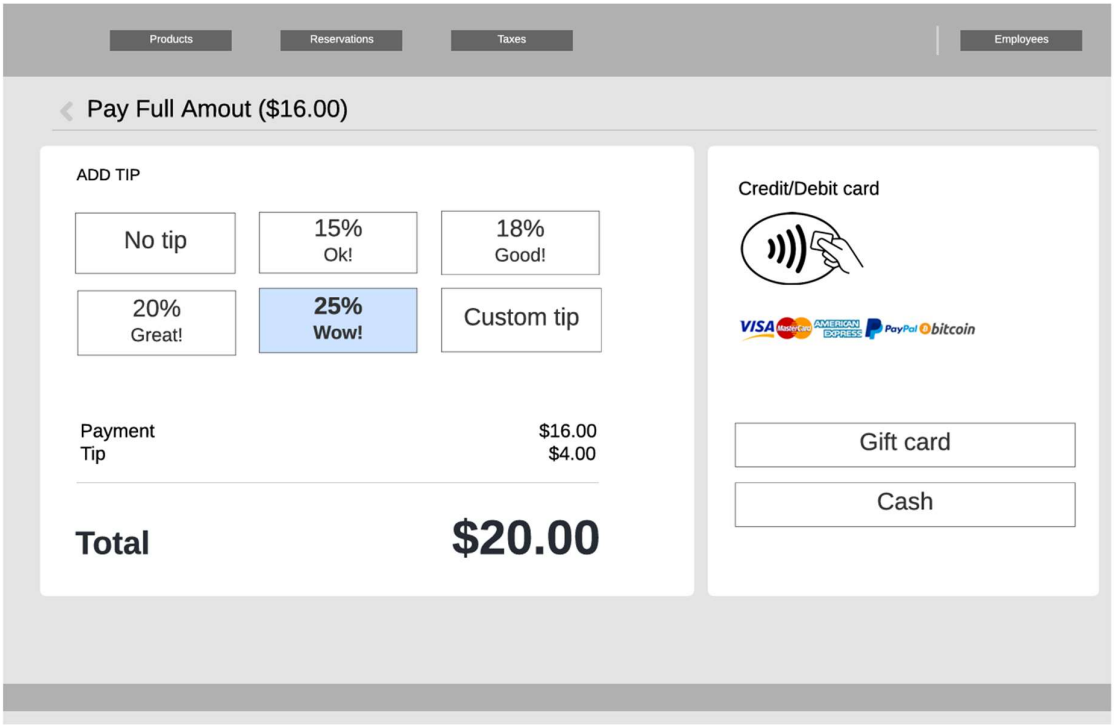

Figure 17: Payment screen wireframe

4.7 Reservation entity

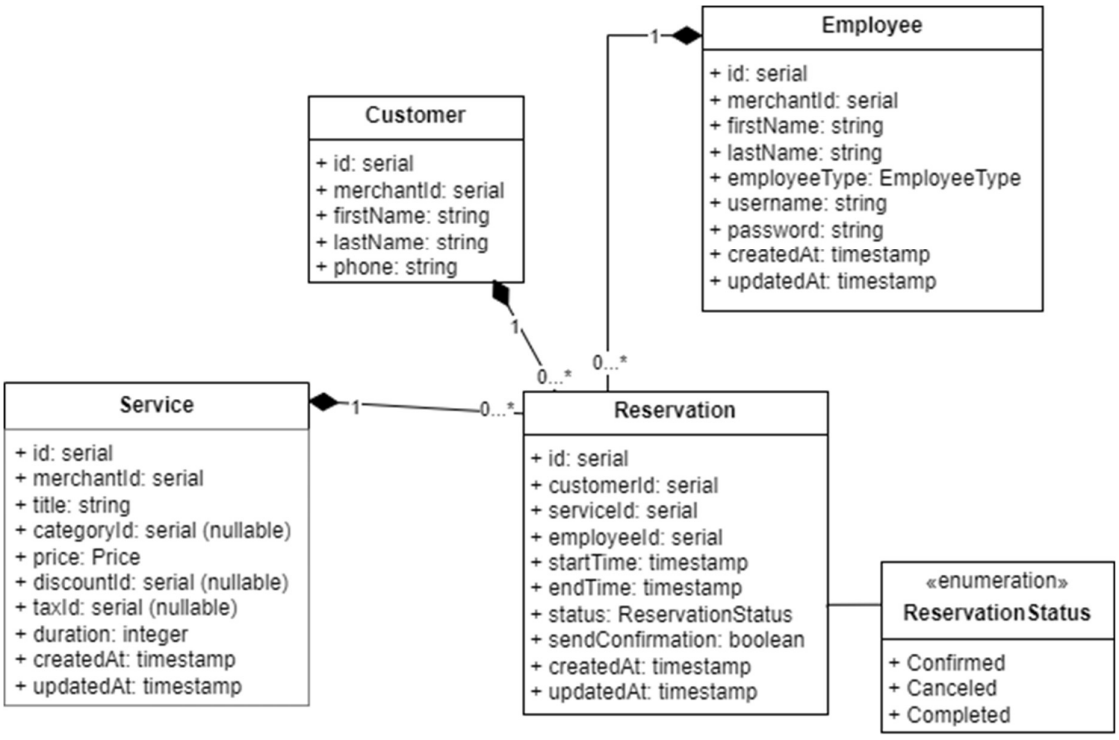

Figure 18: Reservation entity

The Reservation entity (see Figure 18) tracks a booking made by a customer for a specific service at a defined time and is associated with the **Customer**, the **Service**, and the **Employee** (who created the reservation). It records the employee responsible for the reservation, enabling the system to track which employee made which booking. It also includes timestamps for when the reservation was created and the end time, helping prevent overbooking by ensuring no conflicting reservations are made for the same service and time. The reservation's status (confirmed, canceled, completed) allows the system to track the booking's progress and final state. And the send confirmation field which allows the employee to choose whether or not to send a booking confirmation SMS when creating the reservation.

The wireframe below (see Figure 19) outlines the reservation creation process from the employee's perspective. The employee assists a customer in booking a service (a manicure).

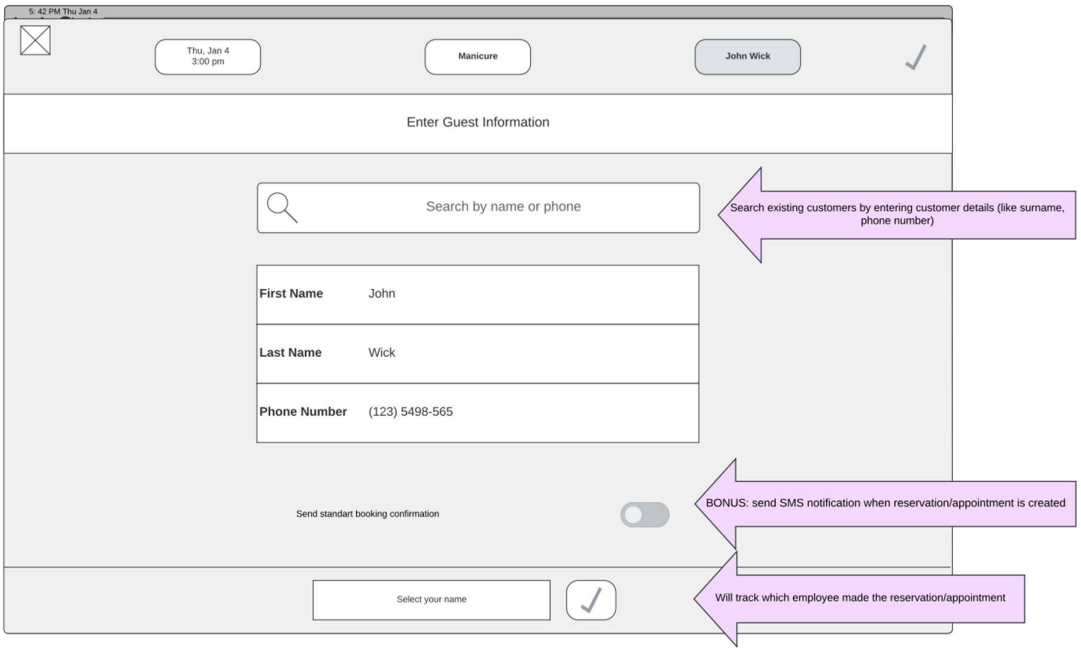

Figure 19: Reservation creation wireframe

Key steps include:

- **Service and time selection**

  The employee first selects the service the customer requested and finds the most suitable date and time. Once a time slot is clicked, the employee proceeds to the next step - entering customer information. The selected service, date and time are already displayed at the top of the form in this next step.
- **Customer lookup**

  The employee can search for an existing customer by entering details such as their name or phone number into the search bar. As the employee types, the system auto-suggests matching customers from the database to prevent duplicate entries and speed up the process of finding existing customers. The search bar acts as an **auto-complete** feature, helping employees quickly retrieve and select customer records. Once a customer is selected or found, their details are automatically populated in the form below. However, if the customer is new and doesn't exist in the database, the employee can **manually enter** their name, surname and phone number.
- **Send booking confirmation**

  At the bottom of the form, there's a toggle switch "Send standard booking confirmation." This toggle allows the employee to choose whether or not to send an

SMS confirmation to the customer when the reservation/appointment is created. Since this functionality is for bonus points, the toggle is set to **false** by default.
If the employee enables it, the system will automatically send a booking confirmation message with the reservation details (e.g., service, time, and location).

- **Finalize reservation**
  The employee selects his name and confirms the reservation by clicking the checkmark button at the bottom, completing the process and saving the reservation to the system. The system links the reservation with the customer and service, ensuring the selected timeslot is not double-booked for the same service, preventing overbooking.

The employee can also see a list of all reservations for a specific day, helping them manage their appointments efficiently (see Figure 20).

Key elements include:

- **Daily Overview**
  The employee selects a specific day to view. All reservations for that day are displayed in a clear list format.
- **Reservation Details**
- Each reservation includes essential information such as the customer's name, the service they booked, the appointment time, and the employee handling the appointment.
- **Walk-in Registrations**
  If a customer walks in without a prior reservation, the system allows employees to register these as new reservations, ensuring no appointment is left untracked.

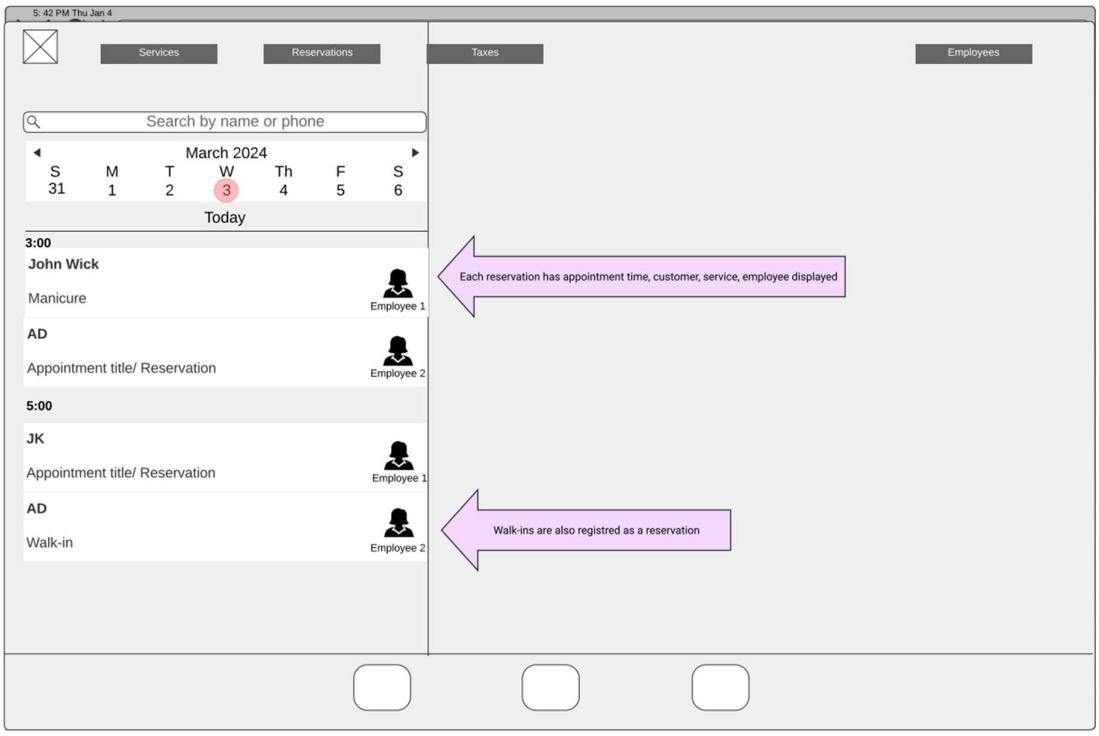

Figure 20: Reservation list wireframe

4.8 Category entity

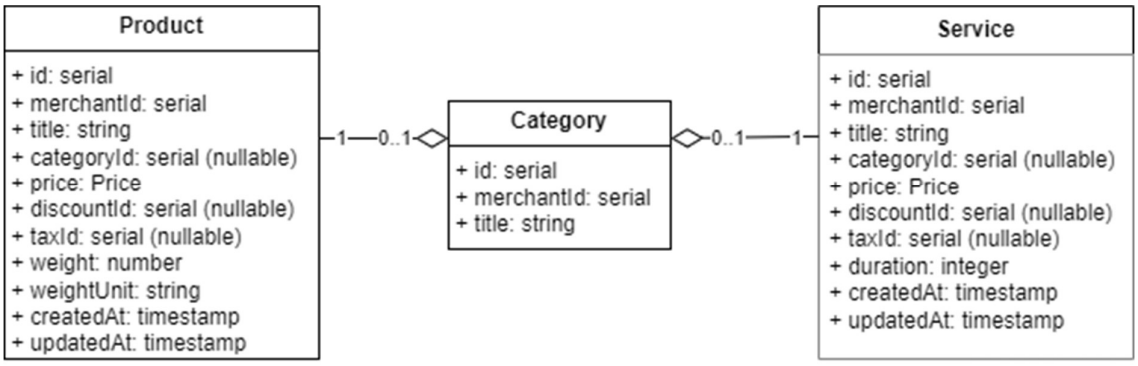

Figure 21: Category entity

The Category entity (see Figure 21) represents the different groups or classifications that can be assigned to products or services in the system. Categories help merchants organize their offerings into logical groups, making it easier to browse or manage.

Key fields include:

- id - unique identifier for each category.

- merchantId - links the category to a specific merchant, ensuring that each merchant manages their own categories.
- title - name of the category (e.g., "Coffee", "Cookies").

Categories are optional for both Product and Service entities, meaning a product or service can either belong to a category or remain uncategorized (0..1 cardinality). Each product or service can only belong to one category, but the same category can group multiple products or services, allowing for flexible organization within the system. Once again, since the category is an optional field, Product and Service will be placed in an “Others” category if no specific category is applied to them.

Categories can be created, edited, and associated with products through the system interface (see Figure 22).

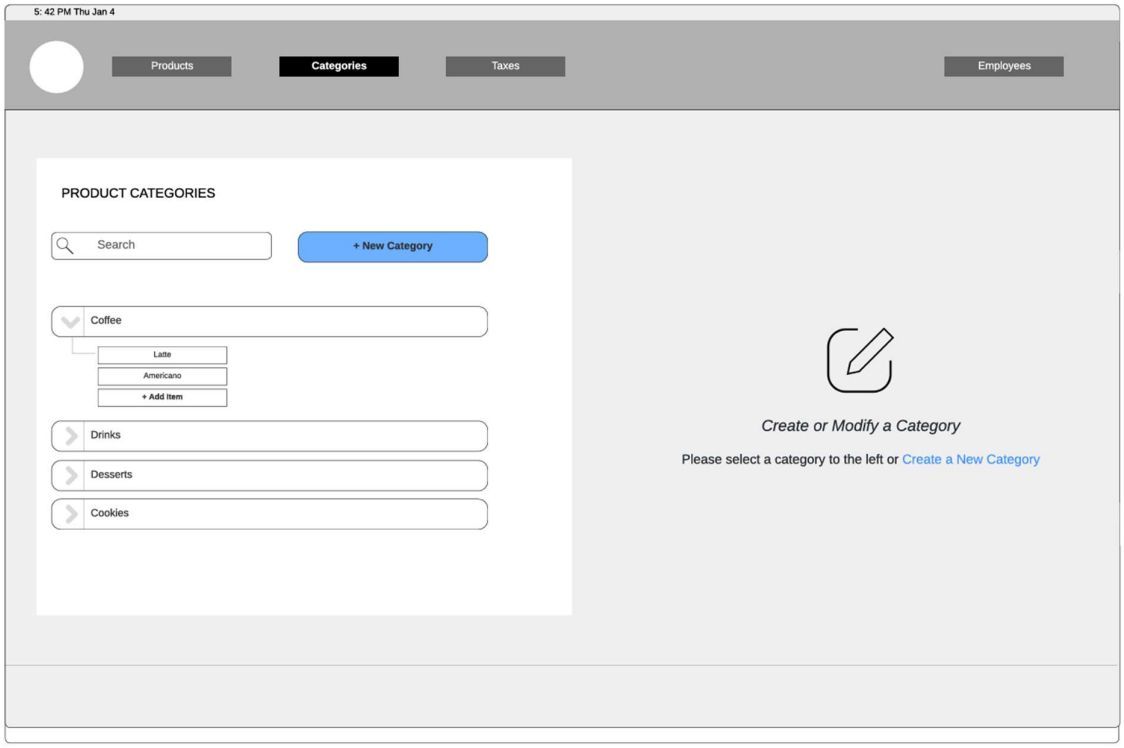

Figure 22: Category management wireframe

4.9 Tax entity

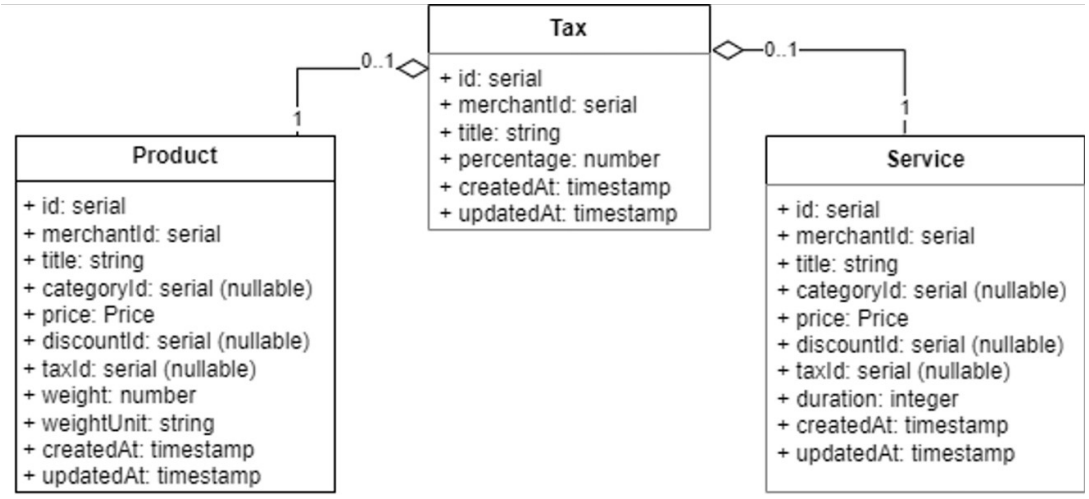

Figure 23: Tax entity

The Tax entity (see Figure 23) in our system defines the different taxes that can be applied to products or services.

Key fields include:

- id - unique identifier for each tax entry
- merchantId - links the tax to a specific merchant, ensuring that each merchant can manage their own tax rules
- title - string representing the name or description of the tax (e.g., VAT)
- percentage - tax rate applied, stored as a percentage. This allows for dynamic tax calculations based on different rates.
- createdAt and updatedAt - timestamps that record when the tax entry was created and last updated, which are useful for tracking and audit purposes.

In the system, a tax can be optionally linked to either a **Product** or a **Service**. Each product or service might be subject to *one* specific tax, but there can also be products or services without any tax applied. Also, a tax can be reused across multiple products and services, making it flexible for merchants who deal with different taxable items.