VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

SOFTWARE ENGINEERING STUDY PROGRAMME

Bar, restaurant and cosmetics PoS system technical specification

Software systems design

Lab work

Authors:

Eglė Orintaitė

Justas Balčiūnas

Vytautas Mielkus

Žygimantas Vidmantas

Gustas Akstinas

Supervisor: Lekt. Vasilij Savin

2024 m.

Contents

1. Intro

This document provides technical specification for a Point of sale (PoS) system, aimed at providing order management, bookings, and system administration functionalities. It covers the essential business operations such as creating and managing orders, services, payments and inventory. The described system also needs to support third-party payment providers such as Stripe and Elavon. Optionally, the system should also be capable of sending SMS notifications and functionality for inventory management

Key objectives:

- Order and payment processing, including support for split payments, refunds, and tipping.
- Service management, allowing employees to create, modify, and cancel reservations.
- System management using user roles, inventory tracking, tax and merchant information.
- Integration with external payment providers for card processing.

The specification includes the data model, API contracts, and system architecture necessary for implementation.

Good luck.

2. Business flows

2.1. Orders & reservations

2.1.1. Create order

The core functionality of the system is to create an order composed of various items selected by the customer. These items can vary depending on the business type. For a restaurant, the items might include food and drinks such as pizza, cola, or pasta dishes. In other business types like a hairdresser, spa, or beauty salon, the items could be services such as massages, haircuts, or other treatments.

The order creation process begins with the customer choosing items or services, which are then presented to the system operator for finalization. The operator selects items from the available menu list, which are then stored as an order in the system's local cache. The system calculates the total price, taking into account any applied discounts or taxes and displays it.

The operator is given the option to finalize the order by either processing the payment immediately (via cash, gift card, or credit card) or marking the order as "Pay Later," which flags it as an "open" order in the database for later payment completion. The flow for this process is detailed in (as shown in Figure 1: Create order BPMN)**.**

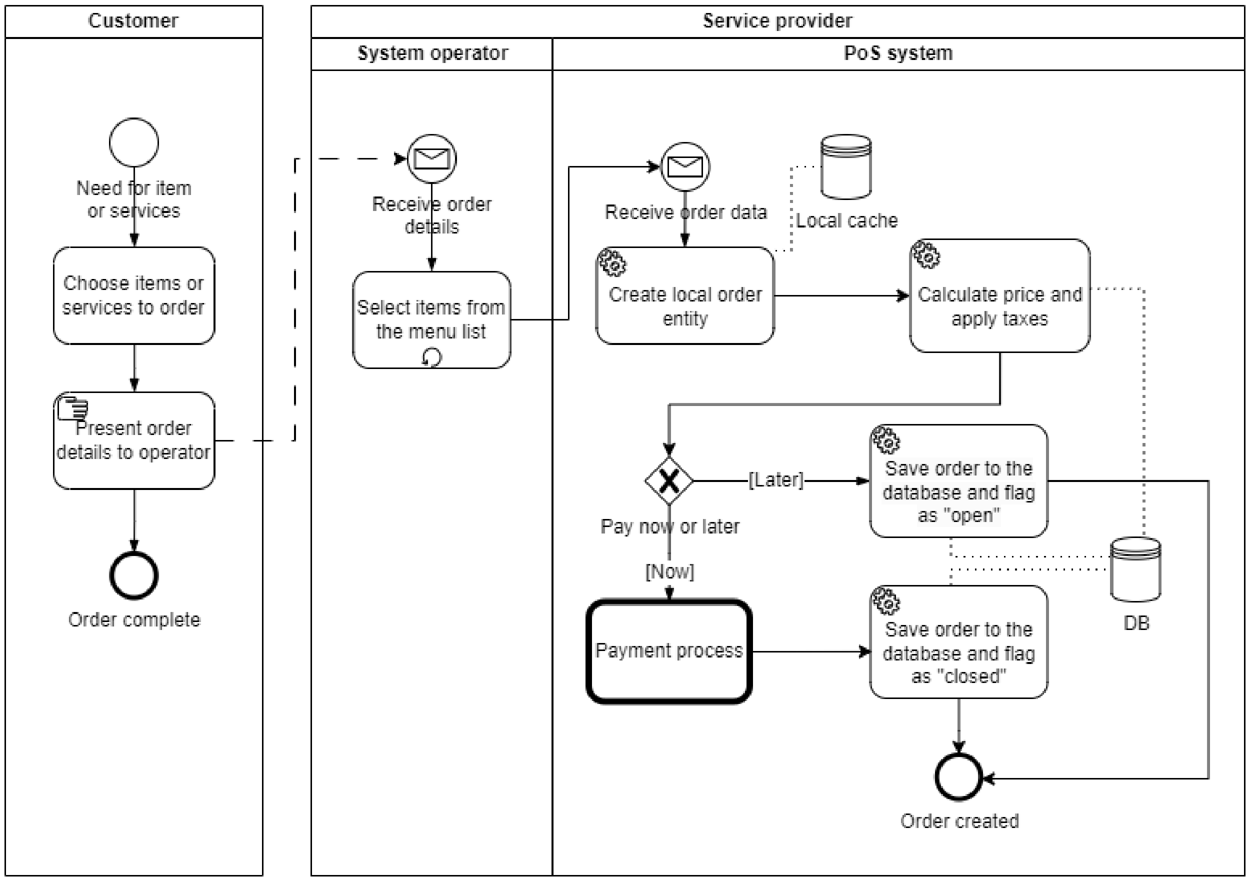

Customer
Service provider
System operator
PoS system
Need for item or services
Choose items or services to order
Present order details to operator
Order complete
Receive order details
Select items from the menu list
Receive order data
Local cache
Create local order entity
Calculate price and apply taxes
Pay now or later
[Later]
[Now]
Save order to the database and flag as "open"
DB
Payment process
Save order to the database and flag as "closed"
Order created

Figure 1: Create order BPMN

Visually, the user interface presents a clear breakdown of the selected items, their respective prices, and the total order amount. Operator has options to edit the order (select items to the order from the menu), split order (split payment check), pay later, or proceed with the payment immediately as seen in (as shown in Figure 2: Main order page wireframe). The user interface is designed to support both quick and detailed order management, allowing the operator to handle customer requests efficiently.

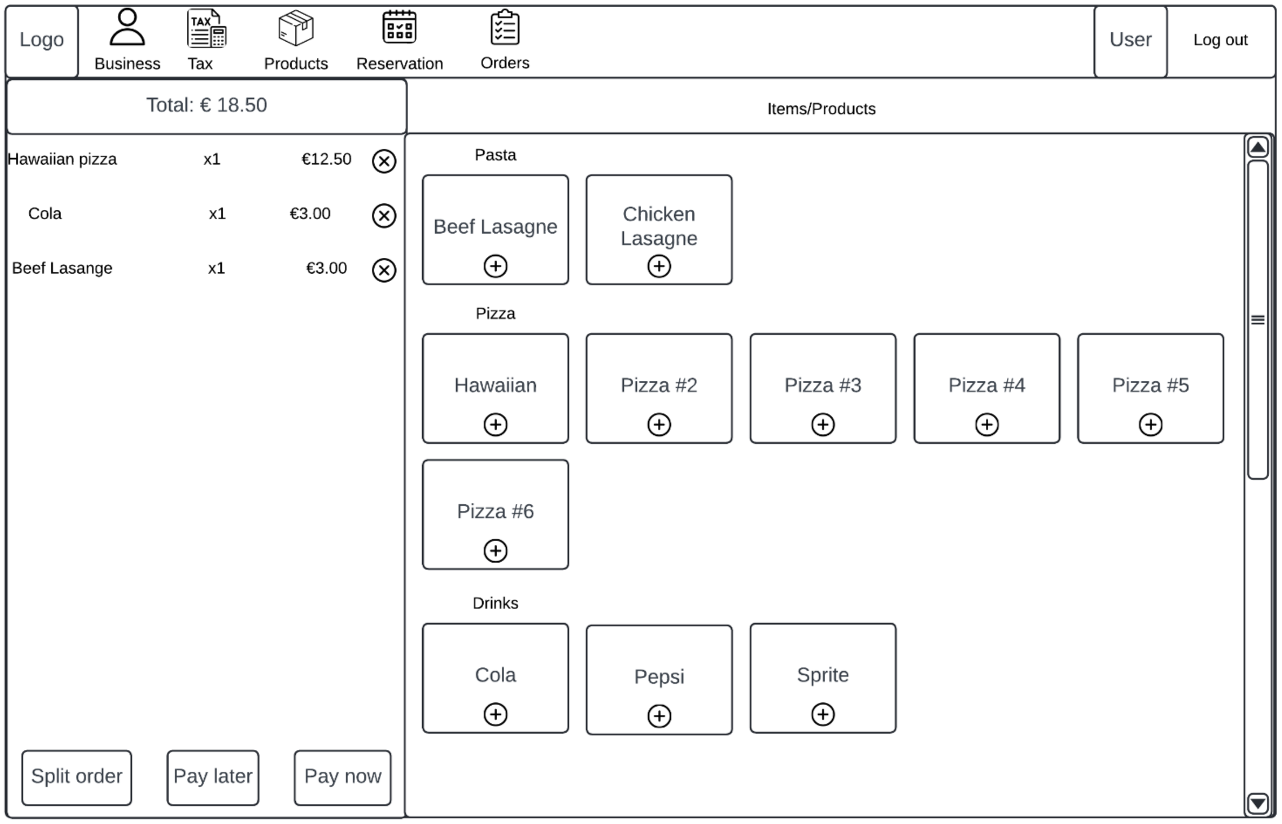


Figure 2: Main order page wireframe

2.1.2. Edit & cancel order

Once an order has been created but not yet paid for, it is saved as an "Open" order in the system. This allows for flexibility if the customer wants to modify their order later, such as adding more items, removing items, or even canceling the order altogether.

The customer might request changes to their original order, such as ordering additional items (e.g., a dessert) or canceling part of the order. The System Operator would navigate to the "**Orders**" section, where a list of all previous orders is displayed. Each order has a label indicating its current status, such as "Open", "Closed", or "Refunded" along with buttons for available actions.

To edit an "Open" order, the operator would locate the relevant order and select the Edit button. This action takes the operator back to the order screen (refer to Figure 2: Main order page wireframe in the Create Order section), where they can make modifications to the order. This includes adding or removing items from the list.

After the changes are made, the updated order is recalculated, and the total price is adjusted accordingly. The system then saves the changes, and the order status remains "Open" until the customer proceeds to payment. If the order is to be fully canceled, it is updated in the system without processing any payment.

This flow is captured in (Figure 3: Edit/Cancel order BPMN), which illustrates the process of editing or canceling an existing order. Additionally, the all "**Orders**" section wireframe in (Figure 4: All orders page wireframe) visually represents how the system displays orders with different statuses, as well as the available actions for each order.

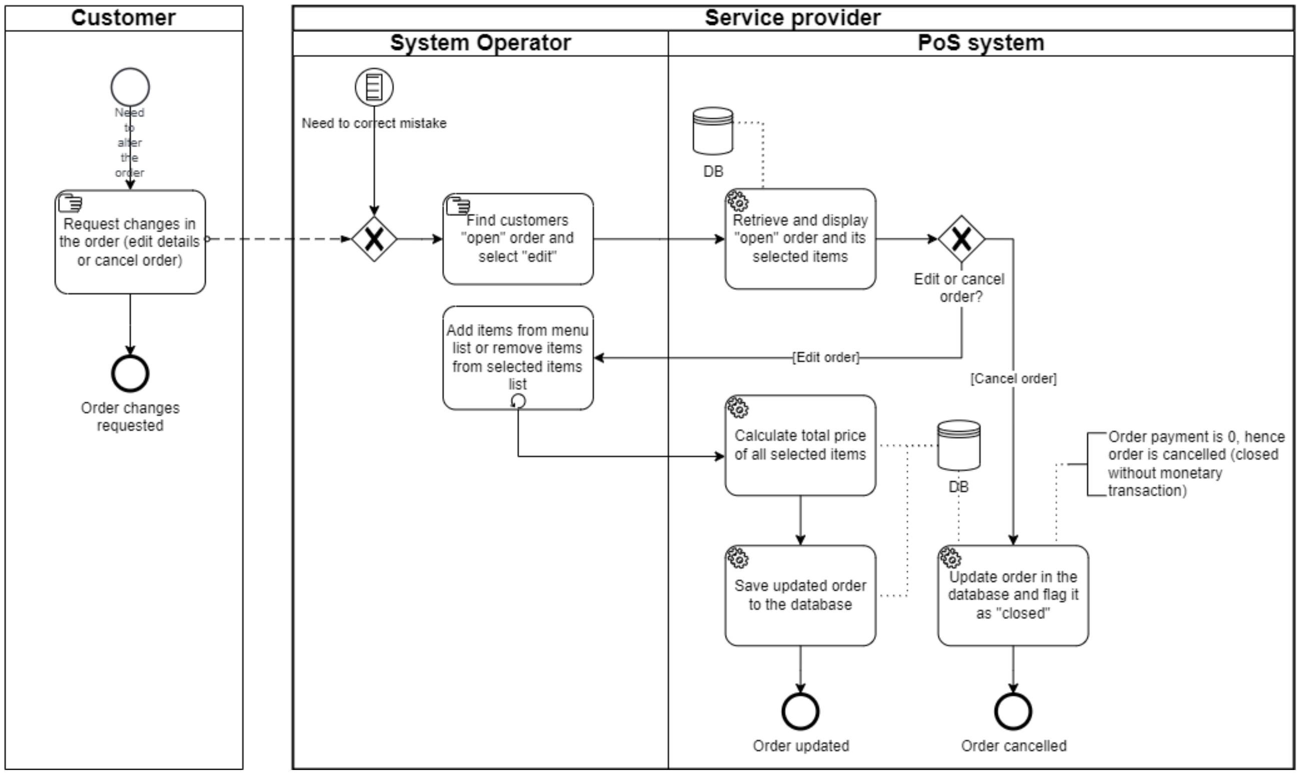
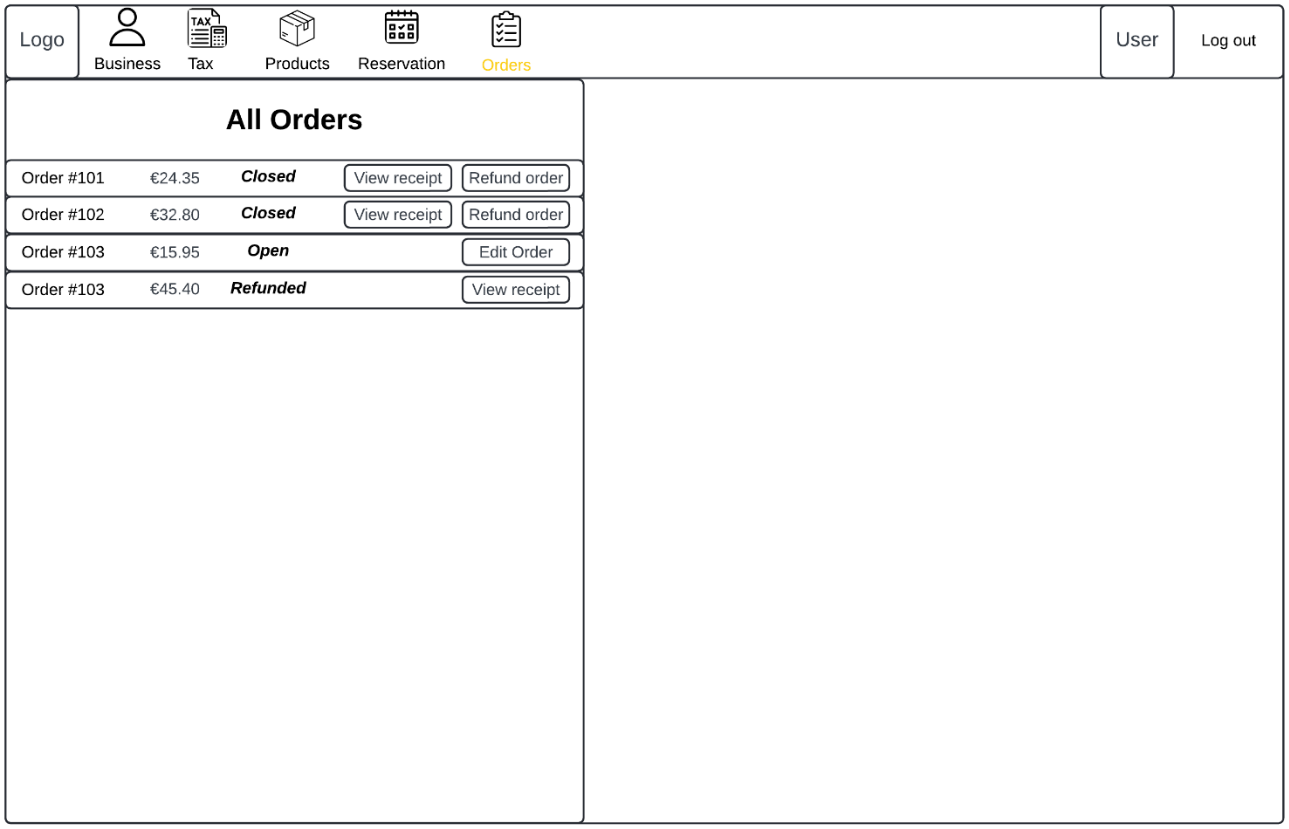

Customer
Service provider
System Operator
PoS system
Need to alter the order
Need to correct mistake
Request changes in the order (edit details or cancel order)
Order changes requested
Find customers "open" order and select "edit"
DB
Retrieve and display "open" order and its selected items
Edit or cancel order?
[Edit order]
[Cancel order]
Add items from menu list or remove items from selected items list
Calculate total price of all selected items
DB
Order payment is 0, hence order is cancelled (closed without monetary transaction)
Save updated order to the database
Update order in the database and flag it as "closed"
Order updated
Order cancelled

Figure 3: Edit/Cancel order BPMN



All Orders

Order #101	€24.35	***Closed***	View receipt	Refund order
Order #102	€32.80	***Closed***	View receipt	Refund order
Order #103	€15.95	***Open***		Edit Order
Order #103	€45.40	***Refunded***		View receipt

Figure 4: All orders page wireframe

2.1.3. Split order

In scenarios where a customer wishes to split the bill into multiple payments, the system provides the ability to split an open order. For example, if a customer has ordered two pizzas and two drinks but wants to pay for one pizza and one drink separately while leaving the remaining items for another payment, the cashier can easily facilitate this using the "**Split Order**" feature.

The cashier starts by navigating to the "**Orders**" section (refer to Figure 4: All orders page wireframe), where a list of all orders is displayed, and selects the relevant **Open Order**. Once the order is found, the cashier presses the **Split Order** button, which brings up a pop-up interface for selecting which items should be moved to the new order. This is depicted in (Figure 5: Split order BPMN), which illustrates the flow for splitting the order. The cashier selects the required items (in this case, one pizza and one drink) for the new split order and presses the "**Confirm**" button.

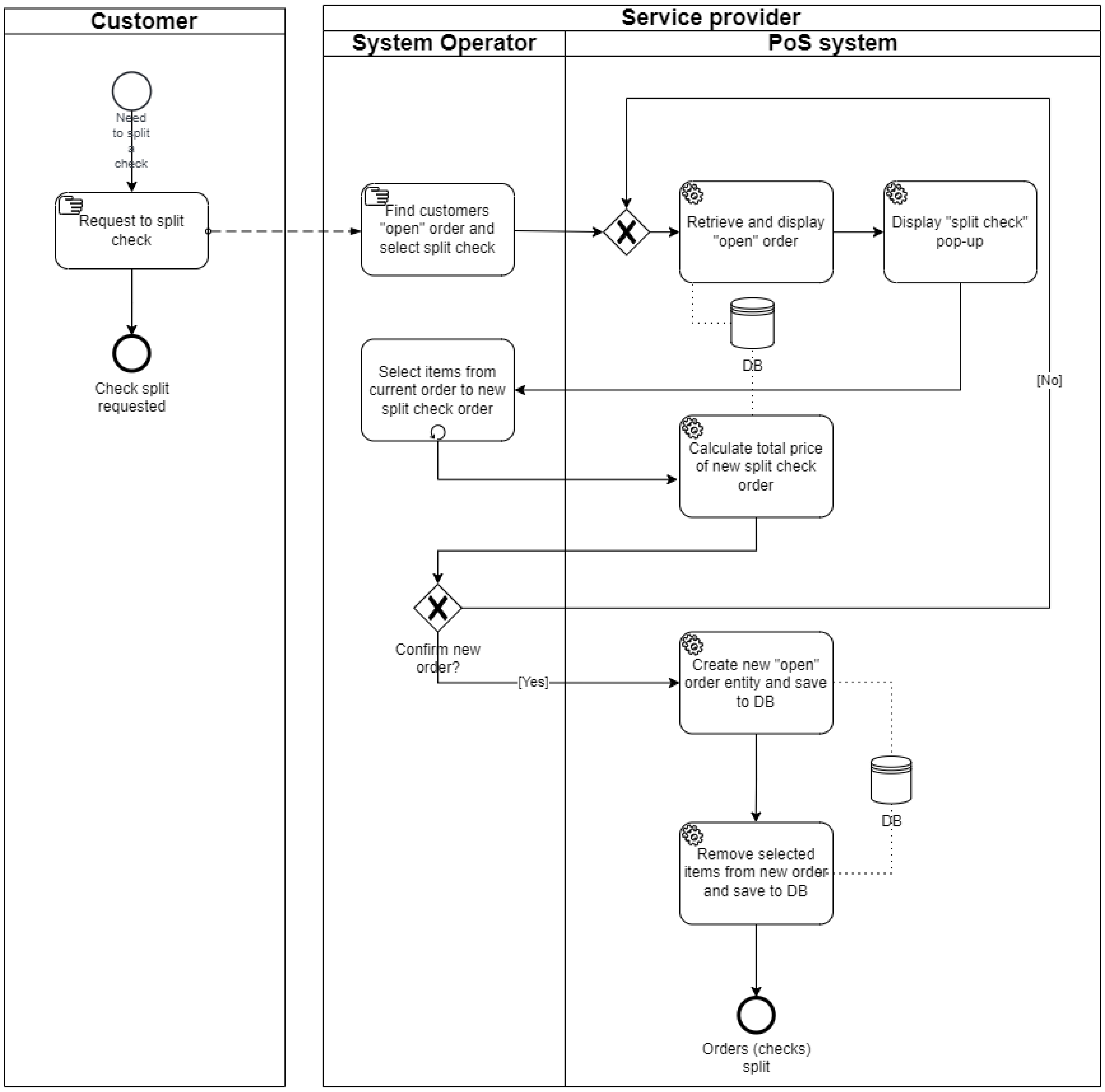

Customer
Service provider
System Operator
PoS system
Need to split a check
Request to split check
Check split requested
Find customers "open" order and select split check
Retrieve and display "open" order
Display "split check" pop-up
DB
Select items from current order to new split check order
Calculate total price of new split check order
[No]
Confirm new order?
[Yes]
Create new "open" order entity and save to DB
DB
Remove selected items from new order and save to DB
Orders (checks) split

Figure 5: Split order BPMN

The system then creates a new "Open" order containing the selected items and automatically updates the original order by removing the split items. This ensures that both orders now reflect the intended separation of items, allowing the customer to proceed with separate payments for each order.

The cashier can now return to the "**Orders**" section and handle each open order individually by proceeding with the payment (e.g., cash, card, or gift card). Both orders can be paid for separately, ensuring flexibility in the payment process.

This flow is captured in (Figure 5: Split order BPMN), which illustrates the split order process from selecting items to creating separate open orders. The corresponding user interface is shown in

(Figure 6: Split order wireframe), where the cashier selects the items for the split and confirms the process.

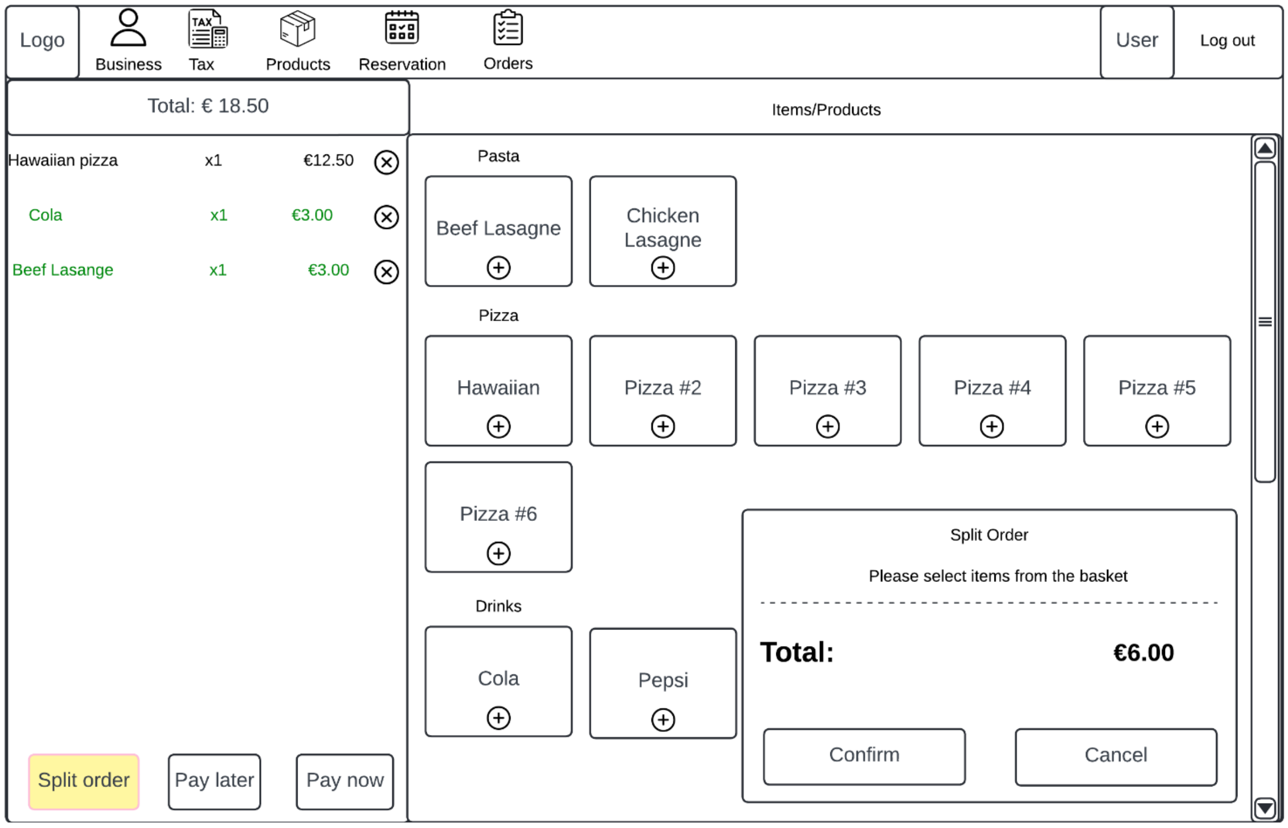


Figure 6: Split order wireframe

2.1.4. Book reservation

The system allows customers to book reservations for services at specific times, such as scheduling a haircut, spa treatment, a table at a restaurant, or other services. This process is designed to handle customer requests by checking available time slots, ensuring that employees are available, and confirming the reservation once all details are validated.

The booking process begins with the customer indicating the need for a service at a specific time. The system operator receives the reservation details and checks the available time slots that match the criteria. If the requested time is available, the reservation is confirmed, and the service is booked. If the requested time is unavailable, the system presents the operator with alternative time slots.

In cases where the alternate times are not suitable to the customer or if the customer requests for the same unavailable time, the booking process is cancelled. However, if the customer agrees to one of the available time slots, the operator confirms the booking and records the reservation in the system.

If the booking for the selection of an employee, which would provide the service (e.g. a hairdresser), the system proceeds to assign the employee to the service booking. In case the customer's chosen service provider is unavailable, a new one is found and attached, and the reservation is completed. The employee details, along with the service information, are stored in the database, ensuring that both the service provider and customer have a confirmed time slot for the requested service. In the case of booking a table in the restaurant employee details are not needed.

This flow is illustrated in (Figure 7: Book reservation BPMN), which depicts the step-by-step process from booking a reservation to confirming the time slot and assigning an employee. Each step ensures that both the system and the customer are aligned in terms of availability and service requirements.

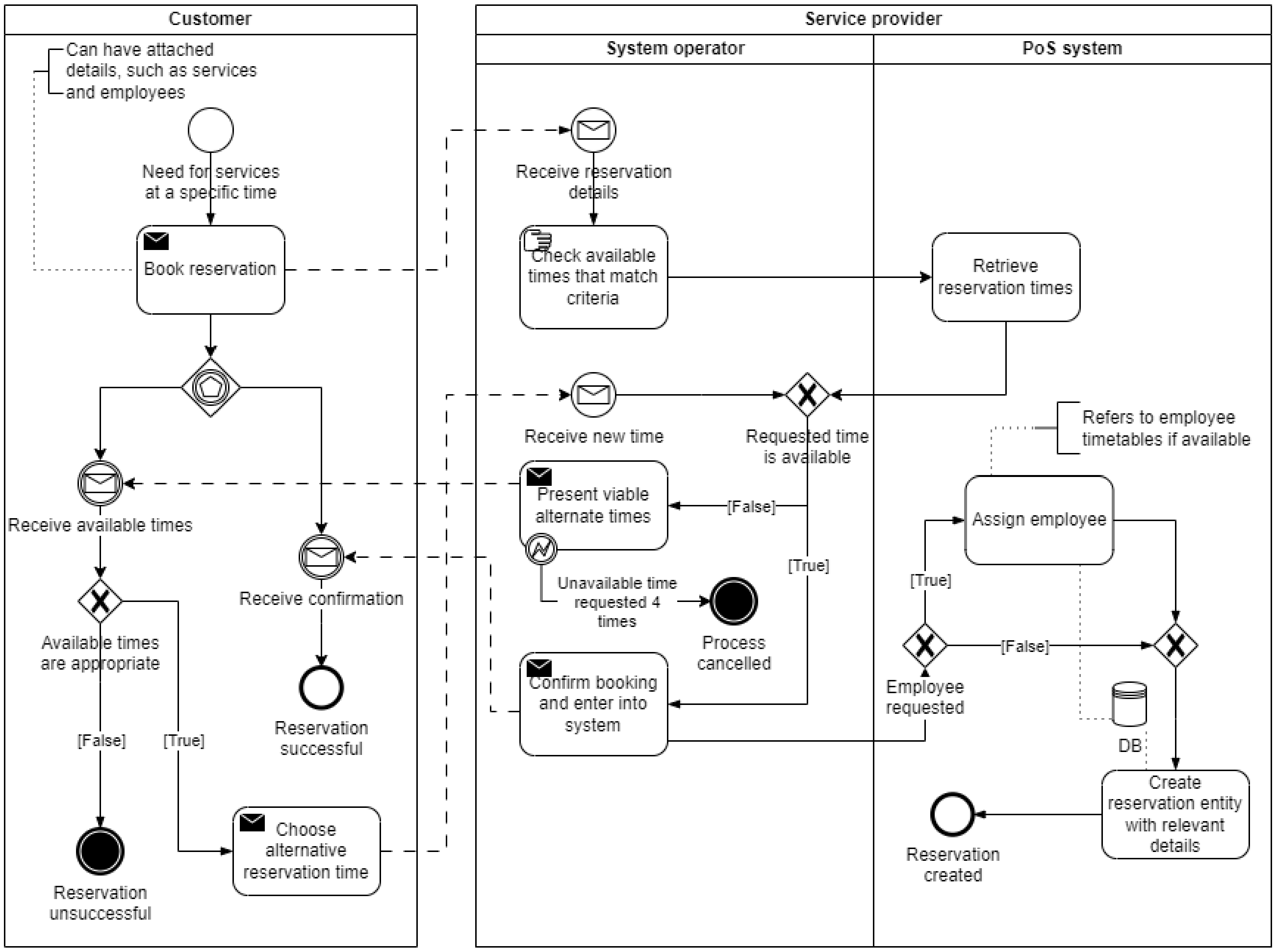

Customer
Service provider
System operator
PoS system
Can have attached details, such as services and employees
Need for services at a specific time
Book reservation
Receive reservation details
Check available times that match criteria
Retrieve reservation times
Receive new time
Requested time is available
Refers to employee timetables if available
Receive available times
Present viable alternate times
[False]
[True]
Assign employee
Receive confirmation
Unavailable time requested 4 times
Process cancelled
Available times are appropriate
Employee requested
[False]
[True]
Confirm booking and enter into system
DB
[False]
[True]
Reservation successful
Create reservation entity with relevant details
Reservation created
Choose alternative reservation time
Reservation unsuccessful

Figure 7: Book reservation BPMN

2.1.5. Modify & cancel reservation

In addition to booking reservations, the system allows customers to modify or cancel existing reservations as needed. This process ensures that customers can make changes or cancel their bookings with ease, while the system operator and PoS system manage the availability of services and employees.

Modification of Reservation:

The customer initiates the process by requesting a change to the existing reservation. The system operator receives the request and pulls up the relevant reservation details from the system. Based on the changes requested by the customer, the system checks whether the requested changes are viable.

If the changes are possible (e.g., requested time and employee availability), the system updates the reservation accordingly and informs the customer that the changes have been successfully made. The updated reservation is saved to the database with the new details, including any modified times or services.

However, if the requested changes cannot be accommodated (e.g., due to unavailable time slots or employees), the system informs the customer and provides an option to adjust their request. If no viable solution is found after multiple attempts, the modification is marked as unsuccessful.

Cancellation of Reservation:

If the customer wishes to cancel a reservation, the operator retrieves the reservation details and chooses the cancellation option. The system then marks the reservation as "closed" in the database, indicating that it has been canceled. No further action is required from the customer, and the system provides confirmation that the reservation has been successfully canceled.

This flow is captured in (Figure 8: Modify/Cancel reservation BPMN), which illustrates both the modification and cancellation processes. The system efficiently handles updates to reservation details, ensuring that both customer satisfaction and resource management are maintained.

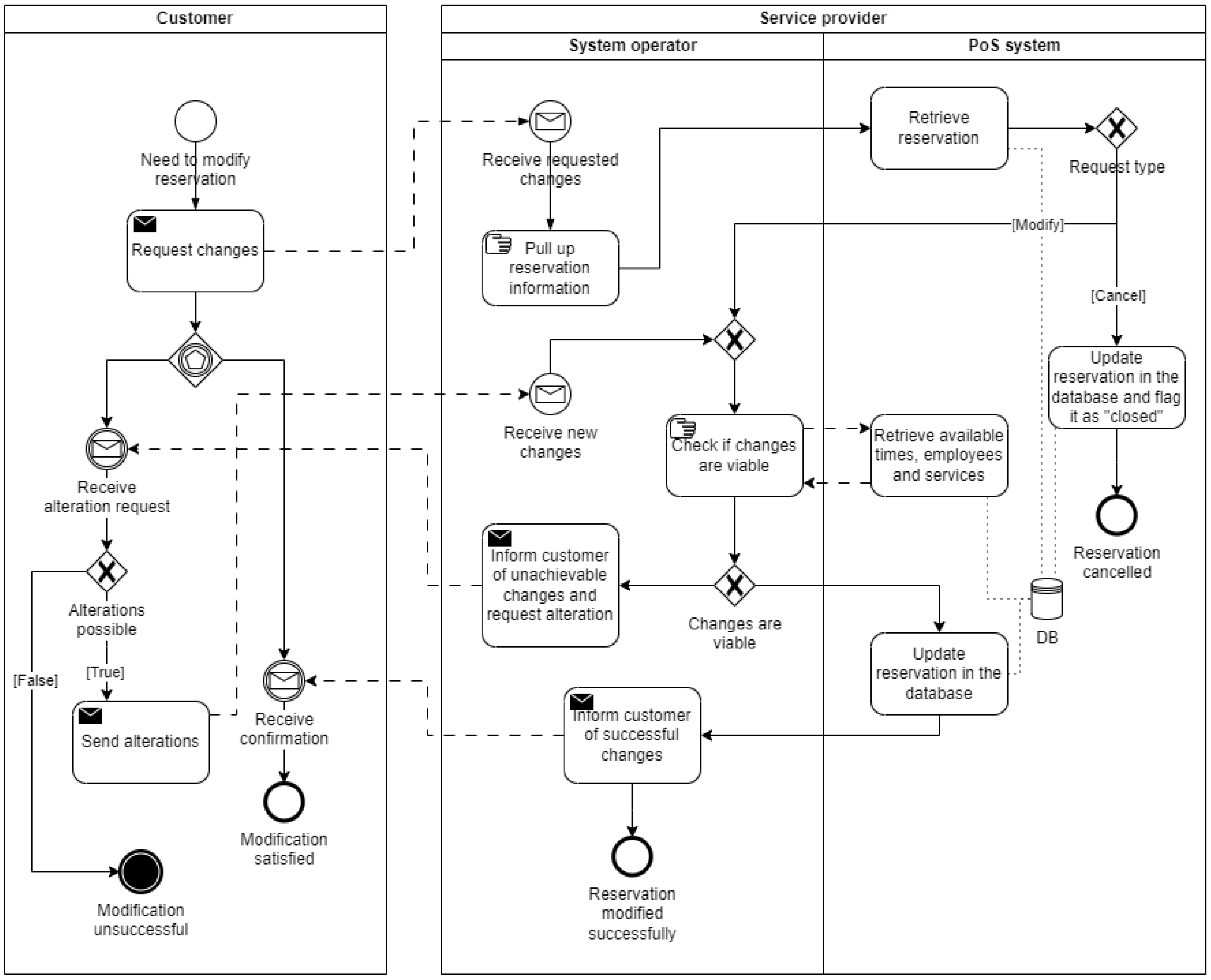

Customer
Service provider
System operator
PoS system
Need to modify reservation
Request changes
Receive alteration request
Alterations possible
[False]
[True]
Send alterations
Receive confirmation
Modification satisfied
Modification unsuccessful
Receive requested changes
Pull up reservation information
Receive new changes
Inform customer of unachievable changes and request alteration
Check if changes are viable
Changes are viable
Inform customer of successful changes
Reservation modified successfully
Retrieve reservation
Request type
[Modify]
[Cancel]
Update reservation in the database and flag it as "closed"
Reservation cancelled
Retrieve available times, employees and services
DB
Update reservation in the database

Figure 8: Modify/Cancel reservation BPMN

2.2. Monetary transactions

2.2.1. Payment

When a customer is ready to pay for their order, the system supports various payment methods, including cash, gift cards, and credit card payments. The payment process allows flexibility in handling discounts and tips, ensuring that the final amount is calculated correctly before proceeding with the transaction.

The payment process starts when the customer requests to pay for their order. The cashier retrieves the open order and clicks on the **Pay** button. At this point, the system presents a pop-up (refer to Figure 9: Payment wireframe) where the cashier can enter any applicable tip amount and apply a discount. These options are visually displayed, allowing for quick adjustments to the total price. Once the final amount is calculated, the cashier proceeds by selecting one of the payment methods: **Cash**, **Gift Card**, or **Credit Card**.

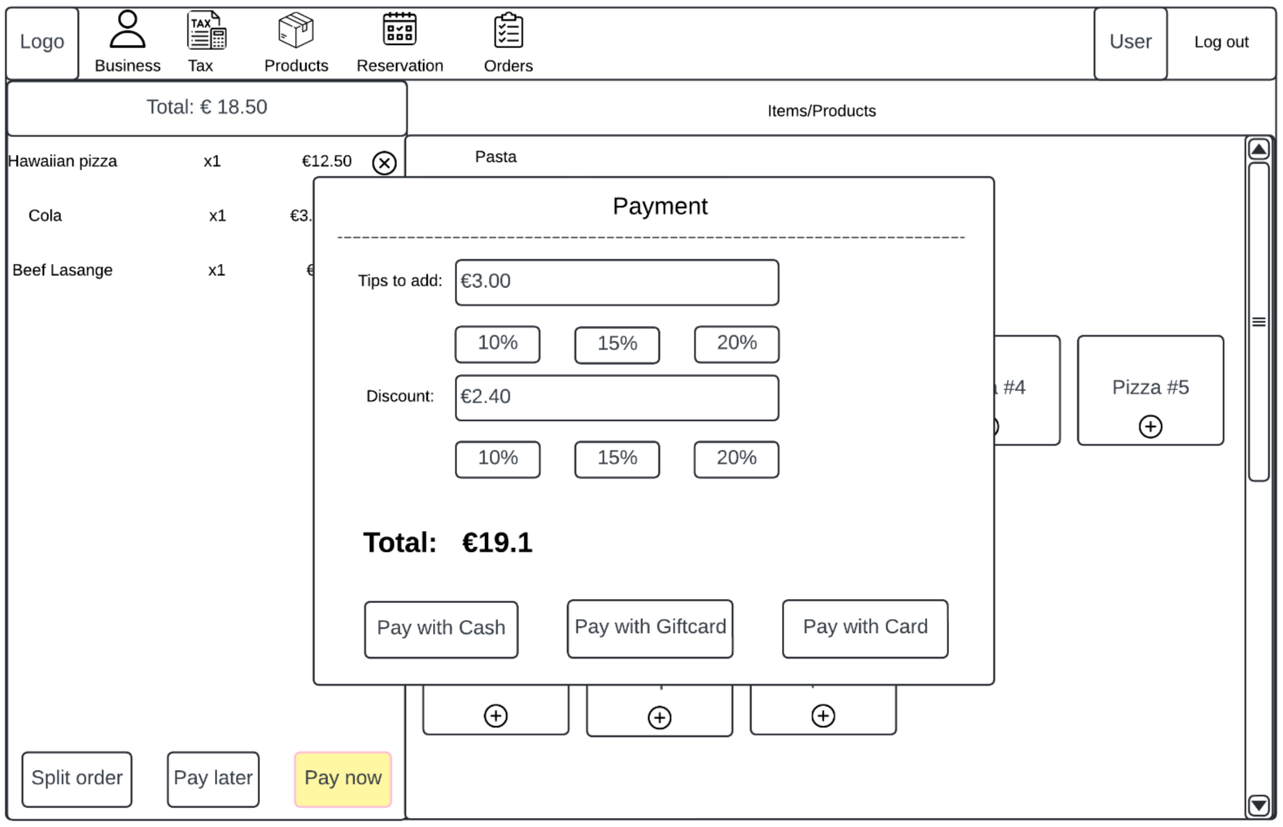


Figure 9: Payment wireframe

Payment by Cash:

If the customer pays with cash, the cashier simply selects the **Pay with Cash** option (see Figure 9: Payment wireframe). The system processes the payment, updates the order status to "Closed," and stores the completed transaction in the database.

Payment by Gift Card:

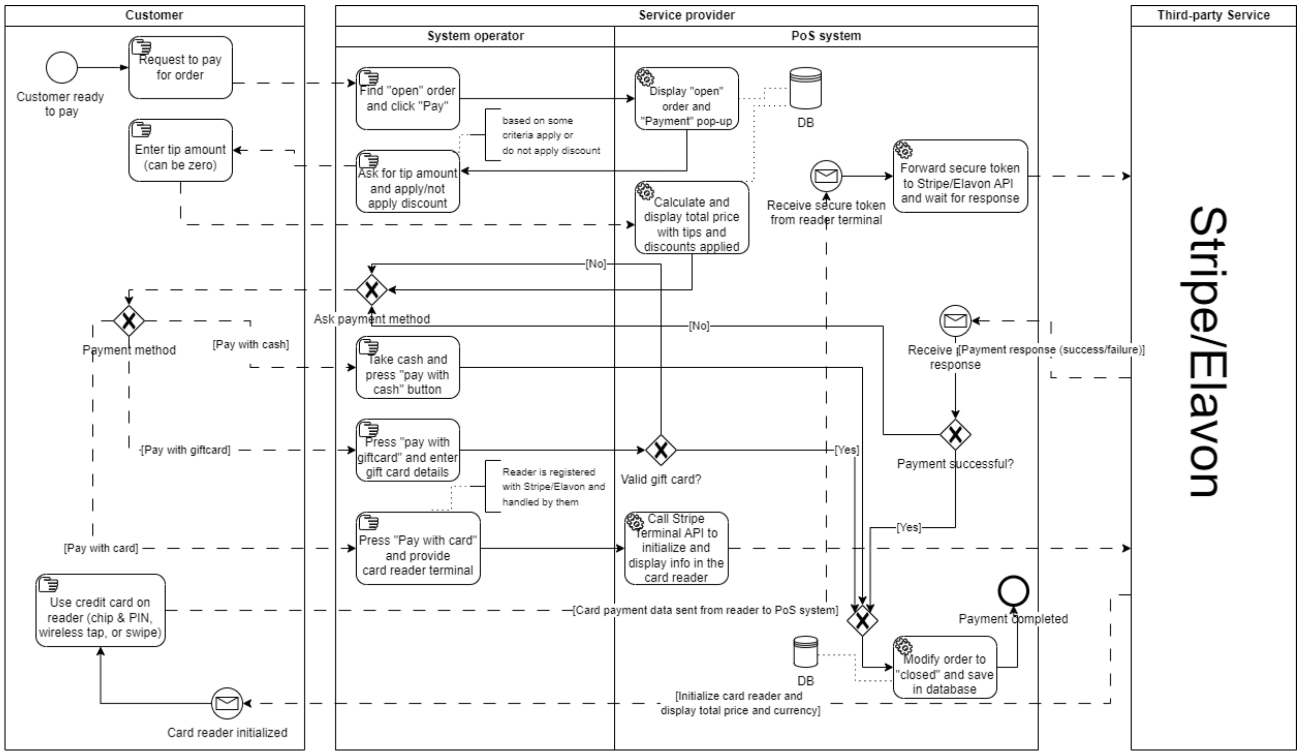
For gift card payments, the cashier selects **Pay with Gift Card** and enters the gift card details. The system verifies the card's validity through an internal process (or possibly an external API if integrated), and if successful, the order is closed.

Payment by Credit Card:

The Credit **Card** payment process involves more steps and includes integration with a third-party service like Stripe or Elavon. When the cashier selects **Pay with Card**, the system initializes the card reader terminal, prompting the customer to use their credit card (insert, tap, or swipe). The card reader captures the necessary payment data and securely sends a token to the PoS system.

The PoS system forwards this token to the third-party service (Stripe or Elavon) for processing. The payment provider handles the transaction and returns a response indicating whether the payment was successful. If the payment is approved, the system updates the order status to "Closed," and the transaction is stored in the database (as shown in Figure 9: Payment wireframe). If the payment fails, the system returns an error, and the cashier can retry or select another payment method.

This flow is illustrated in (Figure 10: Order payment BPMN diagram), showing the various payment options and the interaction with third-party services for credit card payments.



Customer
Service provider
System operator
PoS system
Third-party Service
Customer ready to pay
Request to pay for order
Enter tip amount (can be zero)
Find "open" order and click "Pay"
Ask for tip amount and apply/not apply discount
based on some criteria apply or do not apply discount
Display "open" order and "Payment" pop-up
DB
Calculate and display total price with tips and discounts applied
Receive secure token from reader terminal
Forward secure token to Stripe/Elavon API and wait for response
Stripe/Elavon
[No]
Ask payment method
Payment method
[Pay with cash]
Take cash and press "pay with cash" button
[Pay with giftcard]
Press "pay with giftcard" and enter gift card details
Valid gift card?
[Yes]
Receive response
[Payment response (success/failure)]
Payment successful?
Reader is registered with Stripe/Elavon and handled by them
[Pay with card]
Press "Pay with card" and provide card reader terminal
Call Stripe Terminal API to initialize and display info in the card reader
Use credit card on reader (chip & PIN, wireless tap, or swipe)
[Card payment data sent from reader to PoS system]
Payment completed
Modify order to "closed" and save in database
[Initialize card reader and display total price and currency]
Card reader initialized

Figure 10: Order payment BPMN diagram

2.2.2. Refund

The system allows customers to request a refund for completed orders, ensuring that businesses can efficiently manage refund requests. Refunds can only be applied to **Closed Orders**, which are marked as fully paid. The cashier manages the refund process by selecting the appropriate order and processing the refund request.

When a customer requests a refund, the system operator navigates to the “**Orders**” section (refer to Figure 4: All orders page wireframe), where a list of all previous orders is displayed. Orders that are eligible for a refund are labeled as "Closed" and have a **Refund Order** button next to them. The cashier selects the relevant order and initiates the refund process.

Upon clicking the refund button, the system displays a pop-up where the cashier is prompted to enter the reason for the refund (refer to Figure 11: Refund BPMN diagram). Once the refund reason is entered and confirmed, the system saves the details in the database and updates the order's status to "Refunded".

Finally, the cashier returns the cash to the customer, completing the refund process. The order is marked as "Refunded" in the system, and its details are stored in the database for record-keeping. The refunded order is now visible in the "**Orders**" section with the updated status, ensuring that the refund is properly tracked and managed.

This flow is illustrated in (Figure 11: Refund BPMN diagram), which depicts the step-by-step process from selecting a closed order to confirming the refund and returning cash to the customer.

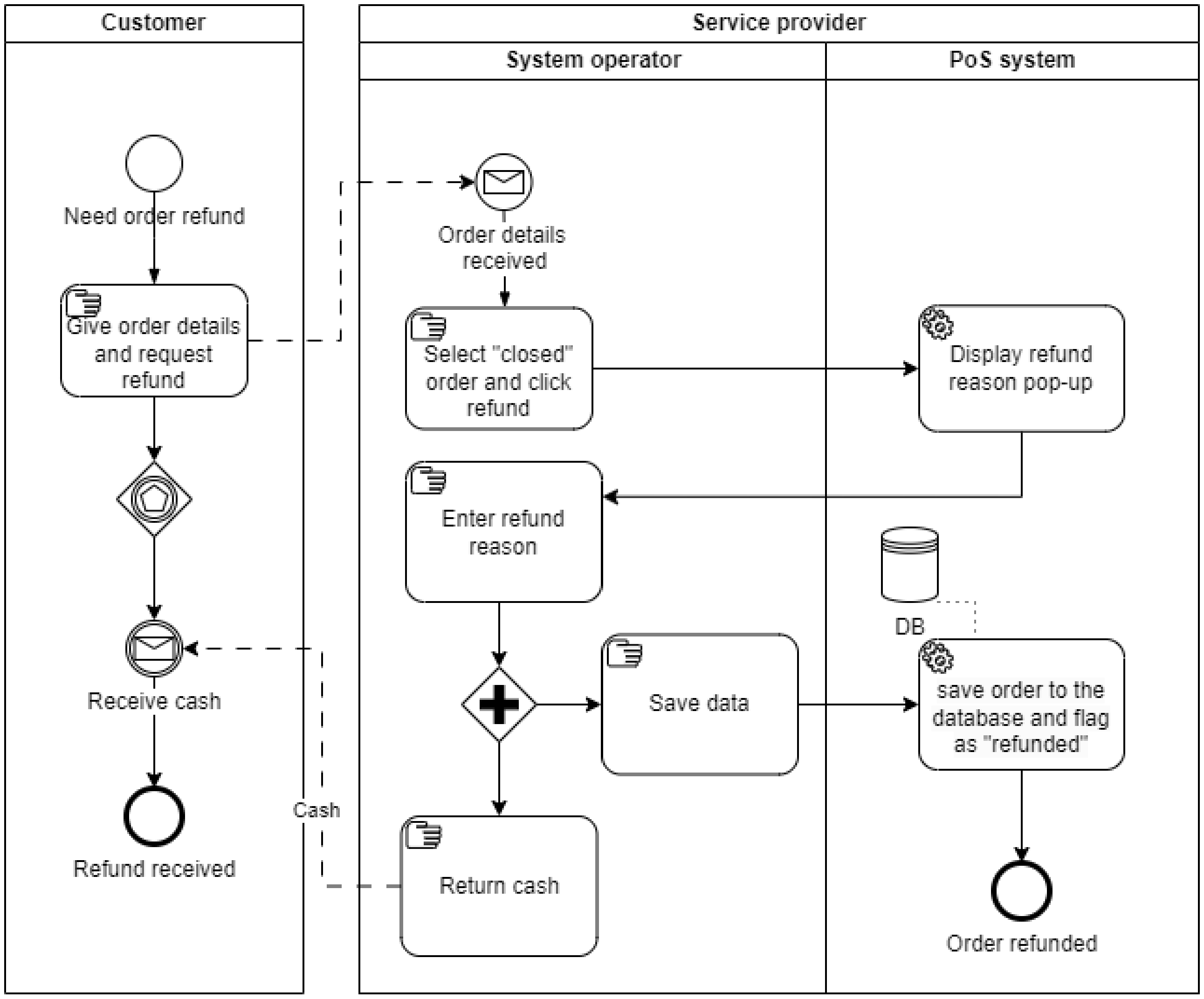


Figure 11: Refund BPMN diagram

2.3. Management

As the system is intended to implement the business processes of both catering and beauty sectors, it is crucial to design it to allow for customizability required for different businesses. This means putting management systems in place meant to control the products or services that are being sold, along with their prices and tax percentages as well as employees, their credentials and system log-in information. It is important to note that all operations in relation to system management is only available to authenticated users, whether it be the business owner, certified employees or a “SuperAdmin”.

2.3.1. Product management

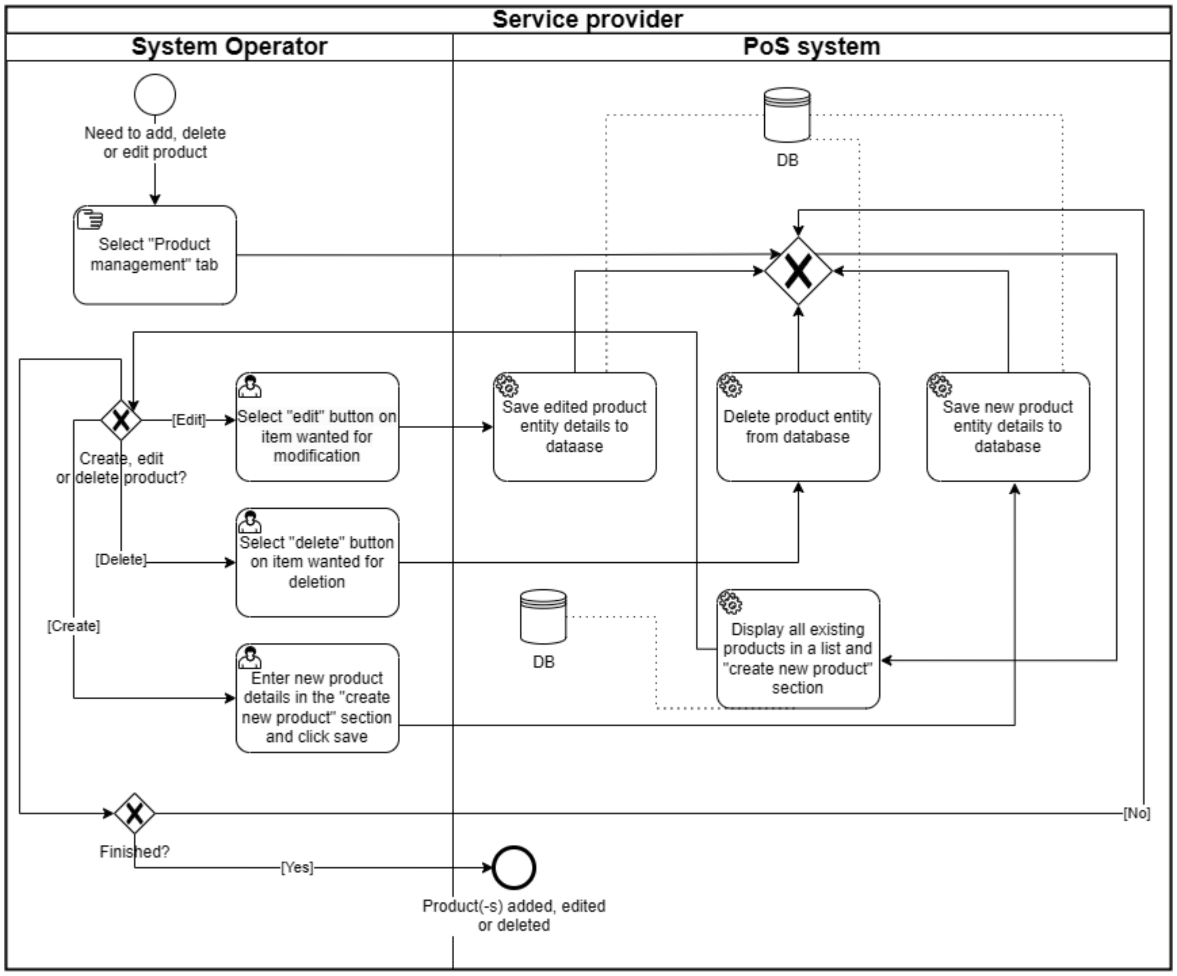
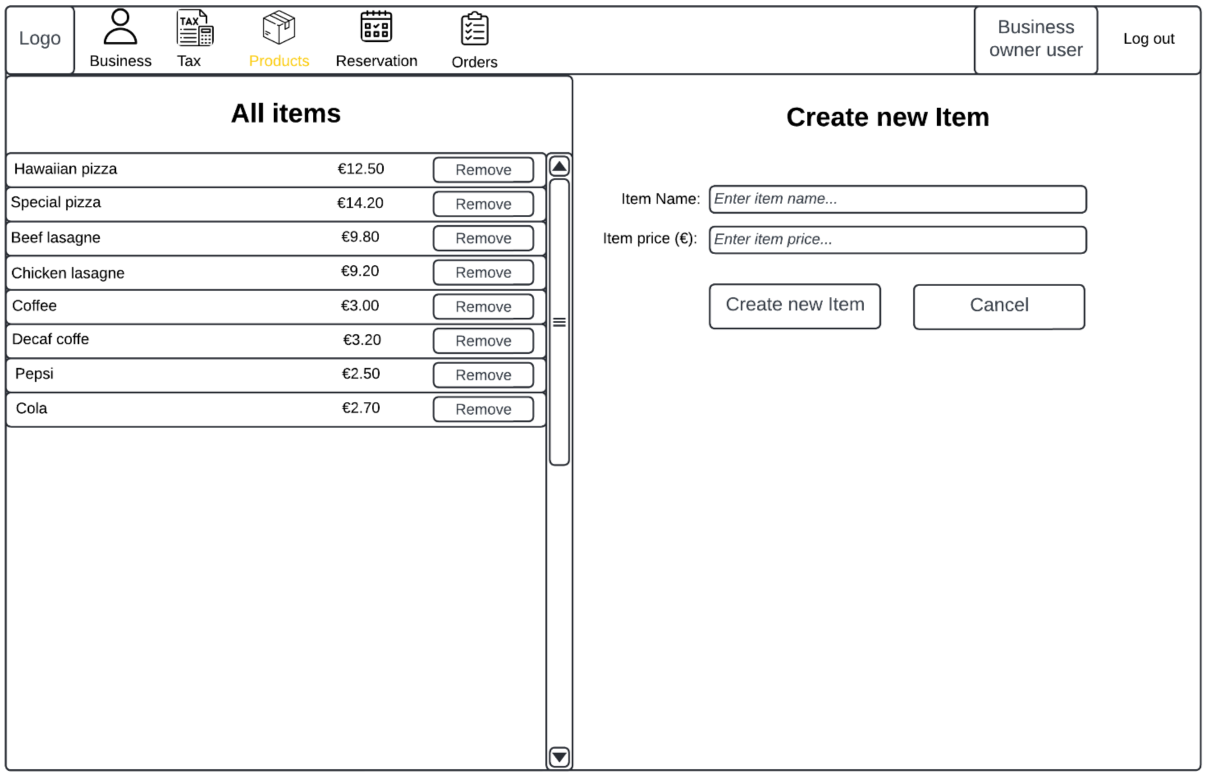

Service provider
System Operator
PoS system
Need to add, delete or edit product
Select "Product management" tab
DB
Create, edit or delete product?
[Edit]
[Delete]
[Create]
Select "edit" button on item wanted for modification
Select "delete" button on item wanted for deletion
Enter new product details in the "create new product" section and click save
Save edited product entity details to dataase
Delete product entity from database
Save new product entity details to database
Display all existing products in a list and "create new product" section
DB
Finished?
[No]
[Yes]
Product(-s) added, edited or deleted

Figure 12. Product management BPMN diagram

Businesses require functionalities that allow system operators (employees) to manage their product or service catalogue. If the business runs a hairdresser saloon, they might want the services they offer from the system’s perspective to be, for example, a beard trim, a haircut, hair styling, hair dying, etc., while restaurants would want to have their product list be the dishes that the kitchen prepares. The process of adding, editing or removing products/services is handled by the system operator, who accesses the "Products" tab to start managing the products. Once in the product management section, the operator is presented with options to create a new product, edit an existing product by selecting one from the list, or delete a product (refer to Figure 13. Wireframe of the product management page, Figure 12. Product management BPMN diagram).

If the operator chooses to edit a product, they select the desired product for modification, make the necessary changes, and then save the edited details to the database. In the case of deletion, the operator selects the product to be removed, and the system proceeds to delete the product entity from the database. For creating a new product, the operator fills in the required details in the "create new product" section and saves this information, which is then stored in the database.

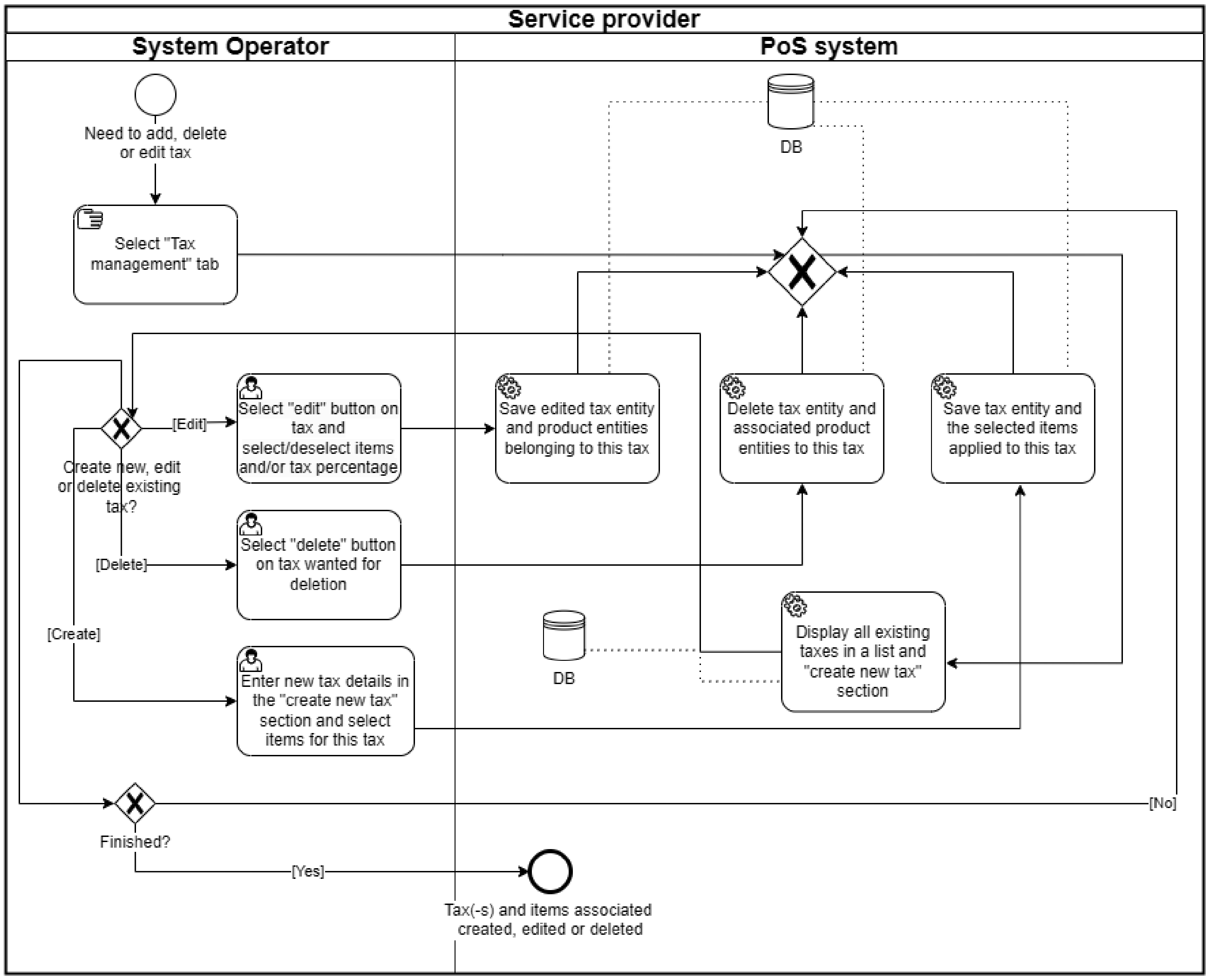
The process concludes with a decision point where the operator can choose to continue making changes or finish the task. If the operator decides to finish, the product(s) are successfully added, edited, or deleted, and the operation is marked as complete. The flow is depicted in (Figure 12. Product management BPMN diagram), where all actions are effectively divided into their respective roles, showing how they interact with the database to manage product entities. This structured approach ensures that all product modifications are properly handled, stored, and updated in the system.



Hawaiian pizza	€12.50	Remove
Special pizza	€14.20	Remove
Beef lasagne	€9.80	Remove
Chicken lasagne	€9.20	Remove
Coffee	€3.00	Remove
Decaf coffe	€3.20	Remove
Pepsi	€2.50	Remove
Cola	€2.70	Remove

Figure 13. Wireframe of the product management page

2.3.2. Tax management

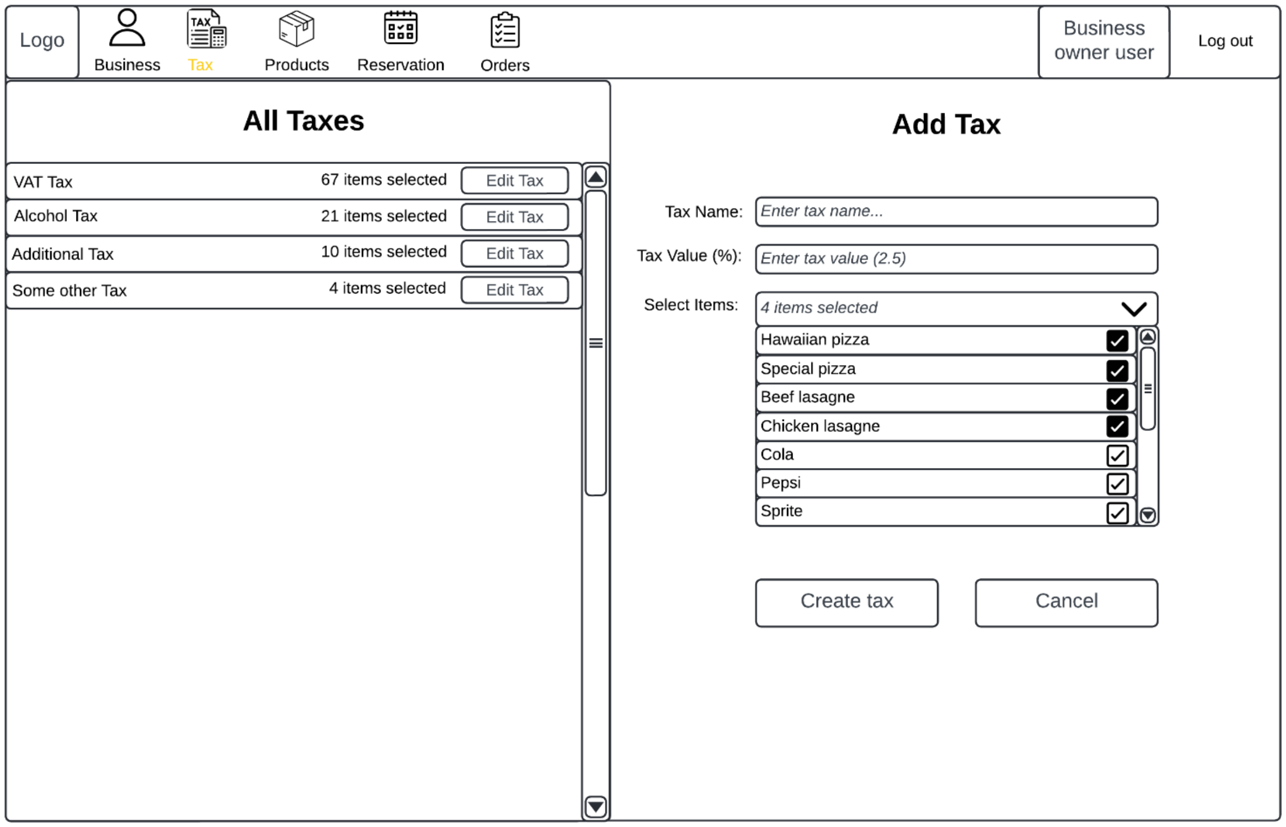


Service provider
System Operator
PoS system
Need to add, delete or edit tax
Select "Tax management" tab
DB
Create new, edit or delete existing tax?
[Edit]
[Delete]
[Create]
Select "edit" button on tax and select/deselect items and/or tax percentage
Select "delete" button on tax wanted for deletion
Enter new tax details in the "create new tax" section and select items for this tax
Save edited tax entity and product entities belonging to this tax
Delete tax entity and associated product entities to this tax
Save tax entity and the selected items applied to this tax
Display all existing taxes in a list and "create new tax" section
DB
Finished?
[No]
[Yes]
Tax(-s) and items associated created, edited or deleted

Figure 14. Tax management BPMN diagram

Businesses selling products or services are usually taxpayers and need a way to handle taxes accurately. The system's workflow is designed to enable operators to add, edit, or delete tax details associated with various items in the system. For example, a restaurant may need to adjust the tax rates for different products like food and alcoholic beverages, where each category might have a distinct tax percentage. Similarly, a salon might need to create a new tax entity for special services that are exempt from the regular sales tax. This process ensures that any changes in tax rates are consistently applied across relevant products or services in the system.

The management process begins by selecting the "Tax" tab (refer to Figure 15. Wireframe of the Tax management tab). If the operator chooses to edit a tax, they select the tax entity, adjust the items or tax percentages, and save these changes to the database. For deleting a tax, the operator selects the specific tax, and the system deletes the tax entity and its associations with products. When creating a new tax, the operator enters the tax details, associates it with relevant items, and saves this information. The flow is depicted in (refer to Figure 14. Tax management BPMN diagram), illustrating how these tasks integrate with the database to maintain up-to-date tax configurations.



VAT Tax	67 items selected	Edit Tax
Alcohol Tax	21 items selected	Edit Tax
Additional Tax	10 items selected	Edit Tax
Some other Tax	4 items selected	Edit Tax

Tax Name: *Enter tax name...*

Tax Value (%): *Enter tax value (2.5)*

Select Items: *4 items selected*

Hawaiian pizza	✓
Special pizza	✓
Beef lasagne	✓
Chicken lasagne	✓
Cola	☐
Pepsi	☐
Sprite	☐

Figure 15. Wireframe of the Tax management tab

2.3.3. User management

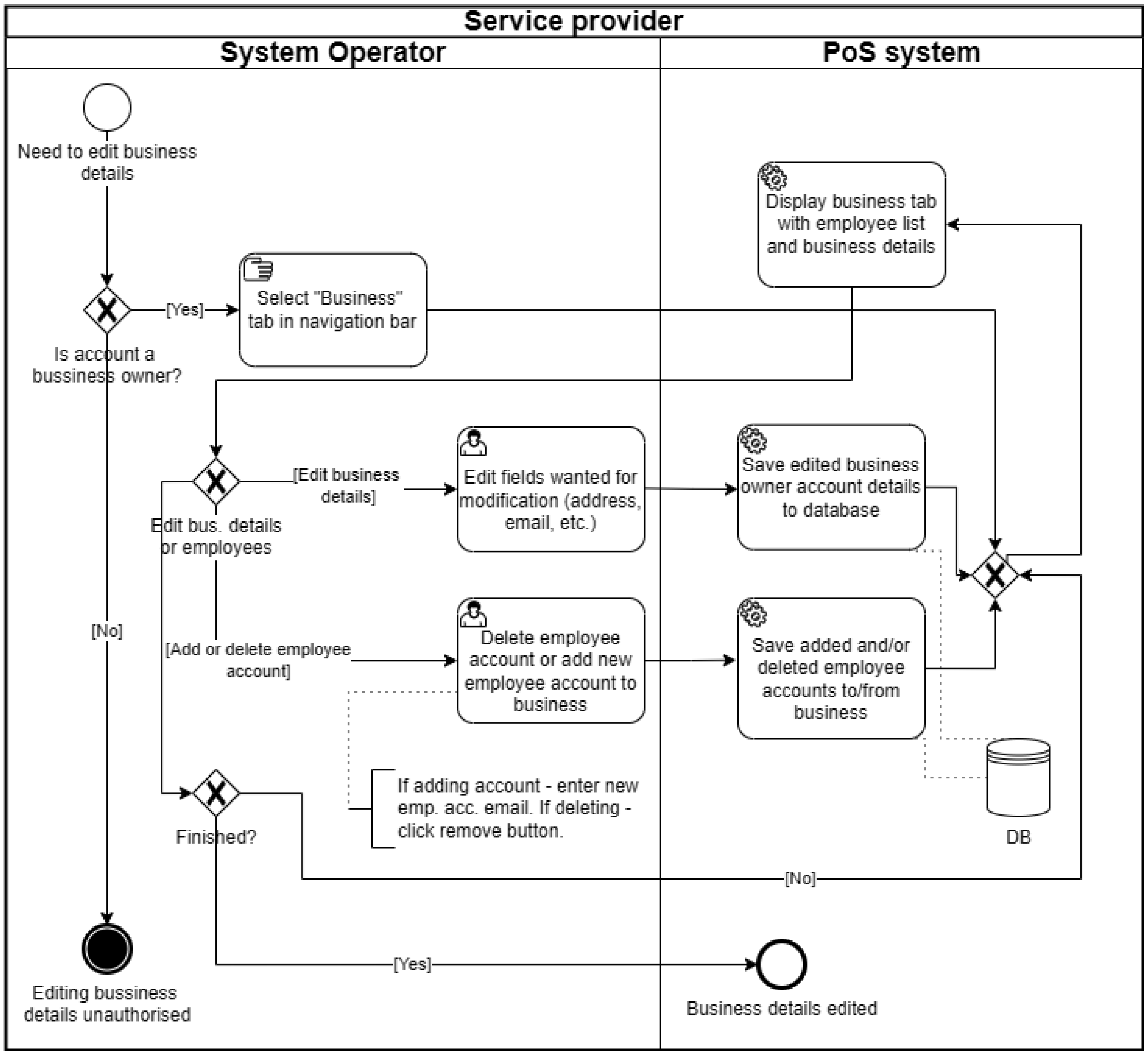
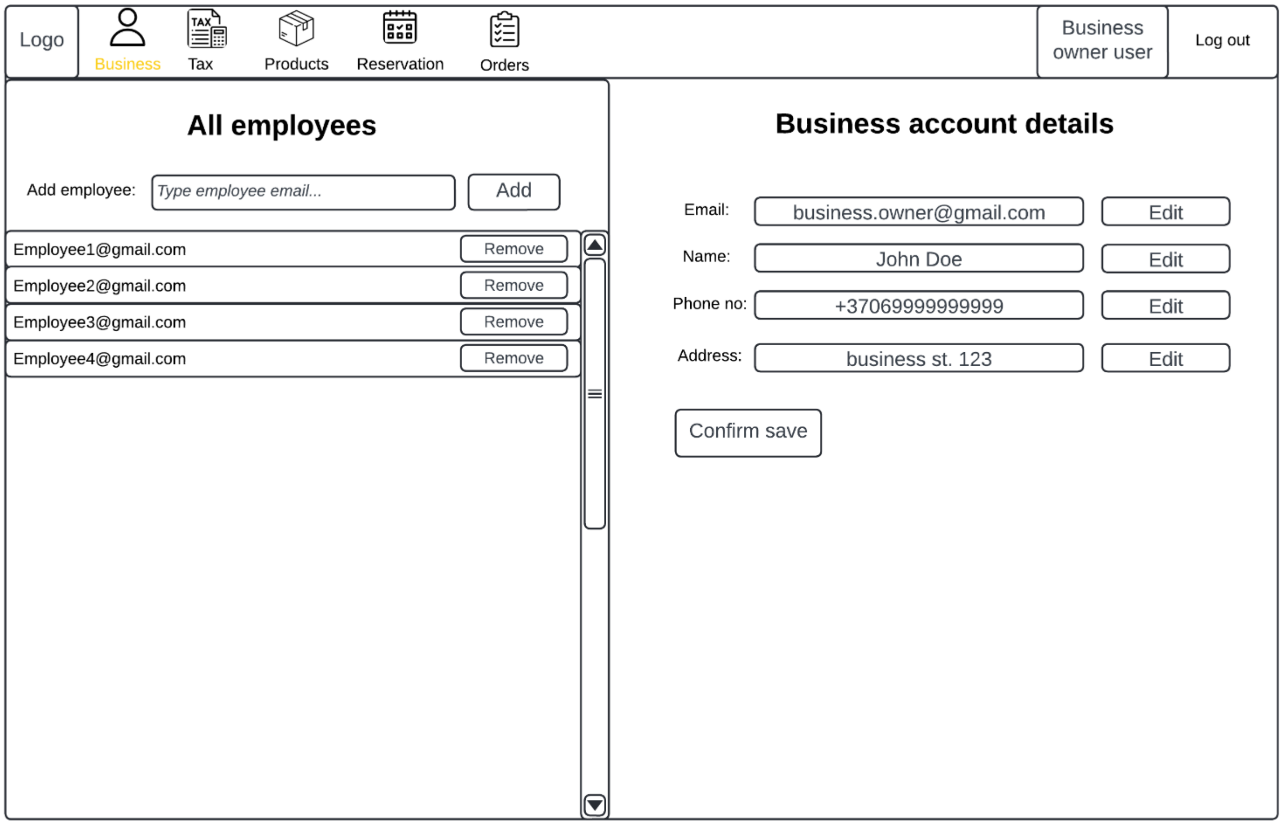

Service provider
System Operator
PoS system
Need to edit business details
Is account a bussiness owner?
[Yes]
Select "Business" tab in navigation bar
Display business tab with employee list and business details
Edit bus. details or employees
[Edit business details]
Edit fields wanted for modification (address, email, etc.)
Save edited business owner account details to database
[No]
[Add or delete employee account]
Delete employee account or add new employee account to business
Save added and/or deleted employee accounts to/from business
If adding account - enter new emp. acc. email. If deleting - click remove button.
DB
Finished?
[No]
[Yes]
Editing bussiness details unauthorised
Business details edited

Figure 16. User management BPMN diagram

This process enables the management of business details and employee accounts, which is essential for maintaining up-to-date information within a company's operations. For example, a restaurant owner might need to update the business address if they open a new location or move to a different part of the city. Similarly, if a barbershop hires a new stylist, the system operator can add the new employee's details, ensuring they have access to the system to manage appointments and orders. On the other hand, when an employee leaves the business, the operator can easily remove their account to prevent unauthorized access.

The process begins by checking if the logged-in user has business owner permissions. If the user is authorized, they access the "Business" tab (refer to Figure 17. Wireframe of the Business tab) to make changes. The operator can then choose to either edit the business details—like updating the contact email or address—or manage employee accounts by adding a new team member or removing an existing one. All these changes are then saved to the database, ensuring that the system remains secure and the data is up-to-date. The flow is depicted in (refer to Figure 16. User management BPMN diagram), which illustrates the steps and decision points involved in this business and employee management process.



Employee1@gmail.com
Employee2@gmail.com
Employee3@gmail.com
Employee4@gmail.com

Email: business.owner@gmail.com
Name: John Doe
Phone no: +37069999999999
Address: business st. 123

Figure 17. Wireframe of the Business tab

3. System hight level architecture

Following high level package diagram (Fig. n) represents systems architecture and main elements. Some elements are expected but not necessarily, therefore they are not displayed here. One of them being a secondary database for archived order data. Saving historical order data is very expensive so it is advised to eventually integrate another database to store that data.

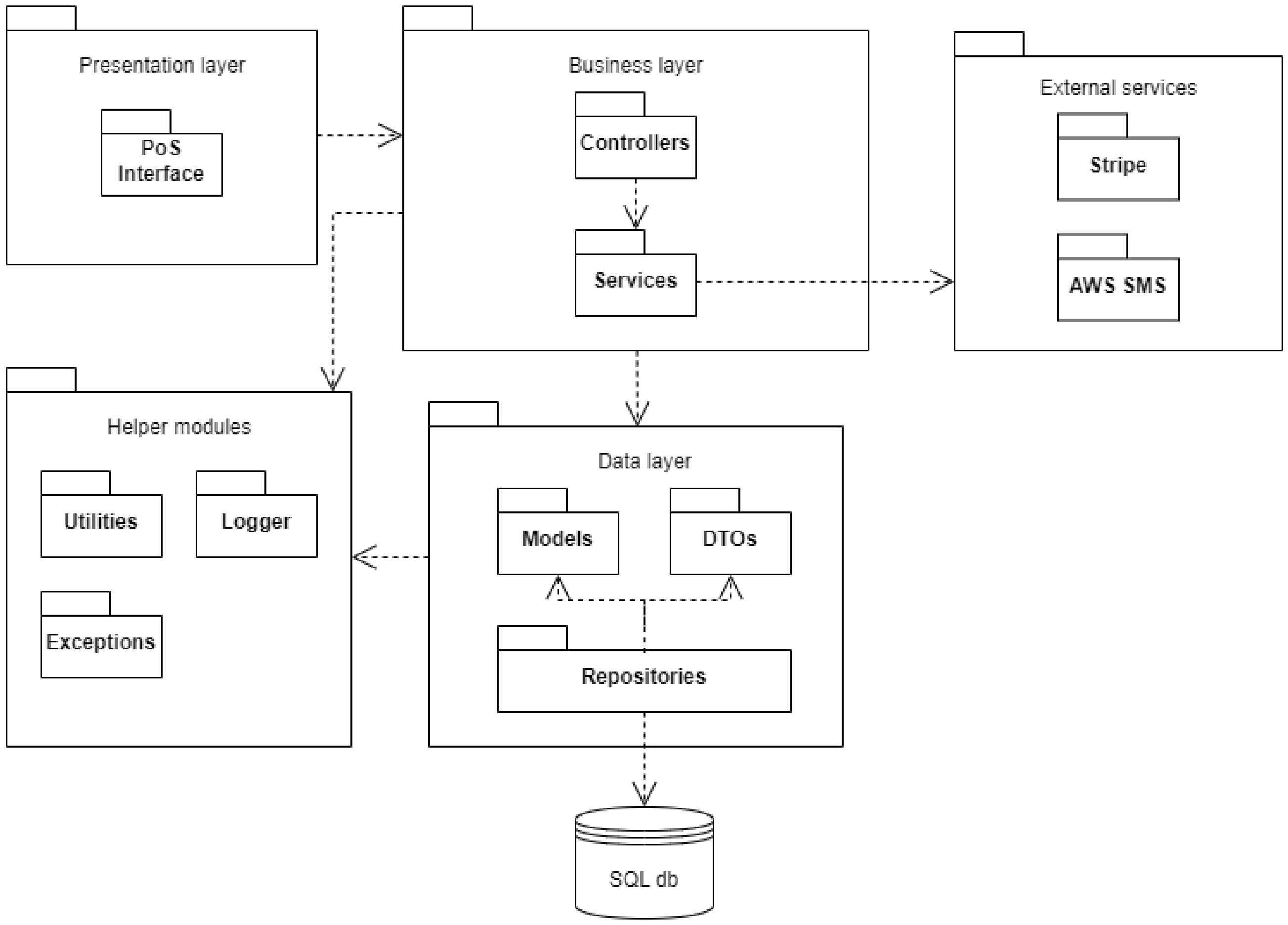


Figure 18 "High level architecture package diagram"

System consists of four main parts: presentation, business, data layers, helper modules and SQL database. It is also dependant on external Stripe's payment and AWS SMS notification services. System is designed as a simple server client structure where most of the logic resides on business server. Client application makes calls to the server chich are handled by the controllers and executed by services, all of the states are stored in the database which is managed using repository design patter.

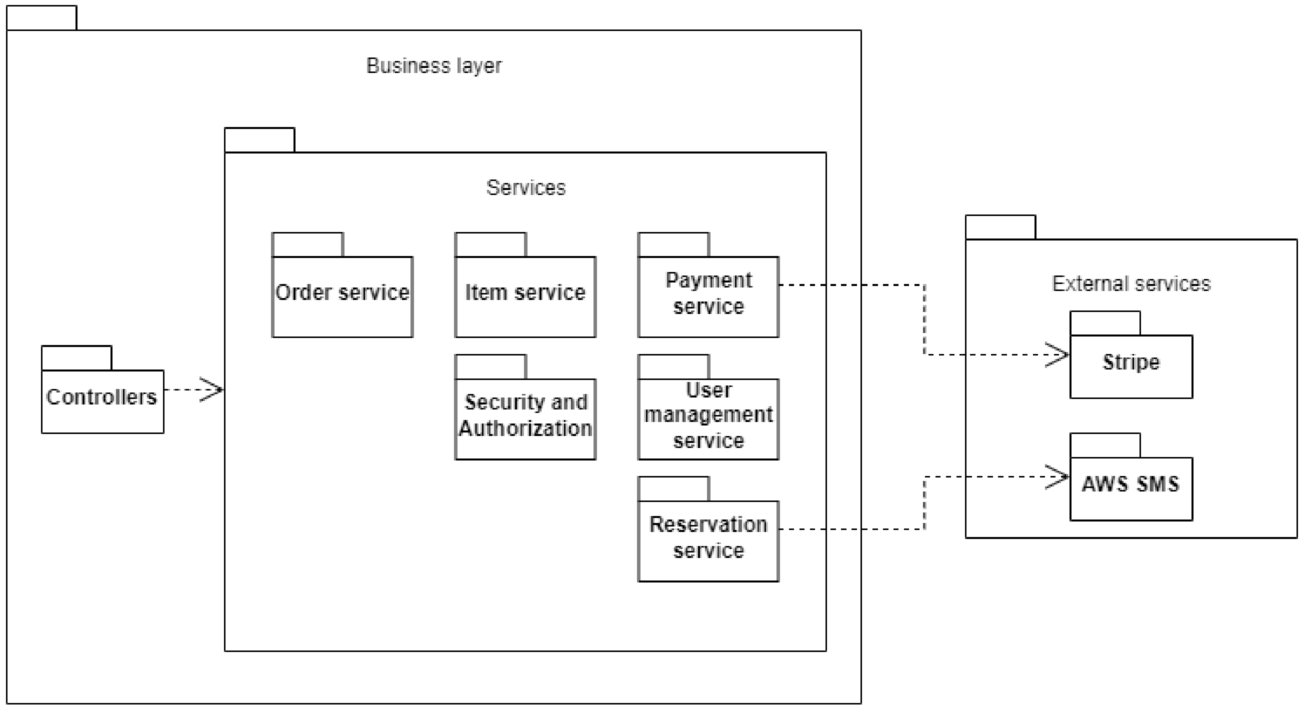


Figure 19 "Business layer package diagram"

This diagram shows the main services that should operate inside the system (Figure 19). After receiving API calls from the client-side application controllers manage those calls and trigger appropriate service operations. Order, Item and User management handle the creation, deletion and updating of corresponding data within the system. Security and Authorization handles new user registration, verification and managing tokens. Payment service processes payments made by gift cards, cash and credit/debit by setting up Stripe card reader terminals and sending payment requests to Stripe API. Reservation service manages appointments and SMS notifications through AWS to remind clients about their upcoming reservation dates.

4. System entity data model

The following class diagram (Figure 20) represents the entity datal model of the planned system. It consists of n main parts: Orders, Users, Payments, Reservation and Items. Most of the design choices can be explained by the need to preserve historical order data which will be explained in the following paragraphs. In all of the entities, for the sake of simplicity, primary keys are chosen to be a simple id integer field but can be changed if felt needed. Foreign keys also are not explicitly written, they are represented by the relationship arrows.

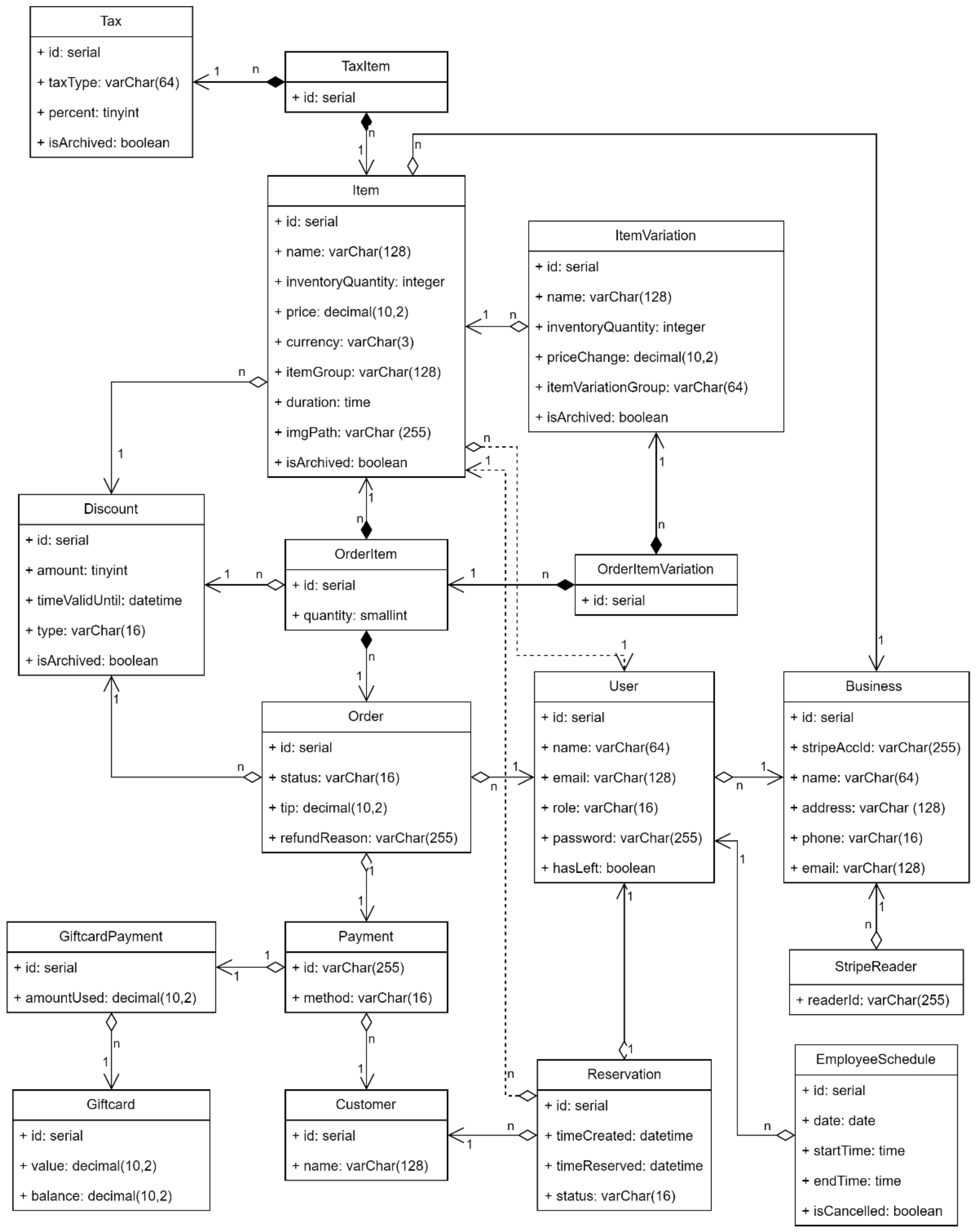


Figure 20 "Class diagram of system's entity data model"

4.1. Management strategy

One of the system requirements involves preserving historical data of past orders. To accomplish this Item, ItemVariation, Tax and Discount entities should never be deleted or updated. If one of the records were to be changed or deleted it should rather be archived and new record inserted if needed. The archived record is identified by isArchived field.

4.1.1. Alternative

Although not required, it is a better practice to archive old entries by moving them to a separate archival database, without using the isArchived flag. This would improve data retrieval speed, allow for allocating more resources and in general make the main database way more manageable.

4.2. Order entity

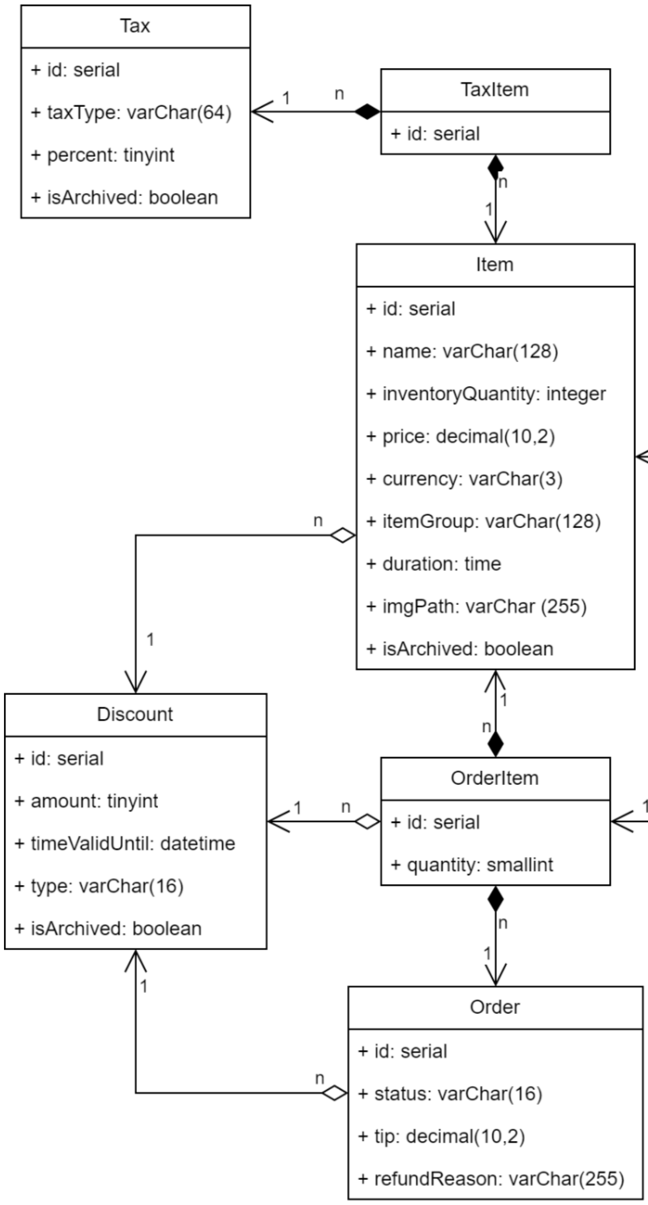


Figure 21 "Order management entity model part"

Order and its adjacent entities are used to store information about customers' orders (Figure 21). There are three junction tables: OrderItem - connects Order and Item, since one order can have multiple goods or services (will be referred to as item) connected to it and one item can be included in many orders; OrderItemVariation – connects variation of an item that was selected in an order with the OrderItem; TaxItem – connects items with their taxes. Items also have a relationship with Business entity so that the system shows only items that are sold in that business

4.2.1. Storing services

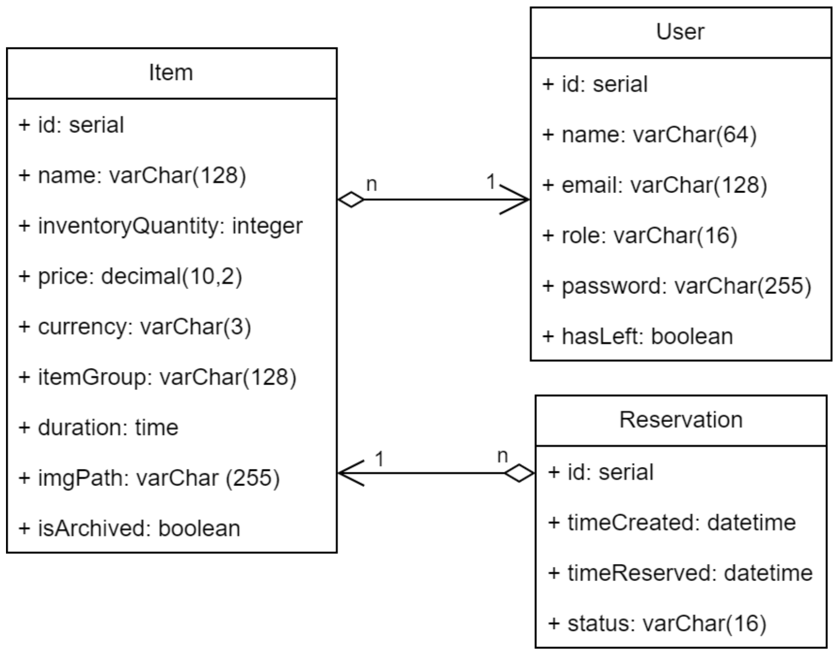


Figure 22 "Item to User relationship"

As mentioned before Item entity will store not only goods but also services. This decision was made to simplify the database since system requirements included only one difference between them, this being connection between service and an employee providing it, which is represented by optional connection between Item and User (Figure 22). Also, system requirements did not note services needing discount and tax management, but it was decided to standardize goods and services by having identical functionalities and therefore shared entity – Item. Although services will have to include duration field, which is used to calculate reservation availability in the timetable.

4.2.2. Item Variations

ItemVariation entity stores different variations that can be applied to items. Items can have multiple different variations applied to them, but never from the same variation group indicated by itemVariationGroup. For example, a latte can have decaf and oat milk variations but cannot have oat milk and soymilk variations at the same time. Variations also change the items price indicated by priceChange field.

4.2.3. Other remarks

Discounts can be applied to either the whole order or automatically to some items, therefore Order and Item entities both have connections to Discount entity.

For the preservation of historical order information there are important rules when archiving old records:

- Item entity should be archived together with ItemVariation, also new TaxItem needs to be created.
- Tax entity's archival requires archival of all associated Items.

4.3. Payment entity

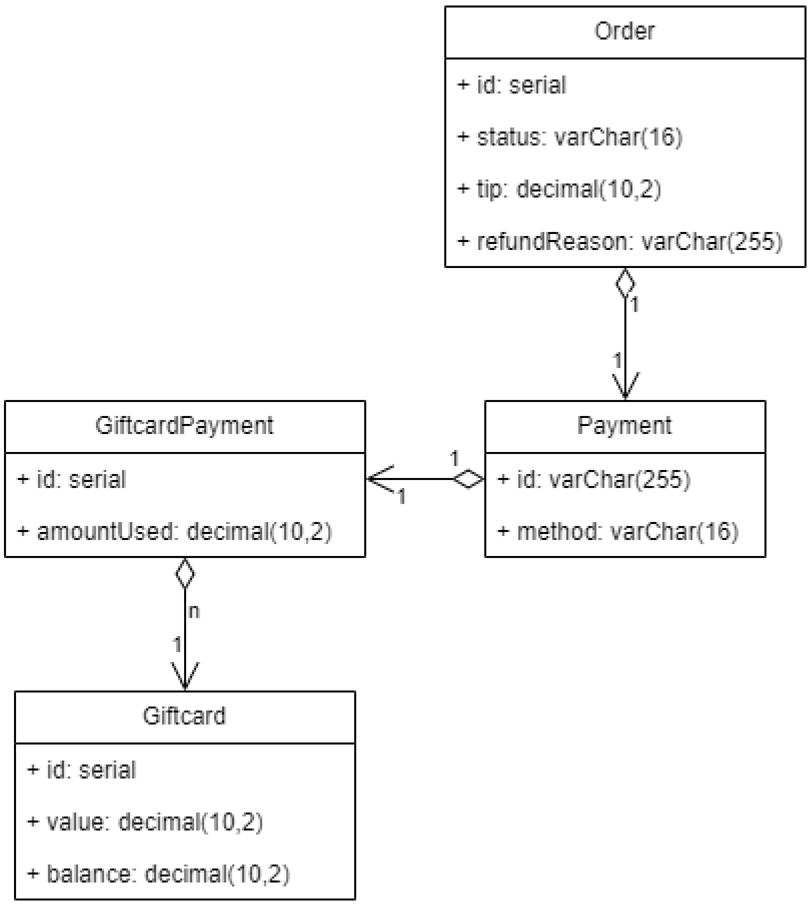


Figure 23 "Payment entity model part"

Each Order entity will have a status field, which can have values of open, closed, cancelled paid or refunded. If order is open, it can be modified, if closed, then it is waiting payment. Paid or refunded orders have a payment entity and depending on payment method it will have card, cash or gift card in the method field. Cash and gift card will have a randomly generated payment id and card will have stripes Charge Id as the key. Gift card method will also have GiftcardPayment entity associated to verify the payment and track gift cards balance. Refunded order's refundReason field should contain the reason of refund, otherwise it is left empty.

4.4. User entity

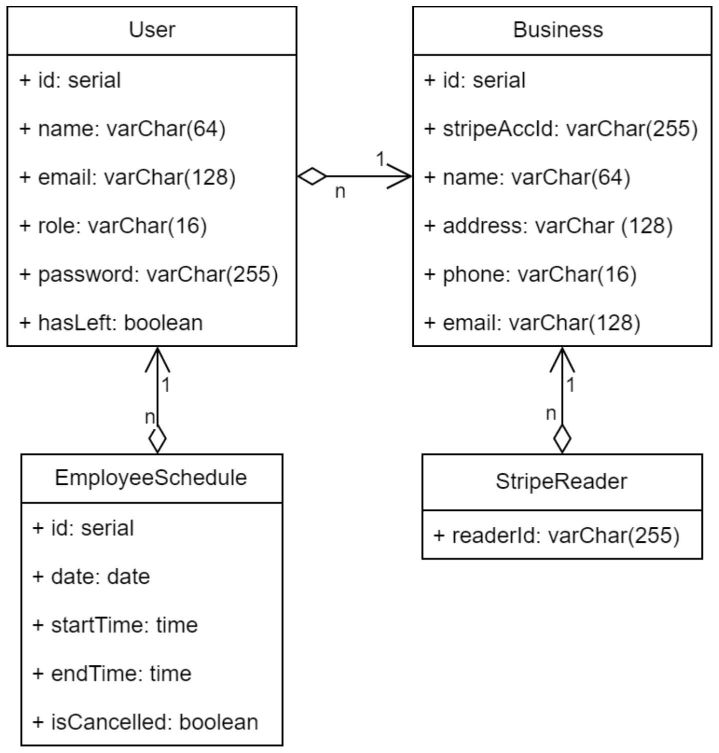


Figure 24 "User entity model part"

Ther will be three types of users in this system: owner (business owner), admin, employee. Business owners will be able to register their business in the system, add stripe readers and employees, also edit them. Admin will have the same privileges as the business owner, but he will not be limited to one business, he can edit all of the businesses data. Employees can only create and modify orders if Order status field allows it. Employees will also have association with EmployeeSchedule entity to represent their work schedule, each entry represents employee's workday.

4.5. Reservation entity

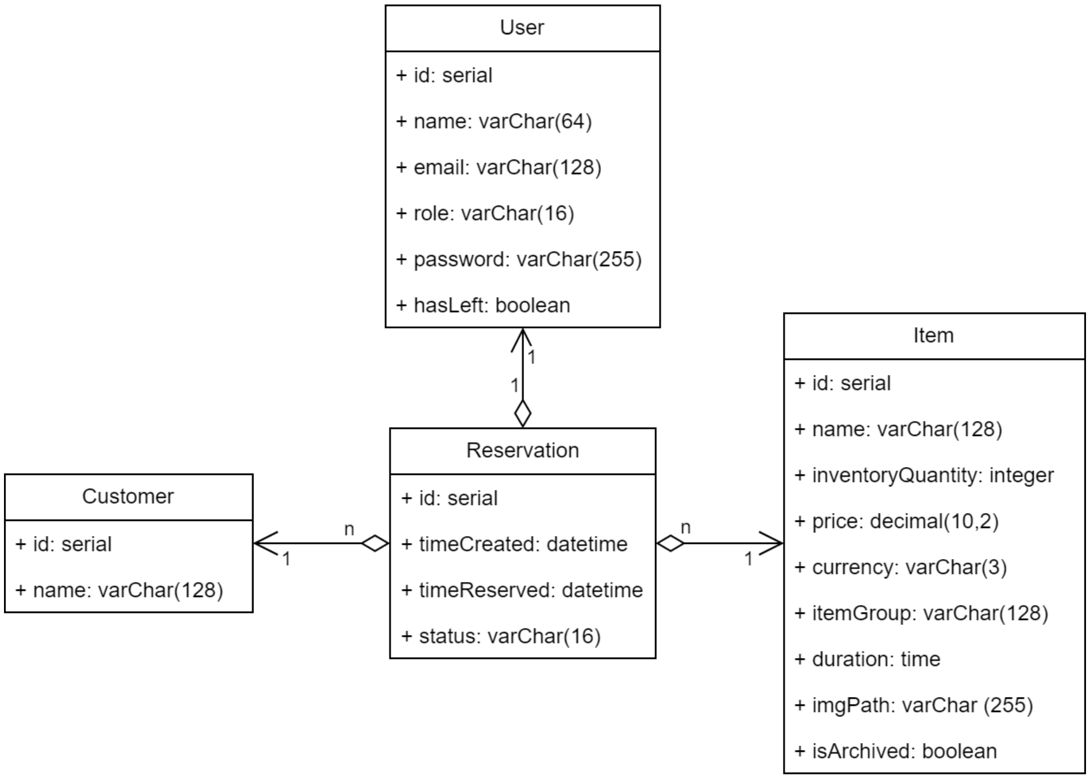


Figure 25 "Reservation management entity model part"

Each will have Customer, User (employee) and Item associated with it. Item can be a service or a table in a bar, which can be marked as a service. User is the employee who registered the reservation and will provide the service if applicable. Customer is only required to identify to whom the reservation belongs to.

4.6. Entity field explanations

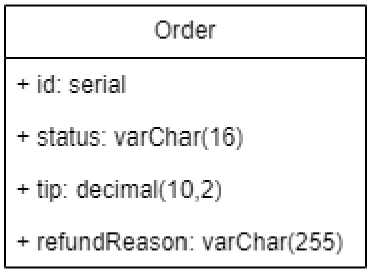


Figure 26 "Order entity"

- status – status of registered order, can be open, closed, cancelled or refunded.
- tip – tip amount left by the customer.
- refundReason – refund reason given by the customer.

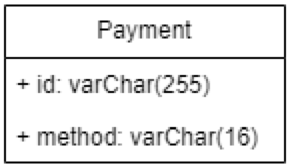


Figure 27 “Payment entity”

- method – states the method of the payment, can be cash, giftcard or card.

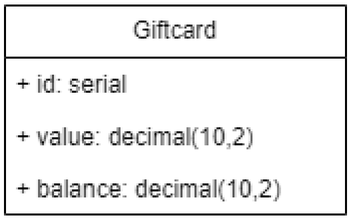


Figure 28 “Giftcard entity”

- value – value of the gift card when first made.
- balance – current amount of value left in the gift card.

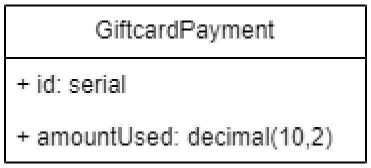


Figure 29 “GiftcardPayment entity”

- amountUsed – amount of balance used in a gift card payment.

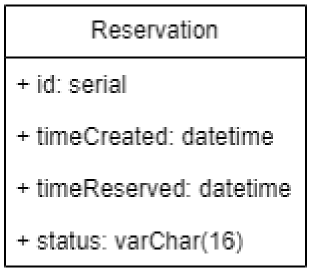


Figure 30 “Reservation entity”

- timeCreated – date and time when reservation was made.
- timeReserved – date and time of the reservation.
- status – status of registered reservation, can be open or cancelled.

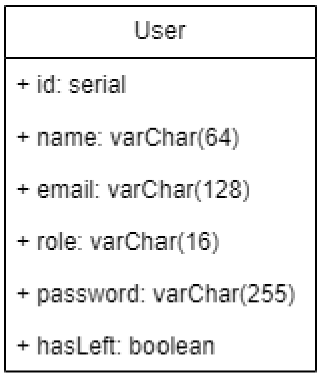


Figure 31 "User entity"

- name, email – general user information.
- role – users' role in the system/business: employee, owner, admin.
- password – users password hash.
- hasLeft – identifies if a person is still an employee in the company.

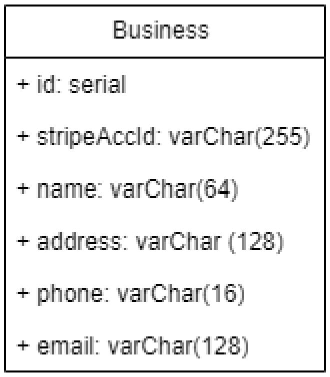


Figure 32 "Business entity"

- name, address, phone, email – general business information.
- stripeAccId – business specific identification key that is used for communicating with stripes system.

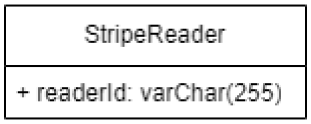


Figure 33 "StripeReader entity"

- readerId – id given by stripe to identify and communicate with one of the connected stripe card reader terminals.

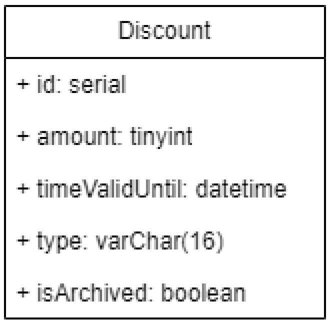


Figure 34 "Discount entity"

- amount – 1-3-digit number representing percentage of the discount.
- timeValidUntil – discount expiration date and time.
- type – defines discount type: item, order. Item discount is automatically applied to items and order discount is manually selected by employee to apply for the whole order.

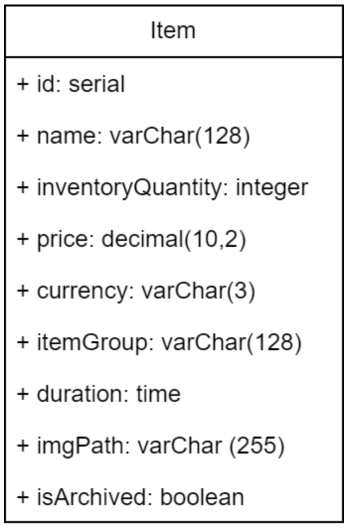


Figure 35 "Item entity"

- name – name of product or service.
- inventoryQuantity – product stock available in inventory.
- price – price of product or service.
- currency – three-character currency code for the price.
- itemGroup – used to identified similar items. Mainly for UI functionality.
- duration – duration of a service (service only).
- imgPath – a path to an image of an item in the file server.

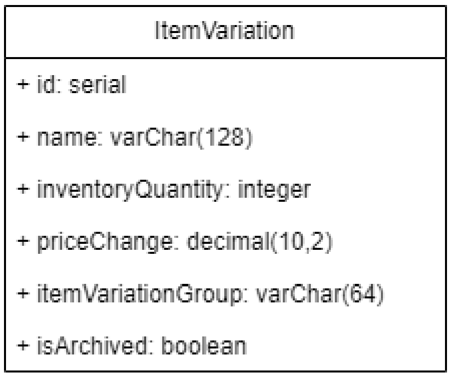

ItemVariation
+ id: serial
+ name: varChar(128)
+ inventoryQuantity: integer
+ priceChange: decimal(10,2)
+ itemVariationGroup: varChar(64)
+ isArchived: boolean

Figure 36 "ItemVariation entity"

- name – item variation name.
- inventoryQuantity – product variation stock available in inventory.
- priceChange – stores price change of adding a variation to an item.
- itemVariationGroup - used to identified similar item variations. Variations from the same group cannot be applied to ordered item.

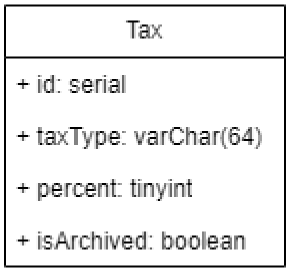


Figure 37 "Tax entity"

- taxType – describes the tax.
- percent - 1–3-digit number representing percentage of tax.

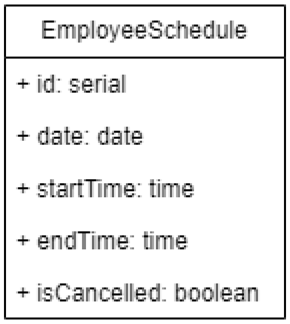


Figure 38 "EmployeeSchedule entity"

- date – date of employee's working day
- startTime – employee's shift start time.

- endTime - employee's shift end time.
- isCancelled – identifies if employee will still work that day. Value is true if his shift was cancelled due to illness or other reasons.