Point of sale system

[**Introduction 2**](#_8momj3gmagen)

[**Flows 2**](#_hnhdiymodj74)

[Order 2](#_6x04q21c9o4b)

[Order discount 3](#_weyyiepez03y)

[Available discounts 3](#_cuplhivzjatb)

[Reservation 4](#_g4u0ztceul9r)

[Employee 5](#_d0cuavscjwiy)

[**Data model 5**](#_40ocslrosky4)

[**Roles and permissions 5**](#_drw2zunk0lp8)

[Roles 5](#_8o5j9exoule)

[Data manipulation 6](#_ufhk1cv0jvrg)

[**Audit data 6**](#_3iwcotu2ulwj)

[**API contracts 7**](#_c1xg8axwy1f)

[**Appendix 7**](#_vqvvc0z7h5m6)

[Audit variation A 7](#_a1p050134i29)

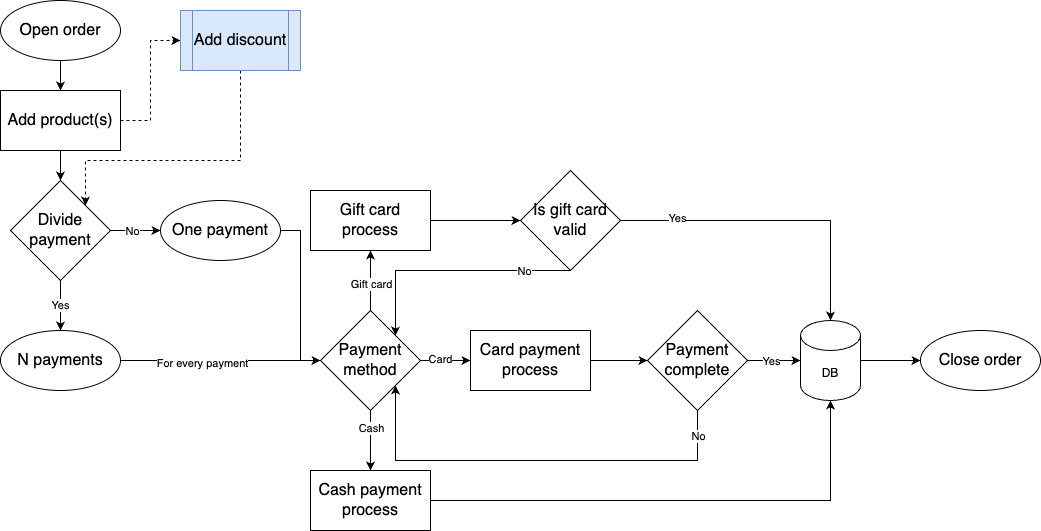
Technical documentation

# Introduction

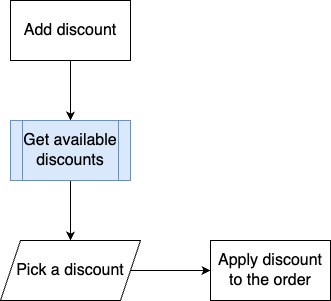
# Flows

The main program flows are presented here. The blueish color is representing an inner process that will be shown in more detail in later flows.

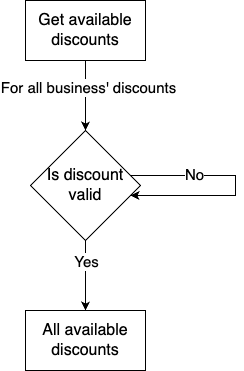
## Order



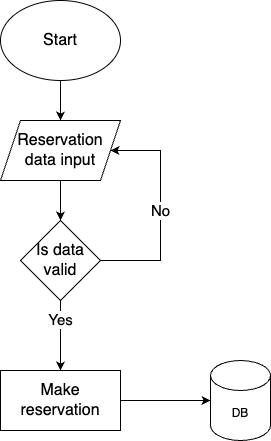
## Order discount



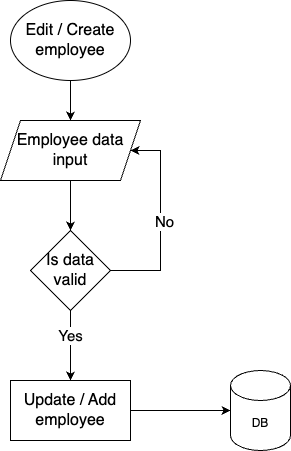
## Available discounts



## Reservation



## Employee



# Data model

# Roles and permissions

## Roles

Each endpoint in our system is assigned a unique permission. For instance, permissions like:

* Employee.Get
* Employee.Put
* Employee.Post

and others will represent distinct permissions. These permissions are stored in the SystemModule table, where each is associated with a unique integer ID.

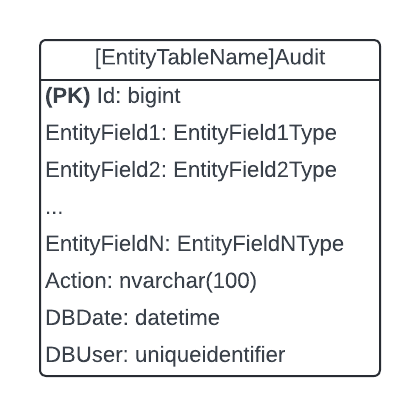
Additionally, businesses will have the flexibility to define custom roles, referred to as UserTypes, tailored to their specific needs. Each role can be assigned a set of permissions, which will be maintained in a many-to-many relationship table called UserSystemModuleRow.

## Data manipulation

Each endpoint should verify whether the user initiating the request intends to modify data belonging to their own business. If this condition is not met, an error message should be displayed to the user.

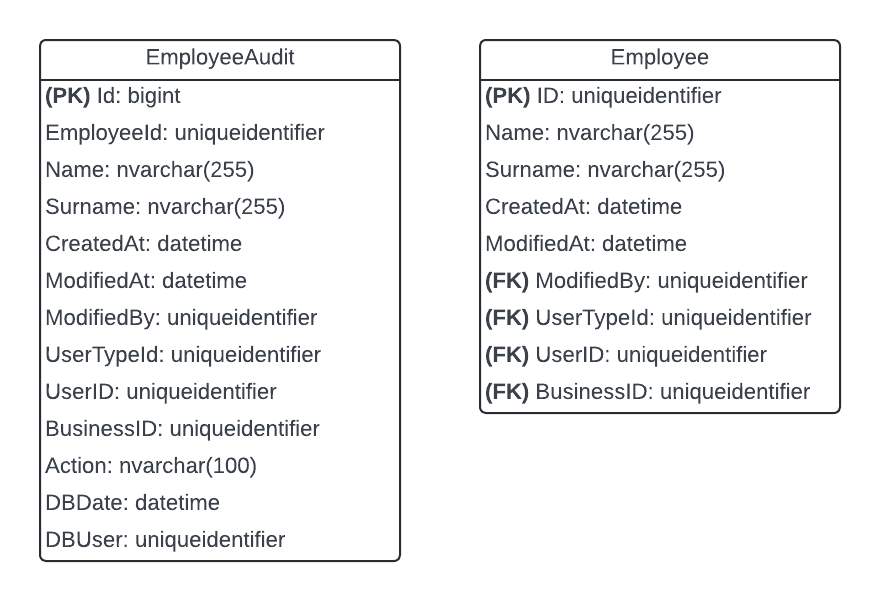
# Audit data

A data audit should be implemented using triggers on every entity table, triggered by the actions INSERT, UPDATE, and DELETE.



The table presented here is an abstract audit table, containing every field from the corresponding entity table. In addition, it includes four extra fields: **Id**, **Action**, **DBDate**, and **DBUser**. Each entity table will have three triggers (one for each action). These triggers will insert the previous data into the audit table and populate the additional fields as follows:

* **Id**: An auto-incrementing integer.
* **Action**: A value from the set {“Insert”, “Update”, “Delete”}, representing the action performed on the entity table.
* **DBDate**: The date the action was performed.
* **DBUser**: The user who performed the action.



Audit table example. In summary, the audit table contains all the fields of the entity table, with the addition of these four new fields: **Id**, **Action**, **DBDate**, and **DBUser**.

# API contracts

# Appendix

## Audit variation A

We also considered an "Insert-only" data model, where data would not be deleted from the entity table but instead marked with a timestamp. The latest data would be retrieved based on this timestamp.

However, this approach did not seem ideal for several reasons:

* The main table would grow rapidly, which would slow down data retrieval.
* Our data model includes several many-to-many relationships, which would make data retrieval more complex in this scenario.
* Retrieving historical data from the audit table is faster and more efficient than doing so from the main entity table.