VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

MIDTERM PROJECT

SERVICE-ORIENTED PROGRAMMING

THE ORDER MANAGEMENT SUBSYSTEM

Instructor: **MSc. Duong Huu Phuc**
Students: **Tran Nhut Anh - 521H0491**
Tran Quoc Bao - 521H0494
Le Nguyen Viet Hiep - 521H0398
Group: **03**

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

MIDTERM PROJECT

SERVICE-ORIENTED PROGRAMMING

THE ORDER MANAGEMENT SUBSYSTEM

Instructor: **MSc. Duong Huu Phuc**
Students: **Tran Nhut Anh - 521H0491**
Tran Quoc Bao - 521H0494
Le Nguyen Viet Hiep - 521H0398
Group: **03**

HO CHI MINH CITY, 2024

ACKNOWLEDGMENT

Best Regards,

We extend our sincere thanks to MSc. Duong Huu Phuc for your invaluable guidance and support throughout the preparation of our midterm project. Your expertise and assistance have been instrumental in our success.

During the process of completing the project, we sincerely apologize for any errors that may occur. We kindly ask for your understanding and forgiveness for any shortcomings. We greatly appreciate your guidance and support throughout this process.

Tran Nhut Anh
Tran Quoc Bao
Le Nguyen Viet Hiep

DECLARATION OF AUTHORSHIP

We now declare that this thesis was carried out by myself under the guidance and supervision of Mr. Duong Huu Phuc; and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by our work and have been duly acknowledged in the reference part.

In addition, other comments, reviews, and data used by other authors, and organizations have been acknowledged, and explicitly cited.

We will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused by our work (if any).

Ho Chi Minh City, April 7th, 2024

Authors

CONTENTS

LIST OF FIGURES

CHAPTER 1. UNDERSTANDING THE SYSTEM

1.1 Introduction

The restaurant wants to implement the subsystem called ‘the order management subsystem’. This new subsystem comes from the need to optimize the ordering and serving process of food service in this restaurant. Before this module appeared, the traditional ordering method was still applied to all stores in the FnB. However, there are many problems that occur such as: Employees have to wait to take notes on dishes, need to verify dishes, ordered dishes have to wait for servers to bring them to the kitchen counter, in addition to taking notes. Manually can lead to errors, etc.

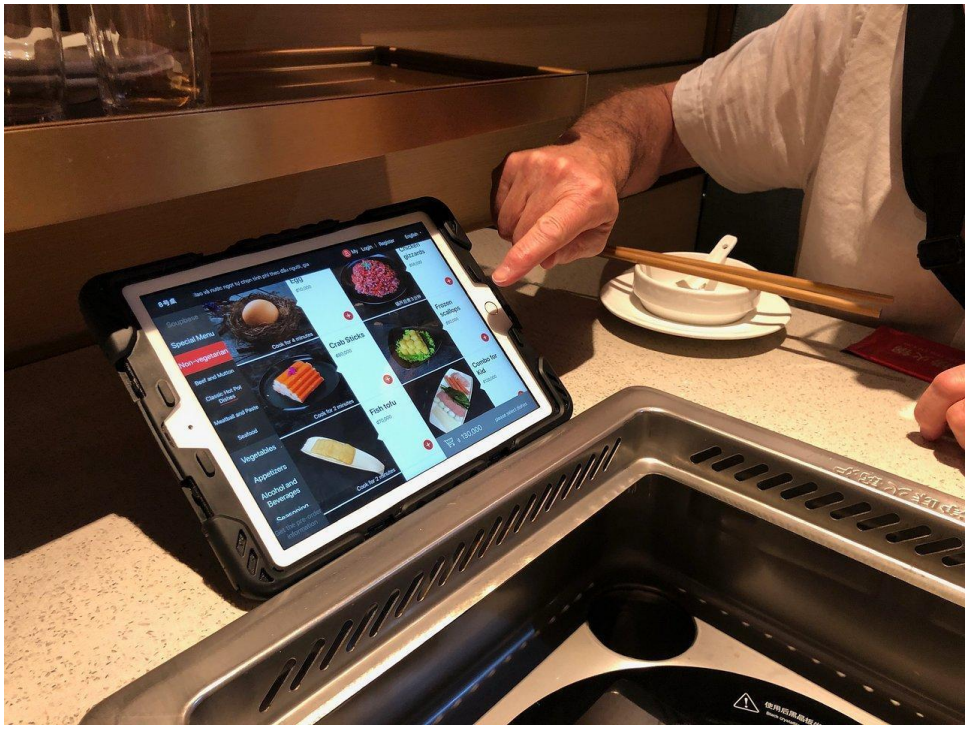

Figure 1: The ordering tablet at Haidilao restaurant.

In 2010, China launched the initiative to order food using tablets with the goal of enhancing customer experience. After a while, large restaurant

chains there and in some other countries quickly adopted this system to improve customer experience. Currently, famous large restaurants such as haidilao, manhwa,... have become one of the characteristics of the restaurant. After being widely applied, the tablet ordering module has received positive reviews from customers and brought many benefits to restaurants that apply it.

1.2 System Overview

The subsystem consists of the following core services:

- **Authentication service**:
  - Verifies user accounts during login.
  - Validates user credentials.
  - Checks user roles (waiter, manager, chef) for access control.
  - Clears the user's current session on logout.
- **Menu service**:
  - Manages the status of dishes in the menu based on ingredient availability.
  - Provides detailed information about each dish (price, description, image, etc.).
- **Order service**:
  - Manages the entire order process from placement to delivery.
  - Handles order creation and modification.
  - Integrates with the Menu service to verify dish availability and update ordered item status.
  - Facilitates communication between diners and staff for real-time order placement.

- **Payment service**:
    - Coordinates with the Order service for payment processing.
    - Handles secure and convenient payment transactions for diner status.
    - Generates detailed payment records for the financial reports.

1.3 System Scope

The scope of the food ordering module is limited to staff and kitchen tablets in the actual restaurant environment. It is specifically designed to handle orders placed directly at the restaurant, rather than catering to online orders or deliveries. This subsystem operates independently of other subsystems in the restaurant management and operation system.

There are three main interacting objects in this subsystem: waiter, manager, and chef. In the context of this subsystem with limited objects and functions, let's explore the roles of waiter, manager, and chef:

- **Waiter:**
    - Waiter is responsible for interacting directly with diners, taking orders and delivering dishes to the chef.
    - Within the system, the waiter can log in using their credentials and access specific features and functions assigned to their role.
    - Functions performed by the waiter include:
        - Taking diner orders and entering them into the system.
        - Checking the status of orders and providing updates to diners.
        - Handling diner requests or modifications to orders.
        - Marking orders as completed once the dishes are prepared.
        - Generating bills for diners.

- **Manager:**
    - Performing the functions of the waiter.
    - Checking reports on sales, revenue, and order statistics.

- **Chef:**
    - The chef role remains separate and focuses on the culinary aspects of the order management system.
    - Chef logs in with their credentials and has access to functionalities specific to kitchen operations.
    - Functions performed by the chef include:
        - Viewing and acknowledging new orders received from the staff.
        - Updating the status of orders to indicate preparation, cooking, or ready for serving.
        - Managing inventory or ingredient availability for each dish.

1.4 Practical Significance

- This subsystem is a crucial component that brings efficiency and accuracy to the order processing workflow.
- It enables real-time order updates, seamless communication, effective inventory management, and data analysis capabilities.
- With this subsystem, restaurants can streamline operations, reduce errors, enhance coordination, and ultimately provide better customer satisfaction while improving overall productivity and profitability.

CHAPTER 2. ANALYSIS AND DESIGN SYSTEM

2.1 System Requirements

2.1.1 Functional Requirements

- **Process-Oriented:**
    - The system must authenticate users by verifying their credentials.
    - The system must grant access to authorized users based on their roles.
    - The system must allow chefs to modify the status on menu items.
    - The system must provide a UI for diners to select dishes.
    - The system must generate unique order IDs and associate them with the corresponding dishes and diners.
    - The system must track the status of orders and update it as they progress through different stages.
    - The system must facilitate secure and convenient payment options for diners.
    - The system must calculate the total amount due for each order.
    - The system must process payment transactions and provide confirmation to diners and staff.

- **Information-Oriented:**
    - The system must store and manage user account details.
    - The system must maintain a database of dishes with details.
    - The system must store information about each order.
    - The system must store payment transaction details.
    - The system must generate payment records for the financial reports.

2.1.2 Non-functional Requirements

- **Operational:**
    - The system can run on tablet devices.
    - The system downtime for maintenance should be minimized and scheduled during off-peak hours.
    - The system must support the addition of new restaurant locations without requiring major modifications.
    - The system must consistently perform its functions without unexpecteds failures.
- **Performance:**
    - The system will respond quickly to user actions.
    - The system should process a large number of orders efficiently to ensure timely delivery and minimize delays.
    - The system should be available for use during the restaurant's operating hours.
- **Security:**
    - The system must ensure strong user authentication and authorization and prevent unauthorized access.
    - The system must have built-in protection against viruses, worms, etc.

2.2 System Functional Diagram

2.2.1 Use Case Diagram

Figure 2: Use Case Diagram of Halo Hotpot.

2.2.2 List of Actors

The system has 3 types of users:

- Waiter
- Manager (inherits from Waiter)
- Chef

2.2.3 List of Use Cases

Use Case ID	Use Case Name	Description	Relationship
UC01	Login	User logs into the system.	-
UC02	Logout	User logs out of the system.	Extend: UC01
UC03	Create order	The waiter creates an order from the system interface.	-
UC04	Enter table ID	The waiter enters the table ID after creating the order.	Include: UC03
UC05	Enter the number of diners	The waiter enters the number of diners after creating the order.	Include: UC03
UC06	Add dish to order	The waiter adds the dish to the diner's order.	-
UC07	Confirm serving	The waiter confirms that the dish has been served on the system screen when delivering it to the diner.	-
UC08	Make payment	The waiter hands the bill to the diner.	-
UC09	Check total report	The manager views the total orders.	-
UC10	Check total report by date	The manager views the total orders by date.	Extend: UC09
UC11	Update menu status	The chef updates dish status on the menu based on ingredient inventory.	-
UC12	Update dish status	The chef updates dish status based on processing.	-

Figure 3: List of Use Cases

2.2.4 Use Case Description

Use case ID	**UC01**
Use case name	Login
Brief	User logs into the system
Primary Actor	Manager, Waiter, Chef
Secondary Actor	None
Pre-condition	User had a valid account of the restaurant.
Trigger	User intends to use and needs to log into the system.
Post-Condition	- User is successfully logged into the system. - Users can access assigned functionalities and perform tasks based on the role and permissions.
Main scenario	1. The system presents a login screen. 2. User enters username and password. 3. The system verifies the credentials against the stored user database. 4. The system displays the user's dashboard.
Alternatives	User account is invalid.

Figure 4: UC01 – Login.

Use case ID	UC02
Use case name	Logout
Brief	User logs out of the system
Primary Actor	Manager, Waiter, Chef
Secondary Actor	None
Pre-condition	Users must be logged into the system.
Trigger	User has successfully completed tasks within the system and wishes to securely log out.
Post-Condition	- The user's session is terminated, the user is no longer authenticated within the system. - User is redirected to the login page.
Main scenario	1. User clicks on the "Sign Out" button. 2. The system clears the current session and directs the user to the login page.
Alternatives	None

Figure 5: UC02 – Logout.

Use case ID	UC03
Use case name	Create order
Brief	Waiter creates an order from the system interface.
Primary Actor	Waiter
Secondary Actor	Diner
Pre-condition	Waiter logged in to the system.
Trigger	Waiter enters the table ID and the number of dinners.
Post-Condition	The order is created and added to the system.
Main scenario	1. Waiter selects the "Add Order" in the home page. 2. The system prompts the waiter to enter the ID of the table and capacity. 3. Waiter clicks the "Create" option and the new order is created.
Alternatives	None

Figure 6: UC03 – Create order.

Use case ID	UC04
Use case name	Enter table ID
Brief	Waiter enters the table ID after creating the order.
Primary Actor	Waiter
Secondary Actor	Diner
Pre-condition	Waiter has clicked on "Create New Order".
Trigger	Waiter starts this use case when a customer requests a specific place at a restaurant.
Post-Condition	The ID of the table is information of the new order.
Main scenario	1. Waiter enters the ID of the table after clicking on “creating a new order”. 2. Waiter proceeds to the next step in the order creation process.
Alternatives	The table ID is invalid.

Figure 7: UC04 – Enter table ID.

Use case ID	UC05
Use case name	Enter the number of diners
Brief	Waiter enters the number of diners after creating the order.
Primary Actor	Waiter
Secondary Actor	Diner
Pre-condition	Waiter has clicked on "Create New Order".
Trigger	Waiter starts this use case when asking about the capacity of diners.
Post-Condition	The number of diners is information about the new order.
Main scenario	1. Waiter enters the number of diners after having the table ID. 2. Waiter finishes filling in the details and the waiter can create a new order.
Alternatives	The number of diners is too large to sit at the same table.

Figure 8: UC05 – Enter the number of diners.

Use case ID	UC06
Use case name	Add dish to order
Brief	Waiter adds the dish to the diner's order.
Primary Actor	Waiter
Secondary Actor	Diner
Pre-condition	The order must have been created before.
Trigger	Waiter starts this use case when a new order is created or diners want to add more dishes.
Post-Condition	The new dishes are added to the order.
Main scenario	1. Waiter adds dishes to the order based on the request of diners. 2. The order details are sent to the cooking place with pending status..
Alternatives	Dish is out.

Figure 9: UC06 – Add dish to order.

Use case ID	UC07
Use case name	Confirm Serving
Brief	Waiter confirms that the dish has been served on the system screen when delivering it to the diner.
Primary Actor	Waiter
Secondary Actor	Chef
Pre-condition	Dish has been served to diners.
Trigger	Waiter uses this use case when the dish has been served.
Post-Condition	- The confirmation status of the waiter screen will be removed. - The dish information will disappear from the kitchen screen.
Main scenario	1. Waiter clicks the "Servings" panel after the dish is served to the diners. 2. The system shows the confirmation of the served dish. 3. Waiter confirms by clicking on the check mark. 4. The dish information disappears from the chef dashboard.
Alternatives	None

Figure 10: UC07 – Confirm Serving.

Use case ID	**UC08**
Use case name	Make payment
Brief	Waiter gives the bill to the diner.
Primary Actor	Waiter
Secondary Actor	Diner
Pre-condition	- All dishes in the order have been served to dinners. - The diners requested payment.
Trigger	The waiter uses this use case when the diners ask to make payment.
Post-Condition	- The order has been paid for by diners. - The order will appear in the Report panel.
Main scenario	1. Waiter clicks on “Create Payment” to view order information and total price. 2. Waiter clicks on the "Confirm payment" option if the diner has finished paying.
Alternatives	None

Figure 11: UC08 – Make payment.

Use case ID	**UC09**
Use case name	Check total orders
Brief	Manager views the total orders.
Primary Actor	Manager
Secondary Actor	None
Pre-condition	- Manager must be logged into the system and has permissions to access orders data. - Orders data is accurate and up-to-date.
Trigger	The manager uses this use case when the manager wants to check total orders.
Post-Condition	- Manager will see the total orders displayed on the screen. - Manager can further analyze the data using other related use cases.
Main scenario	1. Manager clicks on the "Report" panel. 2. The system displays the data of order that has been paid.
Alternatives	None

Figure 12: UC09 – Check total orders.

Use case ID	UC10
Use case name	Check total orders by date
Brief	Manager views the total orders by date.
Primary Actor	Manager
Secondary Actor	None
Pre-condition	- Manager must be logged into the system and has permissions to access orders data. - Orders data is accurate and up-to-date.
Trigger	Manager uses this use case when the manager wants to check total orders by date.
Post-Condition	- Manager will see the total orders displayed by date on the screen.
Main scenario	1. Manager clicks on the "Report" panel. 2. Manager filters data by date by selecting the date on the calendar picker. 3. The system displays order data as of this date.
Alternatives	None

Figure 13: UC10 – Check total orders by date.

Use case ID	UC11
Use case name	Update menu status
Brief	Chef updates dish status based on ingredient inventory.
Primary Actor	Chef
Secondary Actor	Waiter
Pre-condition	The dish has appeared on the menu.
Trigger	Chef uses this use case when ingredients are no longer available.
Post-Condition	The dish status on the menu will be changed.
Main scenario	1. The chef locks the dish on the processing dashboard. 1.1. The chef unlocks the dish on the processing dashboard. 2. The dish status changes to 'Locked'. 2.1. The dish status changes to 'Unlocked'. 3. The menu will show 'This Food Item is Not Available'. 3.1. The dish will be unlocked in the menu.
Alternatives	None

Figure 14: UC11 – Update menu status.

Use case ID	**UC12**
Use case name	Update dish status
Brief	Chef updates dish status based on processing.
Primary Actor	Chef
Secondary Actor	Waiter
Pre-condition	The dish has appeared on the chef dashboard.
Trigger	The chef uses this use case when there is a change in the state of the dish.
Post-Condition	The dish status will be changed.
Main scenario	1. The dish status is “Pending” when it first appears on the panel. 2. The status will be “Confirmed” if the chef confirms the dish request. 3. The status will be “Is Preparing” if the chef is processing the dish. 4. The status will be “Ready To Serve” if the head finishes cooking and notifies the waiter. 5. The status will disappear after the waiter serves the diner. *The status appears both on the order of the waiter session and the chef session.*
Alternatives	1. Ingredients are no longer available after the dish information appears on the control panel. 2. Chef will not cook this dish and the status will be “Declined”.

Figure 15: UC12 – Update dish status.

2.2.5 Entity Relationship Diagram

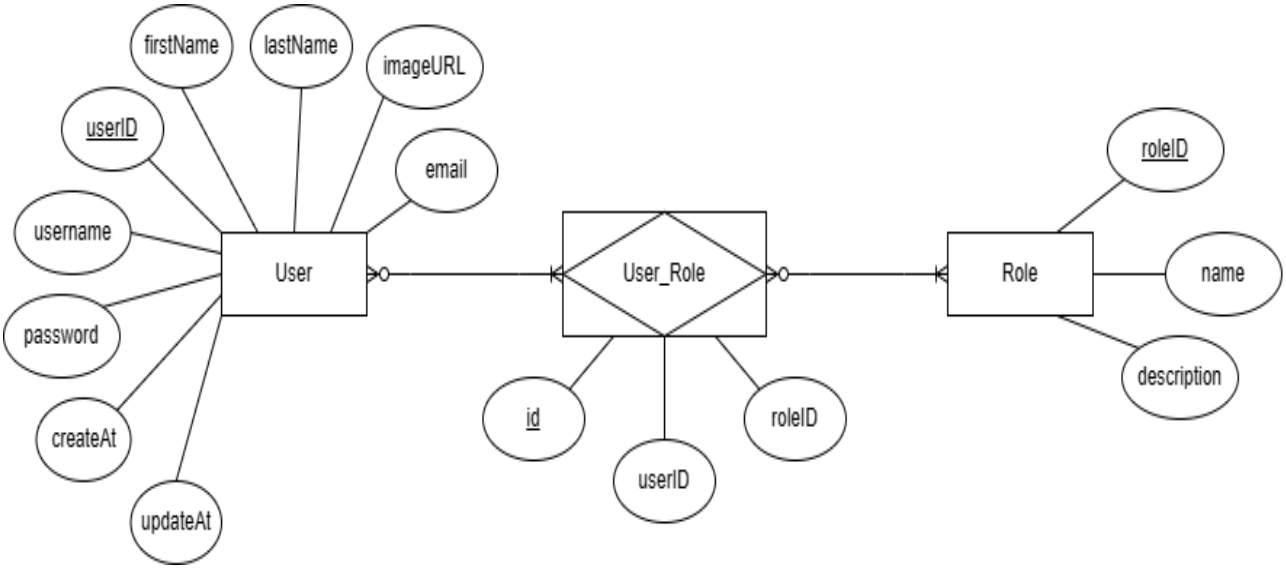

Figure 16: ERD of the authentication service.

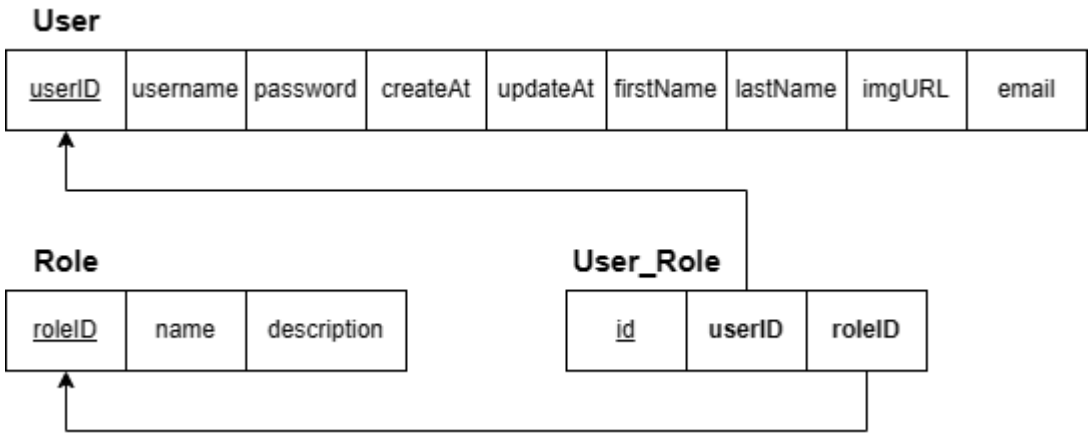

Figure 17: Mapping foreign keys of User_Role.

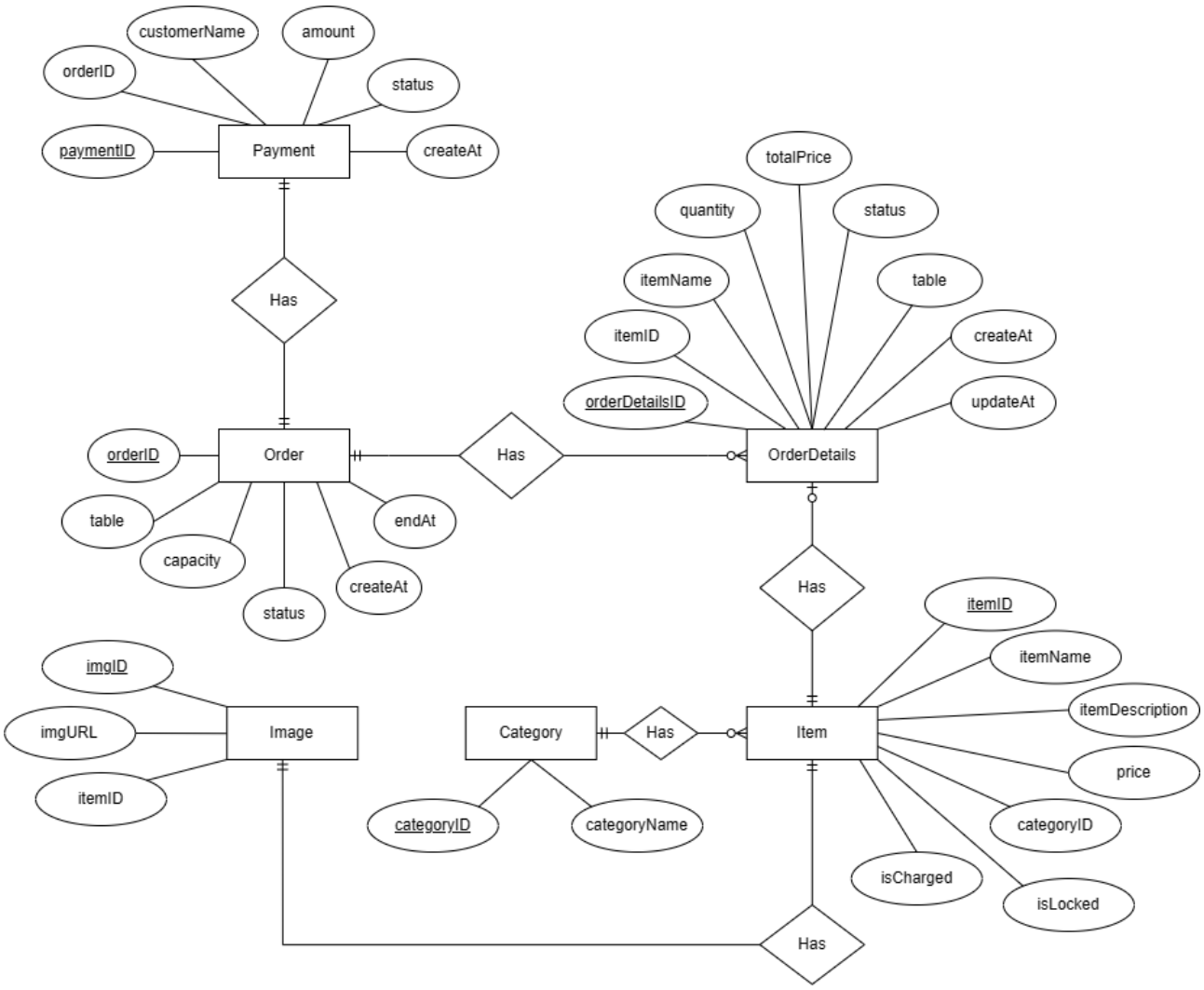

Figure 18: ERD of the restaurant service.

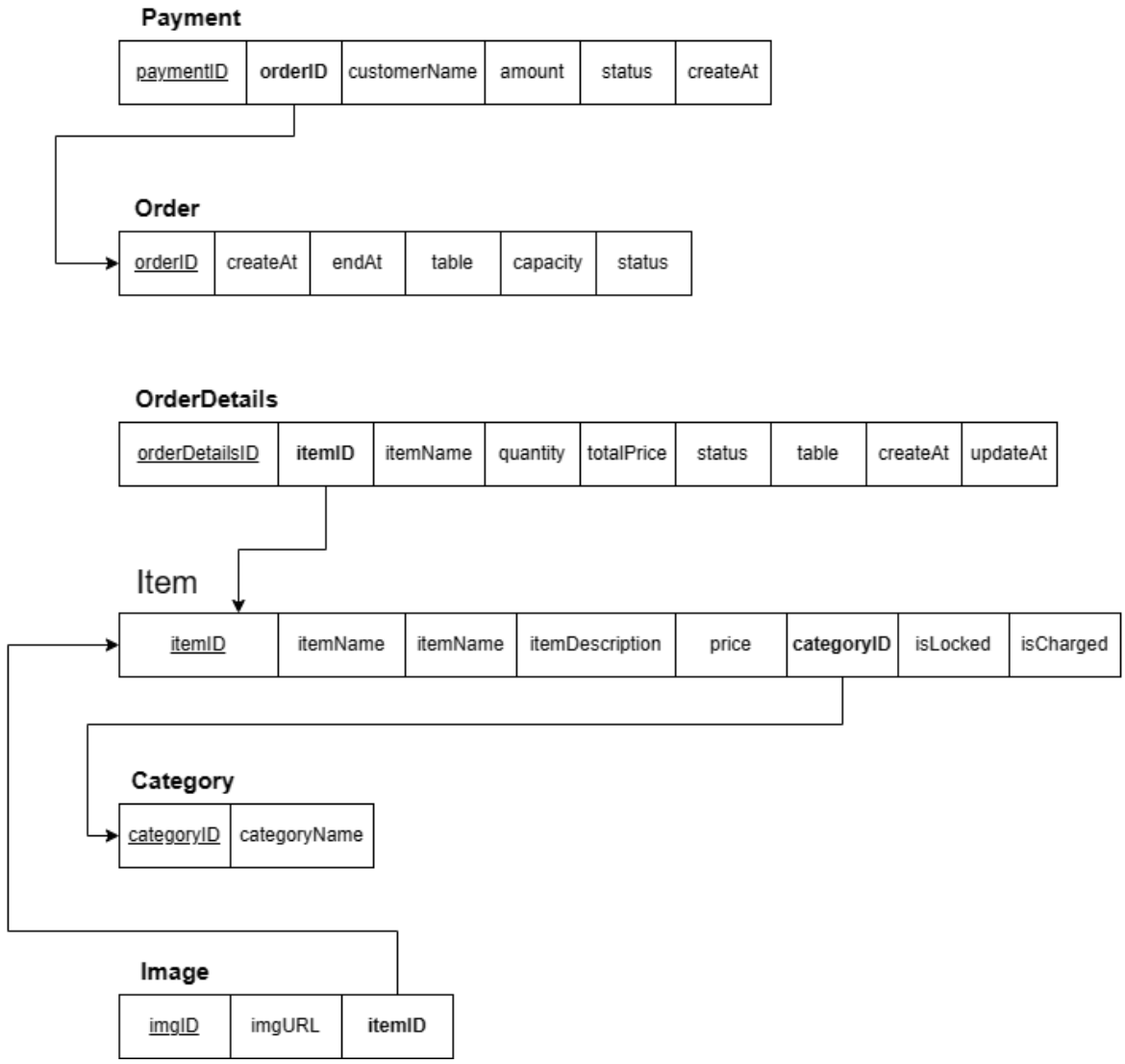

Figure 19: Mapping foreign keys of entities.

2.4 System Function API

These API documentations are specified in order management of a buffet restaurant. All API requests must be Bearer's Authentication for accessing the API resources.

- **User Login API:** This API for authenticating users with JWT token.
    - Input:
        - HTTP Method: POST
        - URL: http://localhost:8086/api/users/login
        - Request Body: JSON object containing user credentials.
    - Output:
        - Response data in JSON format containing authentication results.

- **Get Order By Status API:** Get orders by the status.
    - Input:
        - HTTP Method: GET
        - URL: http://localhost:8085/api/orders/orderstatus/:order_status
        - Path Variable:
            - order_status: The status of the order (occupied or finished).
    - Output:
        - Response data in JSON format containing the orders matching the specified status.

- **Create Order Request API:** Create new order with table ID and number of diners.
    - Input:
        - HTTP Method: POST
        - URL: http://localhost:8085/api/orders/create-order/:tableId/people/:numberOfPeople
        - Path Variables:
            - tableId: The table number assigned by the staff.
            - numberOfPeople: The number of diners at that table.
    - Output:
        - Response data JSON format confirming the successful creation of the order.

- **Get Food Menu Items API:** Get all items food in menu.
    - Input:
        - HTTP Method: GET
        - URL: http://localhost:5267/api/Items
    - Output:
        - Response data JSON format containing all the dishes in the menu.

- **Get Order Details API:** Get all orders coming from the “occupied” state order.
    - Input:
        - HTTP Method: GET
        - URL: http://localhost:8085/api/orders/orderdetails
    - Output:
        - Response data JSON format containing orders with the occupied status.

- **Update Order Details Status API:** Update the status of the specified dish in the order.
    - Input:
        - HTTP Method: PUT
        - URL: http://localhost:8085/api/orders/detail-status/:detailId
        - Path Variable:
            - detailId: The ID of the dish in the order.
    - Output:
        - Response data JSON format containing information about the successful update of the dish status.

- **Lock & Unlock Food Item API:** Modify the status of dishes in the menu.
    - Input:
        - HTTP Method: PUT
        - URL: http://localhost:5267/api/Items/status/:detailId?status={status)
        - Path Variable:
            - detailId: The ID of the specified dish in the menu.
        - Parameter:
            - status: The status of the dish (lock or unlock).
    - Output:
        - Response data JSON format containing a boolean value representing locked state (true) or unlocked state (false).

- **Get Payments By Date API:** Get payments by the specified date.
    - Input:
        - HTTP Method: GET
        - URL: http://localhost:5211/api/Payments?date={date}
        - Parameter:
            - date: The date in the ISO 8601 format, if date is not specified then retrieve all payments.
    - Output:
        - Response data in JSON format containing the payments matching the specified date.

Click here for more details about these APIs.

CHAPTER 3. IMPLEMENTATION

3.1 Overview

This chapter outlines the technology stack utilized in the development of the Order Management microservice project. The project aims to facilitate order processing and management within the system, catering to various user roles including waiters, managers, and chefs.

3.2 Technologies Used

3.2.1 Front-end

- **Framework:** React.js
- **Description:** React.js is used for the frontend interface catering to waiters, managers, and chefs. Its simplicity and efficiency in handling JSON API responses make it an ideal choice for interacting with the backend API resources seamlessly.

3.2.2 Back-end

- Menu Service:
    - **Framework**: ASP.NET Core Web API
    - **ORM** (Object-Relational Mapping): Entity Framework Core
    - **Database**: MySQL
    - **Description**: The MenuService is responsible for managing menu items, including images, within the system. ASP.NET Core Web API along with Entity Framework Core and MySQL provide robust tools for rapid API endpoint development and database connectivity.

- Payment Service:
    - **Framework:** ASP.NET Core Web API
    - **ORM** (Object-Relational Mapping): Entity Framework Core
    - **Database:** MySQL
    - **Description**: The PaymentService enables waiters to create payments and allows managers to review all payments or filter them by date. Leveraging ASP.NET Core Web API and Entity Framework Core connected to MySQL facilitates efficient CRUD operations and data management for payment-related functionalities.

- Authentication & Authorization Service:
    - **Framework**: Express.js
    - **Database**: MySQL
    - **ORM** (Object-Relational Mapping): Sequelize
    - **Description**: Express.js with Sequelize ORM is utilized for handling user authentication and authorization via JWT (JSON Web Tokens). This service generates access tokens for users, enabling access to other API resources within the project securely.

- Order Service:
    - **Framework**: Express.js
    - **Database**: MongoDB
    - **ODM** (Object-Document Mapping): Mongoose
    - **Description**: Express.js with Mongoose ODM is employed for order management services. MongoDB is chosen for its flexibility in data

modeling, allowing for dynamic addition of fields to records as per the varied requirements of order management.

- Additional Features
    - **Socket.io Integration**: Implemented Socket.io for pub/sub pattern in order management services. This enables real-time updates for waiters, managers, and chefs, ensuring prompt notification of order details without page reloads.

3.2.3 Testing & Documentation With Postman

Postman offers a comprehensive suite of tools designed specifically for testing API endpoints. It includes specialized features tailored for testing socket.io, facilitating the precise implementation of pub/sub patterns. Additionally, Postman's documentation features are intuitive and user-friendly, making them easy to learn and use effectively.

3.3 Conclusion

By leveraging a combination of ASP.NET Core Web API, Express.js, MongoDB, MySQL, and React.js, the Order Management microservice project achieves a robust and flexible architecture catering to various functionalities required for efficient order processing and management.

REFERENCES

[1] Phuc H. Duong. (2024). CS502052 - Enterprise Systems Development Concepts, Chapter 03 - Requirements Determination, Use-case Analysis [Online], 2024. Available: ch03.pdf - Google Drive

[2] Phuc H. Duong. (2024). CS502052 - Enterprise Systems Development Concepts, Chapter 05 – Designing Databases [Online], 2024. Available: ch05.pdf - Google Drive