

# *ASSIGNMENT 3*

*Restaurant Information System*

*TRAN DUC ANH DANG  
ERTESAM NASER ZARIF  
LUAN NGUYEN  
Saw Ko Ko Oo*

## Abstract

The Restaurant Information System (RIS) aims to enhance operational efficiency and customer service at The Relaxing Koala, a café and restaurant on Glenferrie Road, which is expanding its seating capacity from 50 to 150 seats. In Assignment 2, we developed an initial object-oriented design to manage reservations, order processing, kitchen operations, and payment transactions. This design included high level UML diagrams and candidate classes. However, the need for refinement and detailed modeling was identified to better manage dynamic and realtime operations. In Assignment 3, we refined this design into a comprehensive, implementation-ready framework. We enhanced class structures, defined dynamic behaviors, and integrated best practices for scalability and security. This document presents the refined design, justifies design choices, and demonstrates the system's readiness for real-world challenges.

## Table of Contents

<b>Abstract.....</b>	<b>1</b>
<b>Introduction.....</b>	<b>3</b>
<b>Detailed Design .....</b>	<b>4</b>
Final Class Diagram .....	4
Justification for Changes.....	4
Responsibilities and Collaborations.....	5
Dynamic Aspects .....	5
<b>Implementation .....</b>	<b>6</b>
Coding Standards and Practices .....	6
System Architecture and APIs .....	6
API Layer .....	6
Core Components.....	7
Database Management .....	7
Manager.....	7
User Interfaces.....	7
Advanced Design Patterns .....	8
<b>Execution and Operation .....</b>	<b>9</b>
Scenario 1: Making a Reservation .....	9
Scenario 2: Updating the Menu .....	12
Scenario 3: Placing an Order for Dine-in.....	14
Scenario 4: Placing an Order for Online .....	17
Scenario 5: User Authentication .....	21
<b>Discussion of Assignment 2 Design.....</b>	<b>24</b>
Strengths of the Original Design .....	24
Weaknesses and Areas for Improvement .....	24
Advanced Security Measures .....	25
Improved Error Handling .....	25
Detailed Integration Strategies.....	25
Lessons Learned.....	25
<b>References .....</b>	<b>26</b>

## Introduction

In the restaurant industry, advanced technology is essential for improving efficiency and customer service. Our previous project, Assignment 2, laid the groundwork for a new Restaurant Information System (RIS) for The Relaxing Koala, a café and restaurant on Glenferrie Road. As the café plans to expand from 50 to 150 seats, a sophisticated system is needed to handle more customers and improve services.

Assignment 2 focused on creating an initial design to manage key tasks like reservations, order management, kitchen operations, and payments. We introduced candidate classes and a high-level UML diagram to show how the system should work. However, we found areas that needed more detail, especially for managing dynamic and real-time operations.

In Assignment 3, we are building on this initial design to create a detailed, ready-to-implement framework. This involves refining and expanding the design to develop a more precise model that can be directly implemented. We will enhance the system's structure and relationships and define its dynamic behaviors to ensure it is robust and flexible.

Based on feedback from testing the prototype, we will also address any flaws and gaps in the initial design. Our goal is to create a detailed design that meets the operational needs of The Relaxing Koala while following best practices in software architecture for scalability, maintainability, and user-friendliness.

This document will show the changes we made, explain our design choices, and highlight the collaborative and dynamic aspects of the system. Our approach ensures the RIS is practical and ready for real world use, setting the stage for a smooth transition from design to deployment.

## Detailed Design

### Final Class Diagram

The class diagram now represents a structured and comprehensive view of the system's architecture, focusing on the separation of concerns between data management (Models) and business logic (Managers). It also introduces integration points for external systems such as payment gateways and inventory suppliers.

#### 1. Classes and Relationships:

##### a. Model Classes:

- i. *Reservation*
- ii. *Order*
- iii. *Menu*
- iv. *Table*
- v. *Supplier*
- vi. *Payment*
- vii. *Inventory*
- viii. *User(Security)*

##### b. Manager Classes:

- i. *ReservationManager*
- ii. *MenuManager*
- iii. *TableManager*
- iv. *UserManager(Security)*
- v. *PaymentManager*
- vi. *OrderManager*
- vii. *InventoryManager*
- viii. *SupplierCoordinator*

#### 2. Core Classes:

- a. **DatabaseManager** for database interactions, serving as a dependency for all Model classes.

#### 3. Collaborations:

- a. **ReservationManager** uses **TableManager** to check and reserve table availability.
- b. **OrderManager** interacts with **InventoryManager** to update stock based on current orders and notifies **KitchenDisplaySystem** to update order status in the kitchen.

## Justification for Changes

#### 1. Refinements:

- a. Enhanced separation of concerns: Each managerial class now handles specific business logic operations, which simplifies maintenance and enables more focused unit testing.
- b. Improved data integrity: By separating data models and business logic, the system ensures that business operations do not directly manipulate the database, reducing the risk of data corruption.

#### 2. Additions:

- a. **SupplierCoordinator**: Manages all interactions with suppliers, improving inventory accuracy and responsiveness to stock changes.
  - b. **UserManager**: Handles user authentication and authorization, essential for supporting multi user roles and secure access.
3. **Removals:**
- a. Redundant classes handling payment directly within the **Order** or **Menu** classes have been removed to centralize payment processing within **PaymentManager**, enhancing security and compliance with payment standards.
  - b. Classes such as Staff, **StaffScheduler**, **FeedBackHandler**, **ReportHandler**, and Invoice have been removed. Additionally, redundant payment handling previously distributed across **Order** or **Menu** classes has been centralized within **PaymentManager**, streamlining payment processes and bolstering security and compliance with payment standards.

## Responsibilities and Collaborations

1. **Model Classes:**
- a. Each model class, like **Reservation**, **Order**, **Menu**, **Table**, **Supplier**, **Payment**, **Inventory** encapsulates data relevant to its domain, handling creation, retrieval, update, and deletion operations through **DatabaseManager**.
2. **Manager Classes:**
- a. **ReservationManager**: Manages booking logic, including availability checks, reservation scheduling, and modifications.
  - b. **OrderManager**: Processes orders, from creation to completion, coordinating with **InventoryManager** and **KitchenDisplaySystem**.
3. **Collaborations:**
- a. **PaymentManager** collaborates with external payment services to handle transactions and interfaces with **OrderManager** to ensure accurate payment status updates.

## Dynamic Aspects

1. **Bootstrap Process:**
- a. Initialization includes setting up **DatabaseManager** to connect to the database and load initial data.
  - b. Subsequent loading of **UserManager** for handling user sessions and **TableManager** for preparing initial table setup for the day's operations.
2. **Interaction Scenarios:**
- a. **Scenario for Making a Reservation:**
    - i. A customer accesses the system via **CustomerInterface**, chooses a date and time, and requests a table.
    - ii. **ReservationManager** queries **TableManager** to find available tables and books accordingly, then sends a confirmation back to the customer.

**b. Scenario for Ordering Food:**

- i. Customer places an order through `CustomerInterface`.
- ii. `OrderManager` receives the order, verifies item availability with `InventoryManager`, and sends order details to the kitchen via `KitchenDisplaySystem`.
- iii. Once prepared, the order status is updated, and `PaymentManager` completes the transaction.

## Implementation

### Coding Standards and Practices

To ensure our code is maintainable, readable, and follows industry standards, we adopted the PEP 8 style guide for Python. Automated tools like Black for code formatting were used to enforce these standards across the codebase. We used Git for version control, following the Git Flow branching model to manage features, releases, and hotfixes effectively. Additionally, object-oriented programming (OOP) principles were applied to ensure a modular and scalable code structure.

### System Architecture and APIs

#### API Layer

The API layer, structured under `api/api_v1`, is a critical component for decoupling the frontend from the backend, enhancing flexibility, scalability, and maintainability. This layer handles requests from user interfaces and communicates with backend managers to perform operations.

#### API Endpoints

- **Inventory (inventory.py):** Manages inventory related operations by interacting with **InventoryController** to handle stock levels, updates, and reordering.
- **Menu (menu.py):** Provides functionalities to fetch and update menu items via **MenuManager**, ensuring the menu displayed to customers is always current.
- **Order (order.py):** Handles order creation, updates, and status tracking through **OrderManager**, ensuring efficient order processing and coordination with the kitchen.
- **Payment (payment.py):** Processes payments securely using **PaymentManager**, supporting multiple payment methods and ensuring compliance with financial regulations.
- **Reservation (reservation.py):** Manages reservations by interacting with **ReservationManager** and **TableManager** to check availability and confirm bookings by sending email.
- **Security (security.py):** Handles authentication and authorization by leveraging **UserManager** to secure user access.
- **Supplier (supplier.py):** Coordinates supplier-related operations through **SupplierCoordinator**, managing supplier relationships and deliveries.

- **Table (table.py):** Manages table assignments and availability via **TableManager**, optimizing seating arrangements.

## Core Components

- **Configuration (config.py):** Manages system-wide settings and configurations, ensuring consistency and flexibility in deployment environments.
- **Security (security.py):** Implements security protocols and encryption methods to protect user data and system integrity.

## Database Management

- Database Initialization and Sessions (**db/init\_db.py, db/session.py**): Initializes the database and manages sessions using SQLAlchemy, ensuring efficient and secure data handling.
- Database Models (models): Define the schema and relationships for **Inventory, Menu, Order, Payment, Reservation, Security, Supplier, and Table**.

## Manager

Each manager class encapsulates business logic and interacts with the database models:

- **InventoryManager:** Manages stock levels, updates inventory, and forecasts needs based on historical data and current trends.
- **MenuManager:** Handles menu item creation, updates, and deletions, ensuring the menu is always accurate and up-to-date.
- **OrderManager:** Processes and tracks orders, updates statuses, and ensures smooth coordination with the kitchen.
- **PaymentManager:** Processes payments, handles refunds, and ensures secure financial transactions.
- **ReservationManager:** Manages reservations, checks table availability, and confirms bookings.
- **SupplierCoordinator:** Coordinates supplier deliveries and manages supply chain logistics.
- **TableManager:** Manages table assignments and availability, optimizing seating arrangements.
- **UserManager:** Handles user authentication, registration, and role management.

## User Interfaces

The user interfaces are designed to be intuitive, responsive, and user-friendly, providing seamless interactions for both customers and staff:

- **customer-auth.html:** Manages customer login and registration, interacting with UserManager to authenticate users and maintain session security.
- **customermenu.html:** Displays the current menu and allows customers to place orders, fetching menu data via MenuManager.



- **inventory.html**: Used by staff to manage inventory levels, providing real-time updates from InventoryManager.
- **order.html**: Enables staff to manage customer orders, updating statuses and details through OrderManager.
- **reservation.html**: Allows customers to make reservations and staff to manage them, checking table availability via ReservationManager and TableManager.
- **staffmenu.html**: Allows staff to update menu items, reflecting changes immediately through MenuManager.
- **table.html**: Manages table layouts and assignments, providing real-time updates and optimizations via TableManager.

## Advanced Design Patterns

### *Factory Method Pattern*

The Factory Method pattern is employed in creating various manager classes (e.g., MenuManager, OrderManager, ReservationManager), allowing for the creation of objects that share a common interface but require different instantiation specifics. This design enhances modularity and scalability by encapsulating object creation logic.

### *Singleton Pattern*

The DatabaseManager class is implemented as a Singleton, ensuring that only one instance manages database operations. This approach maintains data integrity and consistency across the system by providing a single point of interaction for database-related activities.

### *Strategy Pattern*

The Strategy pattern is used in the PaymentProcessor to handle multiple payment methods (example: credit card, online wallet, cash) by defining a common payment interface. Similarly, InventoryController employs different strategies for inventory reordering, such as just-in-time and economic order quantity, to adapt to various business needs.

### *Model-View-Controller (MVC) Pattern*

The MVC pattern divides the application into three main components: Model, View, and Controller. This separation of concerns helps manage complexity by isolating the user interface (View), data (Model), and business logic (Controller).

- **Model**: Managed by DatabaseManager and various data models (example: **User**, **MenuUpdateManager** ).
- **View**: Consists of HTML pages like **staffmenu.html** and **order.html** that provide user interfaces for User to place a new order and MenuUpdateManager to create a new menu Item
- **Controller**: Includes **MenuUpdateManager** to handle data flow and business logic.

### *Command Pattern*

The Command pattern encapsulates requests as objects, allowing for parameterization of clients with different requests, and supports queuing and logging operations. This pattern is implemented in **OrderManager** for handling diverse order-related operations, **KitchenDisplaySystem** for managing order displays, and **PaymentProcessor** for different payment methods. To enhance scalability and reliability, RabbitMQ is integrated for queuing command requests, facilitating better load management and fault tolerance.

## Execution and Operation

### Scenario 1: Making a Reservation

Steps:

1. Starting State:
  - a. The reservation page is empty.

### **Reservation Management**

#### **Create Reservation**

Username:

Password:

Login

#### **Reservations**

2. Inputting Data:

- a. Customer log in to their accounts for reservation.

## Reservation Management

### Create Reservation

Username:

Password:

Login successful!

Table:

Time:

### Reservations

3. Validation of Incorrect Input:
- a. Customer tries to book a reservation without selecting a time.

## Reservation Management

### Create Reservation


Username:

Password:

Login successful!

Table:

Time:

 Please fill in this field.

### Reservations

4. Correcting Input:

- a. Customer selects the correct time and resubmits.

## Reservation Management

### Create Reservation

Username:

Password:

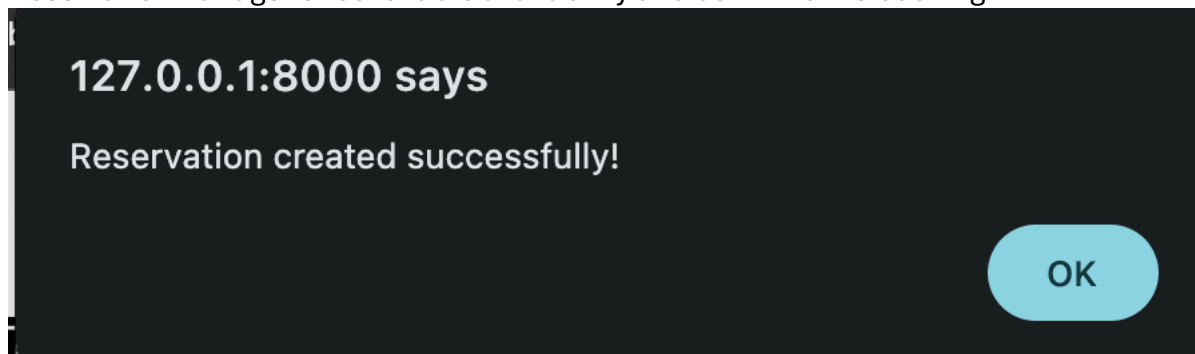
Login successful!

Table:

Time:

### Reservations

5. Submission and Confirmation:
  - a. ReservationManager checks table availability and confirms the booking.



# Reservations

User: test - Table ID: 1 - Time: 10/06/2024, 19:00:00 Edit Delete



closer2713@gmail.com  
to luannn010 ▾  
...

28 May 2024, 22:50 (2 days ago)



## Reservation Details

Your reservation details are as follows:

- Reservation ID: 123456
- Date: 2024-06-10
- Time: 19:00

If you have any questions or need to make changes to your reservation, please contact us.

Thank you for choosing our services!

Best regards,  
The Reservation Team

## Scenario 2: Updating the Menu

Steps:

1. Starting State:
  - a. The menu management page is empty.

## Menu For Staff

### Create Menu Item

Name:

Description:

Price:

Availability:

Inventory Item:

Create

### Menu Items

2. Inputting Data:

- a. Staff adds new dishes to the menu.

## Menu For Staff

### Create Menu Item

Name:

Description:

Price:

Availability:

Inventory Item:

### Menu Items

3. Validation of Incorrect Input:
- a. Staff tries to add a dish without specifying a price.


## Menu For Staff

### Create Menu Item

Name:

Description:

Price:

 Please fill in this field.

Inventory Item:

### Menu Items

4. Correcting Input:

- a. Staff specifies the price and resubmits.

## Menu For Staff

### Create Menu Item

Name:

Description:

Price:

Availability:

Inventory Item:

### Menu Items

5. Submission and Confirmation:

- a. MenuUpdateManager updates the menu.

## Menu For Staff

### Create Menu Item

Name:

Description:

Price:

Availability:

Inventory Item:

### Menu Items

Chicken - Chicken - \$10 - Available

## Scenario 3: Placing an Order for Dine-in

Steps:

1. Starting State:

- a. The order page is displayed, showing the available tables and menu items.

## Order Management

### Create Order

Order Type:

Dine In

Table:

✓ Table 1 - Reserved

Table 2 - Available

Table 3 - Available

Table 4 - Available

Table 5 - Available

Table 6 - Available

Table 7 - Available

Table 8 - Available

Table 9 - Available

Table 10 - Available

Create and Pay

## Orders

2. Inputting Data:
  - a. Customer selects the dine-in option.
  - b. Customer chooses a table from the available options.



- c. Customer selects items from the menu.

## Order Management

### Create Order

Order Type:

Dine In

Table:

Table 1 - Reserved

### Order Items

Chicken - Chicken - Available - \$10.00	2	-
Chicken - Chicken - Available - \$10.00	2	-

Add Item

**Total Amount: \$40.00**

VISA 4242 4242 4242 4242

01 / 27 123 12345

Create and Pay

## Orders

3. Submission and Confirmation:
- OrderManager processes the order and sends details to the KitchenDisplaySystem. Inventory for Chicken also going down from 100 Unit.

Orders

Table: 1 - Total Amount: \$40.00	
o	Item: Chicken - Quantity: 2
o	Item: Chicken - Quantity: 2

Inventory Management

Add Inventory Item

Item Name:

Quantity:

Measurement Unit:

Expiry Day:  ☐

Add Inventory

Inventory Items

- Item Chicken - Quantity: 96 - Unit: Unit - Expiry: 2024-06-09 

EditDelete

Scenario 4: Placing an Order for Online

Steps:

- Starting State:

- a. The online order page is displayed with a login option.

## Order Management

### Create Order

Order Type:

Online

### Login

Username:


Password:

Login

### Order Items

Add Item

**Total Amount: \$0.00**

 Card number

MM / YY CVC

Create and Pay

## Orders

2. Inputting Data:

- a. Customer selects items from the menu.

## Order Management

### Create Order

Order Type:

Online

### Login

Username:

Password:

Login

### Order Items

Chicken - Chicken - Available - \$10.00	2	-
Chicken - Chicken - Available - \$10.00	2	-

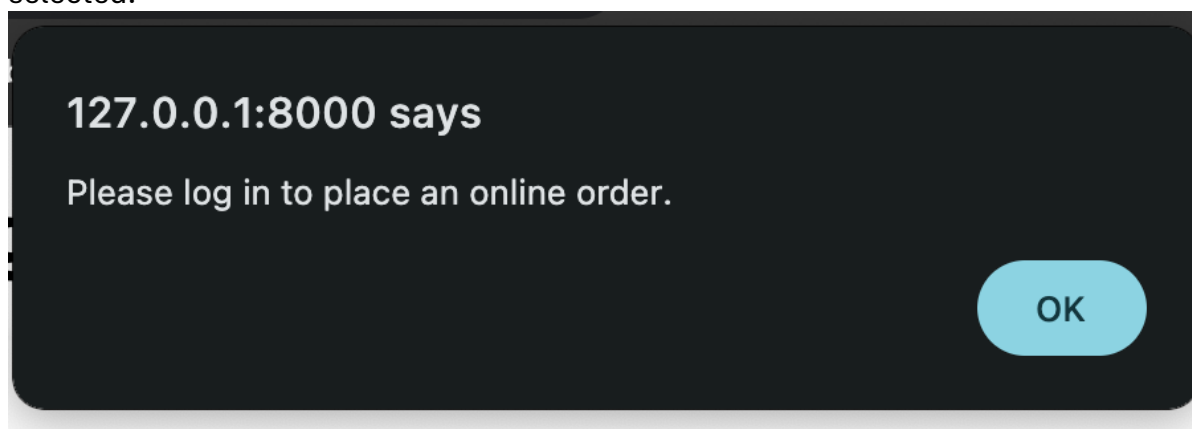
Add Item

**Total Amount: \$40.00**

VISA 4242 4242 4242 4242 01 / 26 123 12345

Create and Pay

3. Validation of Incorrect Input:
- Customer tries to submit the order without log in.
  - Error message is displayed indicating that at least one item must be selected.



4. Correcting Input:

- a. Customer log in resubmits.

## Order Management

### Create Order

Order Type:

Online

### Login

Username:

test

Password:

....

Login

Login successful!

### Order Items

Chicken - Chicken - Available - \$10.00	2	-
Chicken - Chicken - Available - \$10.00	2	-

Add Item

**Total Amount: \$40.00**

VISA 4242 4242 4242 4242

01 / 26 123 12345

Create and Pay

5. Submission and Confirmation:
- a. OrderManager processes the order and sends details to the KitchenDisplaySystem and Inventory going from 96 to 92.

## Inventory Items

- Item Chicken - Quantity: 92 - Unit: Unit - Expiry: 2024-06-09 Edit Delete

## Orders

Table: 1 - Total Amount: \$40.00

- Item: Chicken - Quantity: 2
- Item: Chicken - Quantity: 2

User: test - Total Amount: \$40.00

- Item: Chicken - Quantity: 2
- Item: Chicken - Quantity: 2

## Scenario 5: User Authentication

Steps:

- Starting State:
  - The login/registration page is empty.

## Customer Signup

### Sign Up

Username:

Email:

Password:

Sign Up

## Customer Login

### Login

Username:

Password:

Login

- Inputting Data:

- a. User enters registration details.

## Customer Signup

### Sign Up

Username:

Email:

Password:

3. Validation of Incorrect Input:

- a. User tries to register without filling in all required fields.


## Customer Signup

### Sign Up

Username:

Email:

Password:

 Please fill in this field.

4. Correcting Input:

- a. User fills in all required fields and resubmits.

## Customer Signup

### Sign Up

Username:

Email:

Password:

127.0.0.1:8000 says

Signup successful

5. Submission and Confirmation:
  - a. UserManager creates the account and logs the user in.

## Customer Login

### Login

Username:

Password:

127.0.0.1:8000 says

Login successful



## Discussion of Assignment 2 Design

Assignment 2 laid the foundation for the Restaurant Information System (RIS) by providing an initial object-oriented design aimed at improving operational efficiency and customer service at The Relaxing Koala café/restaurant. Here, we reflect on the quality of the original design, identifying both its strengths and areas for improvement.

### Strengths of the Original Design

- **Clear Objective and Scope:**
  - The original design effectively addressed the primary goal of enhancing operational efficiency and customer satisfaction. It identified key functionalities such as reservations, order management, kitchen operations, and payment processing.
- **Comprehensive Class Definitions:**
  - The initial design included a well defined set of candidate classes, each with specific responsibilities. Classes like OrderManager, ReservationManager, and InventoryController were crucial in modularizing the system's functionalities.
- **Design Patterns:**
  - The application of design patterns, such as Factory Method, Singleton, Strategy, and MVC, demonstrated a solid understanding of best practices in software design. These patterns were chosen to promote flexibility, scalability, and maintainability.
- **Use of UML Diagrams and CRC Cards:**
  - The inclusion of UML diagrams and CRC cards provided a clear visualization of the system's architecture and class interactions. This helped in understanding the overall structure and the flow of operations within the system.

### Weaknesses and Areas for Improvement

- **Limited Dynamic Aspect Coverage:**
  - While the static aspects of the design were well documented, the dynamic behaviors and interactions between objects needed further elaboration. More detailed sequence diagrams or state diagrams could have provided better insights into runtime operations.
- **Scalability Concerns:**
  - The initial design did not fully address potential scalability issues, especially concerning high-volume traffic and concurrent user interactions. The performance under load needed more thorough consideration and stress testing plans.
- **Security Aspects:**
  - Security considerations were relatively basic. Given the sensitivity of handling customer data and payments, more robust security measures, such as encryption and secure authentication mechanisms, should have been detailed.

- **Error Handling and Resilience:**
  - The design lacked comprehensive error handling and resilience strategies. In real-world applications, the ability to gracefully handle errors and maintain operations under various failure conditions is crucial.
- **Integration with External Systems:**
  - The integration points with external systems like payment gateways, supplier systems were not deeply explored. Detailed strategies for managing these integrations would enhance the robustness of the system.

## Advanced Security Measures

- **Data Encryption:** Encrypted sensitive data, including **user passwords** and **payment information**, using industry standard algorithms. Regular security audits were conducted to ensure compliance with best practices and to identify potential vulnerabilities.
- **Stripe Integration for Payments:** Integrated Stripe for payment processing, providing a secure and reliable solution for handling transactions. This integration includes robust error handling and ensures compliance with PCI DSS standards.

## Improved Error Handling

Comprehensive error-handling mechanisms were integrated throughout the system to ensure it can gracefully handle unexpected conditions:

- **Custom Exception Handling:** Defined custom exception classes to manage different types of errors effectively. This provides meaningful error messages and helps maintain system stability.
- **Logging and Monitoring:** Implemented extensive logging and monitoring to track errors and system performance. This allows for quick identification and resolution of issues, improving overall system reliability.

## Detailed Integration Strategies

Clear and robust strategies for integrating with external systems were developed to enhance the reliability and robustness of the RIS. This included:

- **API Specifications:** Detailed **API** specifications were created to ensure smooth integration with external systems, such as payment gateways (Stripe) and supplier systems. These specifications define the expected inputs, outputs, and behaviors for each API endpoint.
- **Thorough Integration Testing:** Conducted comprehensive integration testing to ensure that the interactions with external systems are reliable and handle various edge cases effectively. This testing helps to identify and resolve potential issues before they impact the live system.

## Lessons Learned

From the improvements made in Assignment 3, several key lessons were learned:

### 1. Importance of Detailed Dynamic Modeling:

- a. Detailed dynamic models, including sequence and state diagrams, are crucial for understanding the interactions and flow of operations within the system. They help identify potential issues early in the design phase and provide a clearer roadmap for implementation.
- 2. Scalability Requires Early Consideration:**
  - a. Scalability should be a primary concern from the beginning. Early stress testing and performance optimization can save significant effort later in the development process. Implementing load balancing and caching mechanisms early on ensures that the system can handle increased loads effectively.
- 3. Security Cannot Be Overlooked:**
  - a. Security must be integrated into every aspect of the design. From secure coding practices to regular audits, protecting user data and ensuring secure transactions is paramount. The integration of Stripe for payment processing is a key example of prioritizing security.
- 4. Comprehensive Error Handling is Essential:**
  - a. Robust error handling ensures that the system can maintain operations even under unexpected conditions. Planning for various failure scenarios and implementing custom exception handling, logging, and monitoring can significantly improve the system's resilience.
- 5. Thorough Integration Testing:**
  - a. Detailed planning and testing of integrations with external systems are essential. This ensures that these integrations work seamlessly and can handle various edge cases and failure conditions. Having clear API specifications and robust timeout and retry mechanisms are critical for reliable integrations.

## Conclusion

In conclusion, the refined Restaurant Information System (RIS) for The Relaxing Koala successfully addresses the operational needs of the expanding café and restaurant. Building on the foundation laid in Assignment 2, Assignment 3 provided a detailed and practical design ready for implementation. Key improvements include enhanced dynamic modeling, scalability measures, advanced security features, and robust error handling. By integrating best practices in software design and thoroughly testing the system, we ensured that the RIS is both robust and flexible. This project highlights the importance of iterative design and continuous improvement in developing effective technological solutions for complex operational challenges. The RIS is now poised to support The Relaxing Koala's growth, improving efficiency and customer satisfaction.

## References

- Assignment 1 - Requirements Specification | Assignment1.docx
- Assignment 2 - Object Design | Assignment2.docx