**CS673 Software Engineering** 

**Team X - Project Name**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Jialin Li | QA Leader | *Jialin Li* | 10/18/2023 |
| Zhuoru Song | Requirement Leader | *Zhuoru Song* | 10/18/2023 |
| Chandrasiri S.A. | Security Leader | *Chandrasiri* | 10/18/2023 |
| Siyuan Xue | Team Leader | *Siyuan Xue* | 10/18/2023 |
| Xiankun Niu | Design and Implementation Leader | *Xiankun Niu* | 10/18/2023 |
| Qian Xiang | Configuration Leader | *Qian Xiang* | 10/18/2023 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.4d34og8)

[References](#_heading=h.2s8eyo1)

[Glossary](#_heading=h.17dp8vu)

# 

# Introduction

The "Online Restaurant Ordering System" is a comprehensive digital solution designed to facilitate seamless interaction between restaurants and their customers. This system offers a platform where diners can view the menu, select dishes, and proceed to checkout with minimal effort. To ensure organized structure and functionality, the system is divided into four distinct modules. Although currently only the 'User Model' is elaborated, each module is intended to cater to specific roles within the restaurant, and they will eventually collaborate to ensure a smooth end-to-end user experience.

All modules share a centralized database, ensuring data consistency and real-time updates. This robust connectivity allows for the integration of diverse functions, ranging from menu display to user authentication.

User Model:

Homepage: Acts as the central hub, enabling users to navigate to and from every other page with ease.

Cartpage: Not only showcases the selected dishes but also automatically calculates quantities. It provides a direct link to the confirmation page for order finalization.

Menupage: Displays a comprehensive list of dishes, allowing users to explore options and add desired items to the shopping cart.

Register Page: A gateway for new users to join, it captures personal details and includes an email verification step to ensure genuine sign-ups.

Loginpage: Existing users can access the platform through this page, leading them directly to the homepage upon successful authentication.

Confirmation Page: This versatile page functions as a checkpoint before final order submission. Its adaptability is evident, as future iterations might incorporate multiple return pages to accommodate varying scenarios.

Kitchen Model:

Currently not elaborated.

Waiter Model:

Currently not elaborated.

Management Model:

Currently not elaborated.

The design goals for this software system include ensuring user-friendliness, maintaining data integrity, and allowing for future scalability. This document offers a broad overview of the system's components and structure, laying the groundwork for subsequent development phases.

# Software Architecture

## System Architecture

## 

As you can see, this is the software system architecture of our "Online Restaurant Ordering System".

## System Decomposition:

The "Online Restaurant Ordering System" is decomposed into the following main components:

### User Management Component

This component handles user(including Customer, Server, and Manager) registration, login, profile, and access management and request management.

### Order Management Component

This component handles order create, add, delete, finish operations.

### Kitchen Management Component

This component handles dish creation, finish, update operations and ingredients short alert, update operation.

### Table Management Component

This component allows automatically allocating and releasing the tables by the QRcode function.

### Review Management Component

This component handles the feedback and reviews of the dishes and services.

## **Interface & Dependencies**:

* The interface of each component is dependent on the Flask and React Bootstrap framework.
* User Management component fetch and post the order details to Order Management and Review Management component.
* Kitchen management component fetch the order details from Order Management component and send message to the User Management component.
* Table Management component automatically updates the table information.

## Framework and Tech stack:

* **Framework:** Flask
* **Frontend:** React
* **Backend:** Python
* **Database:** SQLite

# Class Diagram

Our web application is intended to face both the customer and the staff team; thus, the system is more than just a menu list. Below are our class diagram with UML class definition.

# 

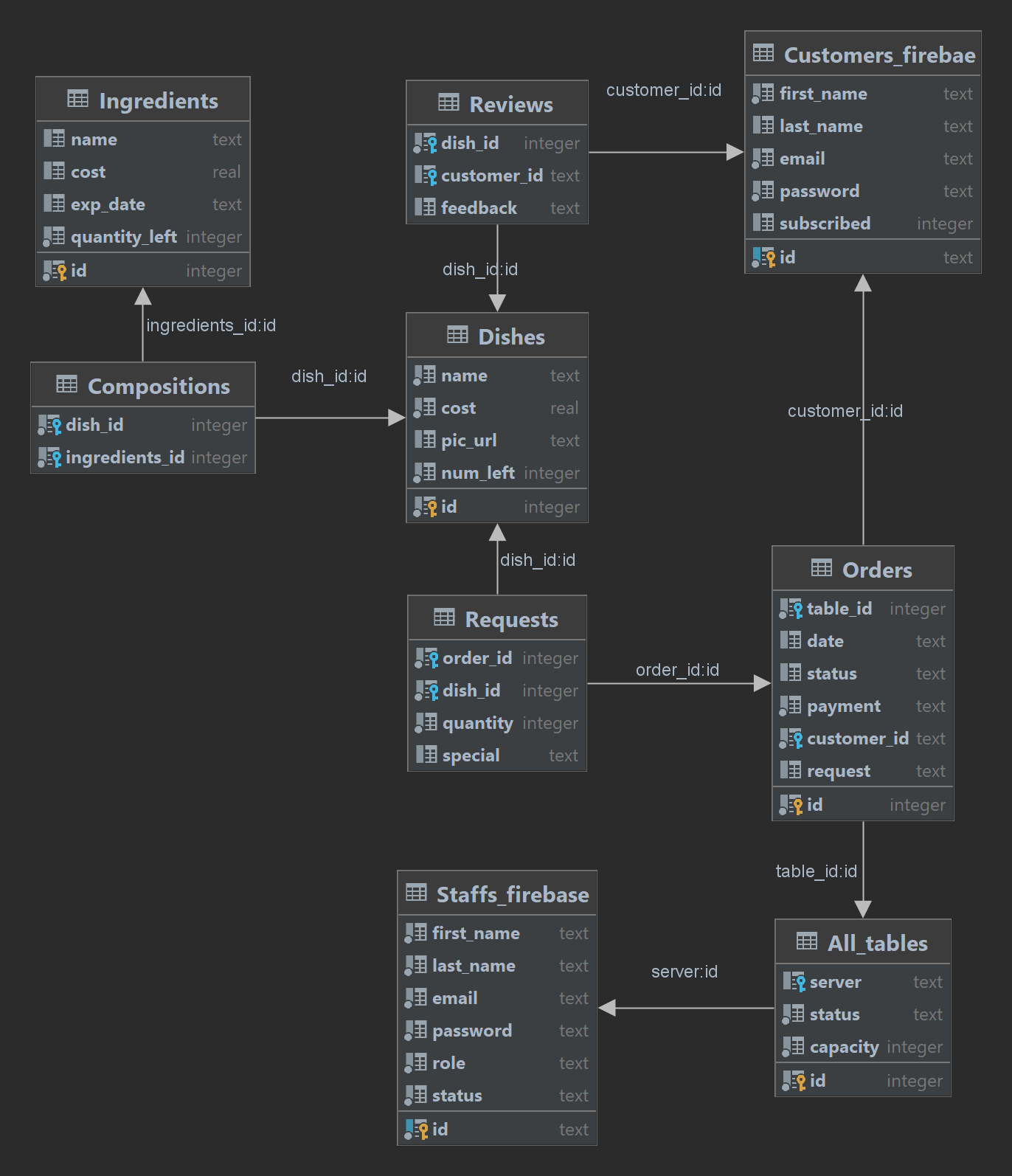
# UI Design (if applicable)

In this section, you can describe your UI design

# 

# Database Design

Our database strives to be as close to the third normal form as possible, while considering the practical application and day-to-day operations, which often consider the second normal form to be good enough (C.H., 2023). Note that the customer and staff data involve passwords and personal data, thus we decided to use firebase as our identity provider. Below is the schema:



# **Security Design**

* The restaurant ordering system prioritizes the security of customer data and operational processes.
* While online payments are not involved, data security, authentication, and authorization measures are implemented to protect against potential threats.

**Authentication and Authorization:**

1. **Customer Authentication:**
   * Customers are required to create accounts through email sign-up. The process involves the following steps:
     1. ***Registration***: Customers provide their email addresses and create a password during the registration process.
     2. ***Email Verification*:** An email verification link is sent to the customer's provided email address. Customers must click this link to verify their email and activate their accounts.
     3. ***Password Storage*:** Customer passwords are securely stored in the system using cryptographic hashing.
2. **Staff Authentication:** 
   * Staff members are authenticated with unique credentials, including user roles and permissions.
3. **Authorization**:
   * Role-based access control (RBAC) is used to determine access rights for different system functionalities.
   * The system ensures that customers can only access their own data and order information.
4. **Password Policy**:
   * Enforce strong password policies for customer and staff accounts.
   * Encourage regular password changes.

**Data Security**

* + Customer profiles, order histories, and related data are stored in a secure database.
  + Data at rest (e.g., customer profiles and order history) is encrypted to prevent unauthorized access.
  + Access controls are implemented to ensure that only authorized personnel can view or modify customer data.
  + Use encryption to secure data in transit (e.g., HTTPS)

**Data Privacy and Compliance:**

* + Comply with data protection regulations such as GDPR or CCPA, if applicable.
  + Define data retention policies and obtain explicit consent when necessary.

**Secure User Input Handling:**

* + In order to prevent common web application vulnerabilities(e.g., SQL injection).
    1. Using an ORM framework
    2. Validating and sanitizing user inputs. Reject any input that doesn't adhere to expected patterns by considering the following concepts.
       1. Data Type Validation
       2. Length Validation
       3. Format Validation
       4. Whitelist Validation
       5. HTML Encoding - with Flask we can use the markupsafe library to perform HTML encoding.

**Session Management:**

* + Secure session handling to prevent session-related security issues.
  + Use random, long-lived session tokens, and implement timeout mechanisms.

**Logging and Monitoring**

* + Comprehensive logging captures system activities and user actions.
  + Monitoring systems are in place to detect and alert administrators to suspicious or unauthorized activities.

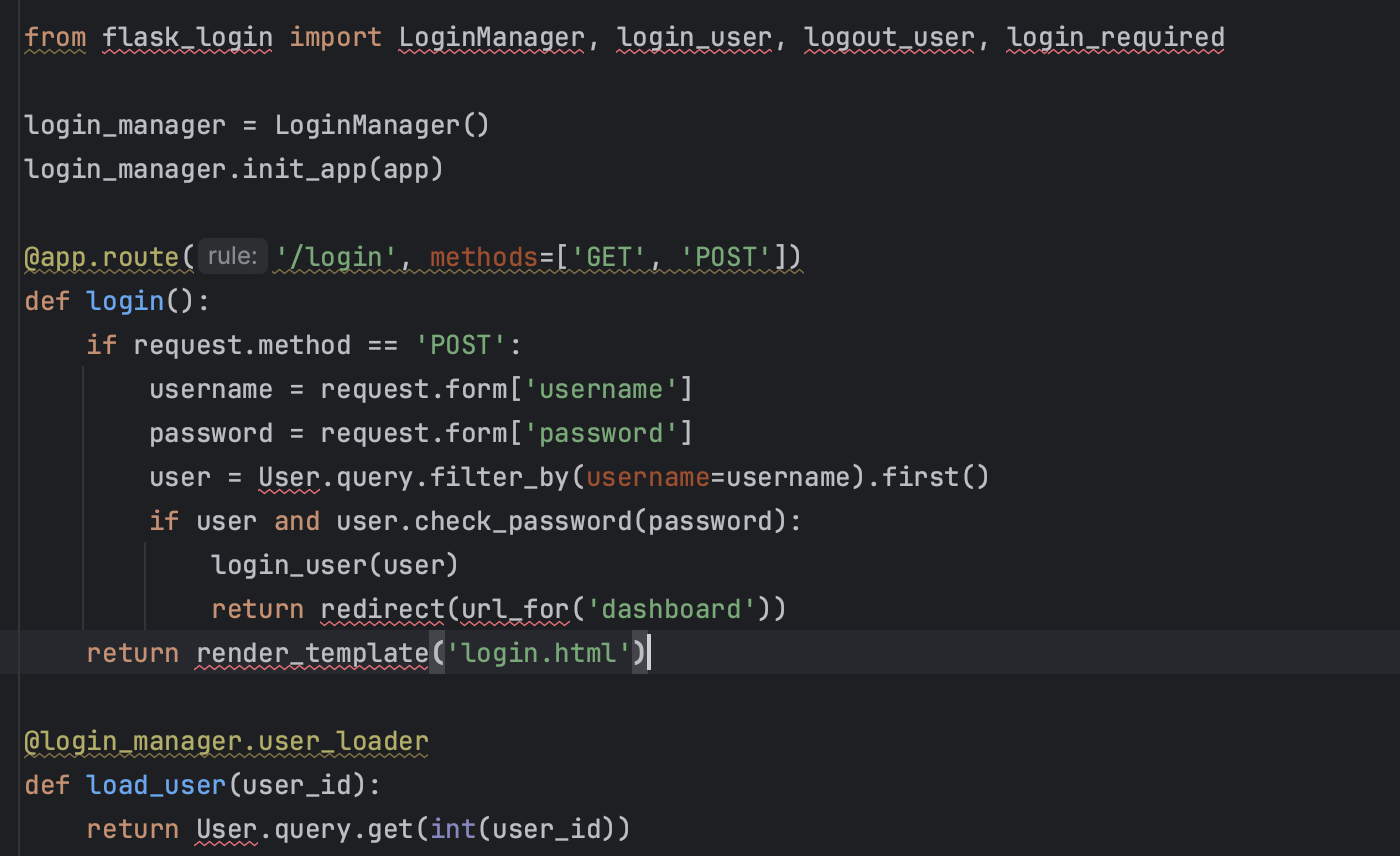
**Backup and Disaster Recovery:**

* + Regularly back up the system's data and have a disaster recovery plan in place to ensure data availability and system continuity.

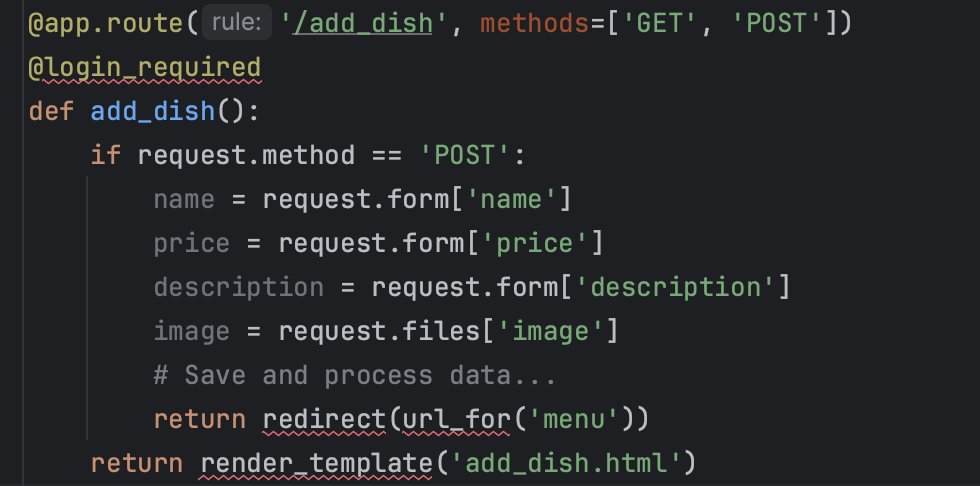
# Business Logic and/or Key Algorithms

### User Authentication

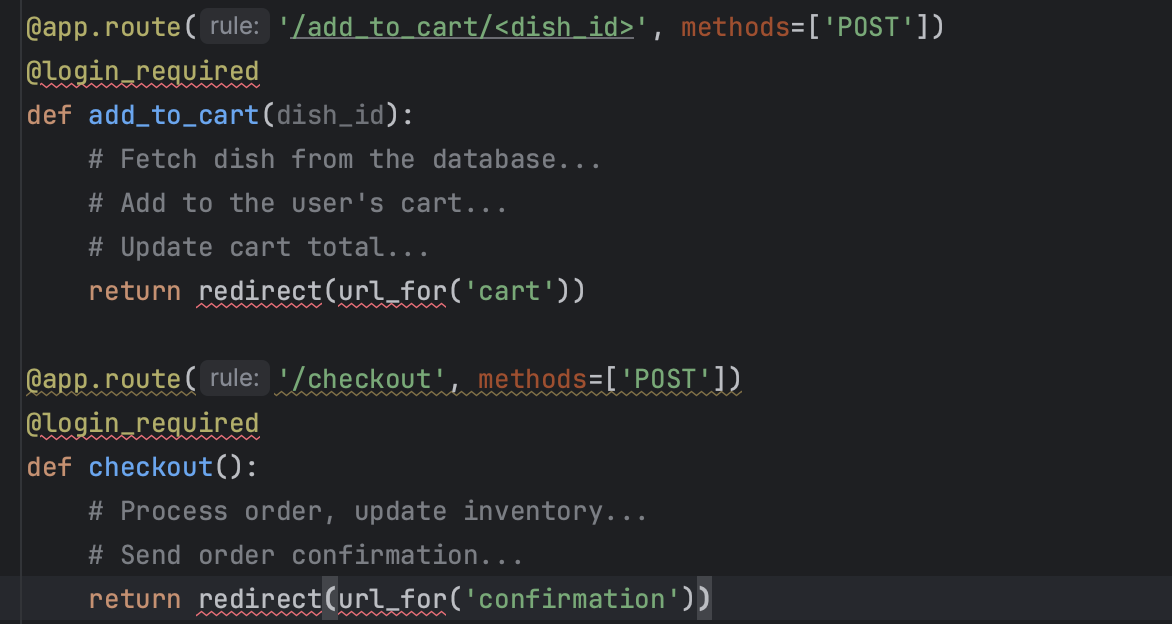
Using Flask-Login for user authentication:



### Menu Management



### Order Handling



# Design Patterns

**Observer pattern**

Observer pattern: used in one-to-many dependency between objects

Observer pattern was used in the design of dishes. When the details or availability of a dishes changed, any other place such as manual and cart should also respond and make changes.

**MVC Pattern**

MVC Pattern: Separates an application into three interconnected components: Model (data and business logic), View (user interface), and Controller

MVC Pattern was used in the entire project since it is a web application.

Model: Data, database operations, and the rules that govern how the data can be manipulated

View: The user interface (UI) of the application. The web page.

Controller: The logical code to receive user input and processes it into the database.

# Any Additional Topics you would like to include.

# 

# References

Flask Documentation: Flask's official documentation is a good starting point and covers all the basic Flask features. Link: <https://flask.palletsprojects.com/en/2.0.x/>

C, H. (2023, March 2). *Understanding the basics of database normalization*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/03/understanding-the-basics-of-database-normalization/

# Glossary

* Flask: A miniature web framework written in Python.
* Route: In Flask, a route is a mapping of urls to Python functions.
* HTTP Methods: Such as GET, POST, PUT, and DELETE, define how the client interacts with the server.