COMPUTING WITH PLYMOUTH UNIVERSITY

School of Computing and Mathematics

PUSL3119
Computing Individual Project

BSc (Hons) Computer Science

Rathnayake M. Rathnayake

Food Desire: An Online Food Ordering System

2022/2023

Food Desire: An Online Food Ordering System.

Haritha Chamara Rathnayake.
10747887@students.plymouth.ac.uk
University of Plymouth

Supervised By
Mr. Pramudya Thilakaratne.
Pramudya.h@nsbm.ac.lk

Abstract

The food domain is constantly evolving with the introduction of new technologies and innovations. The "Food Desire" project aims to contribute to this progress by developing an online food ordering system that allows customers to order food online and customize their meals according to their preferences. Additionally, the project intends to implement a recommendation system that suggests meal items to users based on their preferences and previous orders. The system also includes an inventory management system for restaurants interested in the project, which allows them to manage ingredients, recipes, orders, and deliveries. Restaurant owners can also use the system to manage their employees. The document outlines the requirements, design, implementation, testing, integration and deployment, and maintenance of the system. It also discusses the challenges faced during development and the future for the project.

KEYWORDS: Foods, Food ordering, Restaurant, Recommendation system, Inventory management system, Employee management system, C#, .NET, ML.NET, ASP.NET Core, Blazor, WinUi3, Windows App SDK.

Contents

Table Of Figures

INTRODUCTION

The purpose of this project was to develop an online food ordering system called “Food Desire” that allows customers to order food from a restaurant and customize their meals by adjusting the amount of ingredients in each meal item. This feature enhances the customer experience by giving them more control and flexibility over their orders. In addition, a recommendation system was implemented to suggest meal items to users based on their preferences and previous orders, using a collaborative filtering machine learning algorithm. This system improves the user experience by providing personalized and relevant suggestions, while also benefiting the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue. To support this feature, an inventory management system was also developed for the restaurant to manage ingredients and recipes, allowing them to keep track of available ingredients and update the list of ingredients for each meal item accordingly. This report presents the results of the project, including the design, implementation, testing, and evaluation of the Food Desire online food ordering system and its associated inventory management system.

Chapter 1
BACKGROUND, OBJECTIVES & DELIVERABLES.

1.1 Introduction

In this chapter, an overview of the background, objectives, and deliverables of the Food Desire online food ordering system project is provided. The aim of the project was to develop a user-friendly online food ordering system for a single restaurant that allows customers to customize their meals and receive personalized recommendations. An inventory management system was also developed for the restaurant to manage ingredients and recipes. The context and motivation behind the project are discussed, its main objectives and goals are outlined, and the key deliverables that were produced are described.

1.2 Background

Online food ordering has become increasingly popular in recent years, offering customers a convenient and efficient way to order food from various restaurants. However, many online food ordering systems do not allow customers to customize their meals according to their preferences, dietary needs, or allergies. This can limit customer satisfaction and loyalty, as well as reduce sales opportunities for restaurants that offer such options.

1.3 Objectives

The main objective of this project was to develop an online food ordering system called “Food Desire” for a single restaurant that allows customers to customize their meals by adjusting the amount of ingredients in each meal item. This feature enhances the customer experience by giving them more control and flexibility over their orders. In addition, a recommendation system was implemented to suggest meal items to users based on their preferences and previous orders, using a content-based filtering machine learning algorithm. To support this feature, an inventory management system was also developed for the restaurant to manage ingredients and recipes.

1.4 Deliverables

The key deliverables of this project included:

- A user-friendly web-based interface for customers to browse and order food from the restaurant.

- A customization feature that allows customers to adjust the amount of ingredients in each meal item.
- A recommendation system that suggests meal items to users based on their preferences and previous orders.
- An inventory management system for the restaurant to manage ingredients and recipes.
- A comprehensive final report detailing the design, implementation, testing, and evaluation of the Food Desire online food ordering system and its associated inventory management system.

1.5 Summary

In summary, this chapter provided an overview of the background, objectives, and deliverables of the Food Desire online food ordering system project. The project aimed to develop a user-friendly online food ordering system for a restaurant that allows customers to customize their meals and receive personalized recommendations. An inventory management system was also developed for the restaurant to manage ingredients and recipes. The key deliverables of the project included a web-based interface for customers, a customization feature, a recommendation system, an inventory management system, and a final report.

Chapter 2
LITERATURE REVIEW.

2.1 Introduction

The rise of technology has brought about many changes in the way businesses operate, particularly in the food industry. One such change is the adoption of self-ordering and online food ordering systems, which allow customers to place orders for food via the internet or other computerized systems. This literature review will explore various aspects of these systems, including their benefits, challenges, and potential for improvement.

2.2 Self-Ordering Systems

Self-ordering systems refer to systems that allow customers to place orders for goods or services by themselves, without the need for interaction with a staff member. These systems can be either computerized or manual and can take various forms, including telephone, text message, email, or internet-based ordering. Research has shown that self-ordering systems can provide numerous benefits to businesses, including increased efficiency, reduced waiting times, and increased demand (Ratto *et al.*, 2008). In the food industry, self-ordering systems have been found to be particularly valuable due to their ability to personalize orders based on customer preferences (Fujita, Shimada and Sato, 2014).

2.3 Online Food Ordering

Online food ordering refers specifically to the use of internet-based systems to place orders for food. These systems have become increasingly popular in recent years due to their convenience and ease of use (R. *et al.*, 2017). Research has shown that customers value factors such as website quality, speed of response, and service quality when choosing an online food service (Daim *et al.*, 2013). However, there is also room for improvement in these systems. For example, some researchers have suggested that online food ordering platforms could be more human-centered and provide greater freedom for customers to customize their orders (Goffe *et al.*, 2021).

2.4 AI and Machine Learning in E-Commerce

The use of AI and machine learning techniques has become increasingly common in e-commerce, including in the food industry. One common application of these technologies is in recommendation systems, which use machine learning algorithms to predict what customers may want to buy or eat based on their past behavior (Ahrens, 2012). These systems have been

found to be effective at increasing customer engagement and sales (Sarwar *et al.*, 2001). Item-based collaborative filtering algorithms have been shown to provide better performance than user-based algorithms (Sarwar *et al.*, 2001).

2.5 Unsupervised machine learning

Unsupervised machine learning is a type of machine learning in which models are trained using unlabeled datasets and are allowed to act on that data without any supervision. These algorithms discover hidden patterns or data groupings without the need for human intervention ((n.d.)., no date). Unsupervised learning is often used for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition ((n.d.)., no date).

Common unsupervised learning approaches include clustering, association, and dimensionality reduction. Clustering is a data mining technique that groups unlabeled data based on their similarities or differences. Clustering algorithms can be categorized into several types, including exclusive, overlapping, hierarchical, and probabilistic ((n.d.)., no date).

Unsupervised learning cannot be directly applied to a regression or classification problem because, unlike supervised learning, we have the input data but no corresponding output data ((n.d.)., no date). However, unsupervised learning can still be useful in many applications. For example, it can be used to explore and cluster data, generate new data, or analyze existing data.

2.6 Summary

The literature review discusses self-ordering and online food ordering systems, their benefits, challenges, and potential for improvement. It also covers the use of AI and machine learning techniques in e-commerce, particularly in recommendation systems.

Chapter 3
REQUIREMENTS GATHERING.

3.1 Introduction

Requirements gathering was a crucial step in the online food ordering system project. It involved communicating with stakeholders to identify their needs, expectations, and constraints for the system. The goal was to create a clear and comprehensive requirements specification document that defined the scope, objectives, functionality, quality attributes, and acceptance criteria of the system.

3.2 Functional Requirements

The functional requirements for the online food ordering system included features such as browsing through different food categories and items; customizing orders by changing the amount of ingredients; seeing the price of orders based on customization; placing orders online and paying with various methods; tracking order status and delivery time; rating and reviewing orders after delivery; viewing order history and recommendations.

The inventory management system had features such as registering new employees, suppliers, chefs, and deliverers; assigning roles and permissions to different employees; allowing suppliers to supply ingredients to the restaurant based on demand; enabling chefs to create recipes for different food items using ingredients; updating recipes based on customer feedback or availability of ingredients; allowing deliverers to take out orders for delivery based on location and priority; enabling admins to pay for maintenance, salaries, taxes etc.

The recommendation system had features such as collecting data from customer orders, ratings, reviews, preferences etc.; analyzing data using machine learning algorithms to find patterns and trends; suggesting food items for customers based on their most consumed food and individual ingredients; considering factors such as seasonality, availability, popularity, nutrition etc.; displaying suggestions on the website's home page.

3.3 Non-Functional Requirements

The non-functional requirements for the online food ordering system included having a user-friendly interface with clear buttons, colors, fonts etc.; supporting English language; having high performance by loading fast and responding quickly; handling concurrent requests; having high security by encrypting sensitive data such as payment information.

The inventory management system had a reliable database that stored all information related to inventory, employees, suppliers etc.; had a backup mechanism that saved data regularly in case of failure or data loss.

The recommendation system had a scalable architecture that allowed it to handle increasing numbers of customers, orders, data etc.; had an adaptive algorithm that updated itself based on new data or feedback.

3.4 Use Case Diagram

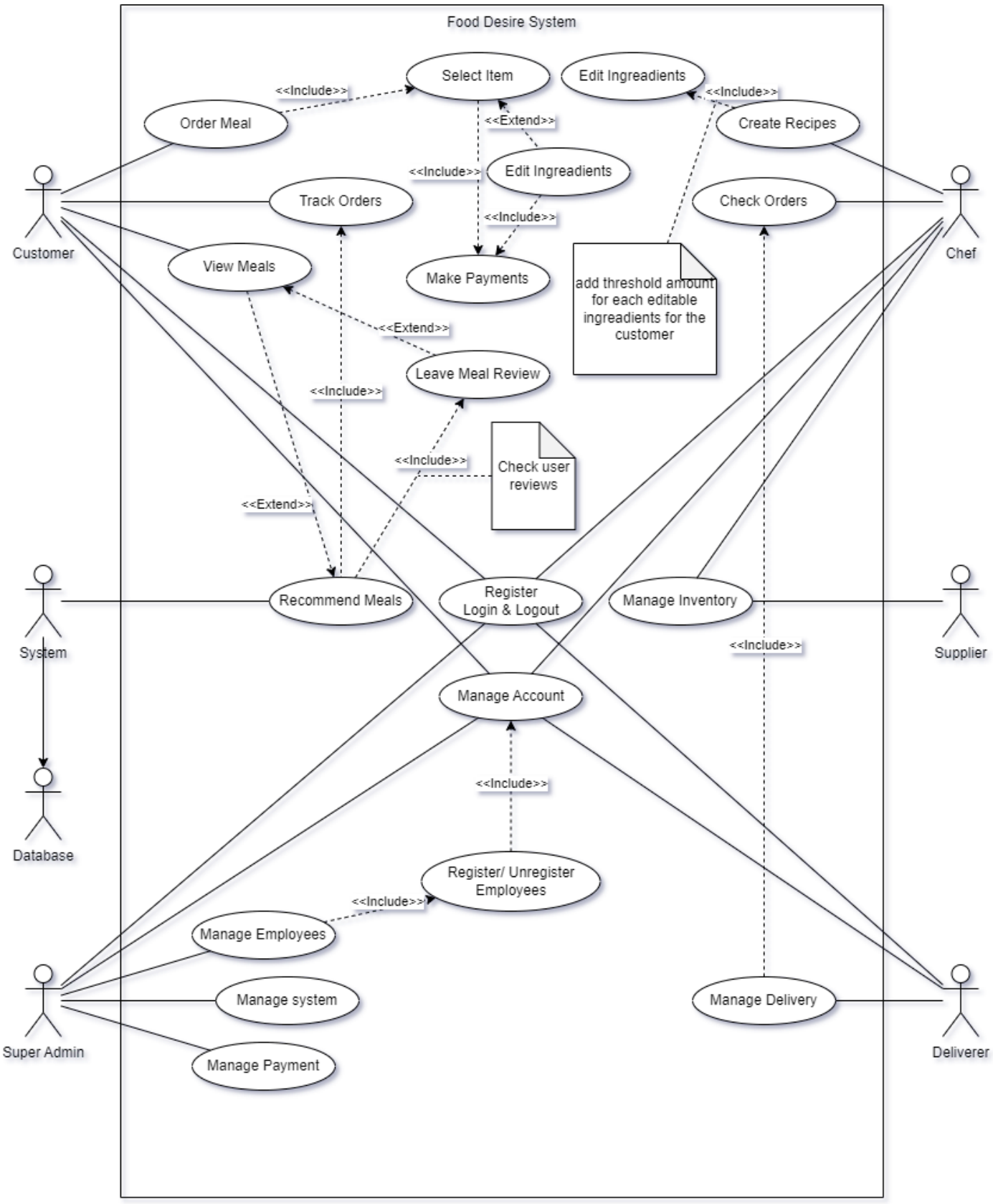


Figure 3. 1 Use case diagram.

The use case diagram in Figure 3.1 illustrates the interactions between different actors and the Food Desire system. The diagram shows how customers can browse and order food, customize

their meals, and receive personalized recommendations. It also shows how restaurant staff can manage inventory and process orders.

3.5 Hardware / Software Requirements

The hardware and software requirements for the Food Desire system include the following:

3.5.1 Hardware Requirements

The Food Desire system can run on any device that supports a web browser, such as desktops, laptops, tablets, or smartphones.

3.5.2 Software Requirements

To use the Food Desire system, the device must meet some minimum specifications to run .NET 7.

3.6 Network Requirements

The Food Desire system should have a reliable and secure network connection to communicate with the WinUI 3 desktop application, the Web API, and the Blazor web UI.

3.7 Summary

In summary, this chapter outlined the functional and non-functional requirements for the online food ordering system. The functional requirements included features such as browsing through different food categories and items; customizing orders by changing the amount of ingredients; seeing the price of orders based on customization; placing orders online and paying with various methods; tracking order status and delivery time; rating and reviewing orders after delivery; viewing order history and recommendations. The non-functional requirements included having a user-friendly interface with clear buttons, colors, fonts etc.; supporting English language; having high performance by loading fast and responding quickly; handling concurrent requests; having high security by encrypting sensitive data such as payment information.

Chapter 4
METHOD OF APPROACH

4.1 Introduction

The project is a distributed system consisting of multiple projects, including an inventory management system and a web application. To ensure efficient development, it is important to adhere to good design practices. The web application relies heavily on the inventory management system, as it renders content based on the data provided by the inventory management system. Both systems utilize shared libraries to minimize code duplication. The project follows sound application architecture principles to achieve a robust and well-designed system.

4.2 Design

This distributed system involves multiple actors, each with unique types of interactions. In Figure 3.1, the customer and system actors interact through the web application, while the remaining actors participate in the inventory management system. As a result, the system is divided into two separate projects.

The restaurant actor can produce food items by creating recipes, as depicted in Figure 3.1. These food items will be rendered in the web application for the users to see and interact with.

Furthermore, the system can also manage employees, as shown in Figure 3.1. However, this management functionality is only a small portion of restaurant management interactions and is included within the inventory management system. It should be noted that the inventory management system does not encompass the complete range of restaurant management tasks. Instead, it focuses on managing actors directly involved in inventory-related activities, such as chefs who create recipe items, delivery persons, and supplier persons.

4.2.1 Database

To implement the backend of the distributed system, a Microsoft SQL Server relational database is utilized as the database system. To configure and interact with the database system, the Entity Framework, which is a Microsoft .NET Framework data-access platform, is being

used.

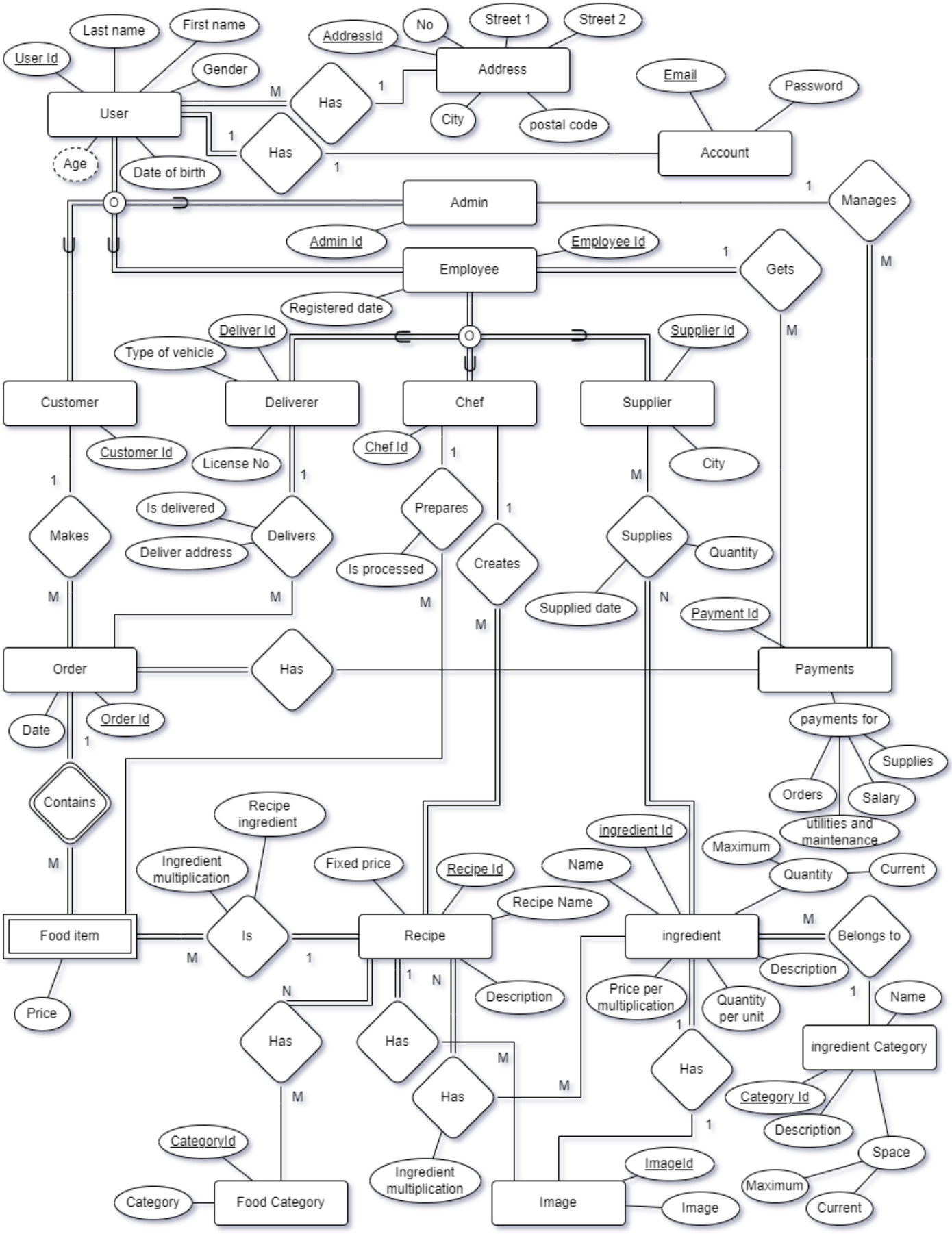


Figure 4. 1 ER Diagram.

Figure 4.1 presents the Entity-Relationship (ER) diagram for the system. Several assumptions have been made in constructing the diagram:

1. Every customer and employee are considered a user in the system.
2. Each user can have one account.
3. Multiple users can share the same address.
4. A user can have one default address, but a different address can be used when placing food orders.

Every delivery person, chef, and supplier are an employee.

5. All employees receive payment for their services.
6. Payment management is handled by the system administrator, who may also be the restaurant owner.
7. Food items are dependent on both customer orders and recipes.
8. Food items can have ingredient modifications, allowing customization of the original recipe.

Based on the ER diagram in Figure 4.1, a shared class library was deployed to be shared between the Inventory system and the web application. This shared class library includes all the entities depicted in Figure 4.1 as models. These models were utilized to generate the database schema using Entity Framework migrations.

Overall, the combination of Microsoft SQL Server as the relational database system, Entity Framework for database configuration, and the shared class library for modeling entities ensures a unified and consistent approach for data management in the distributed system.

4.2.1.1 Models

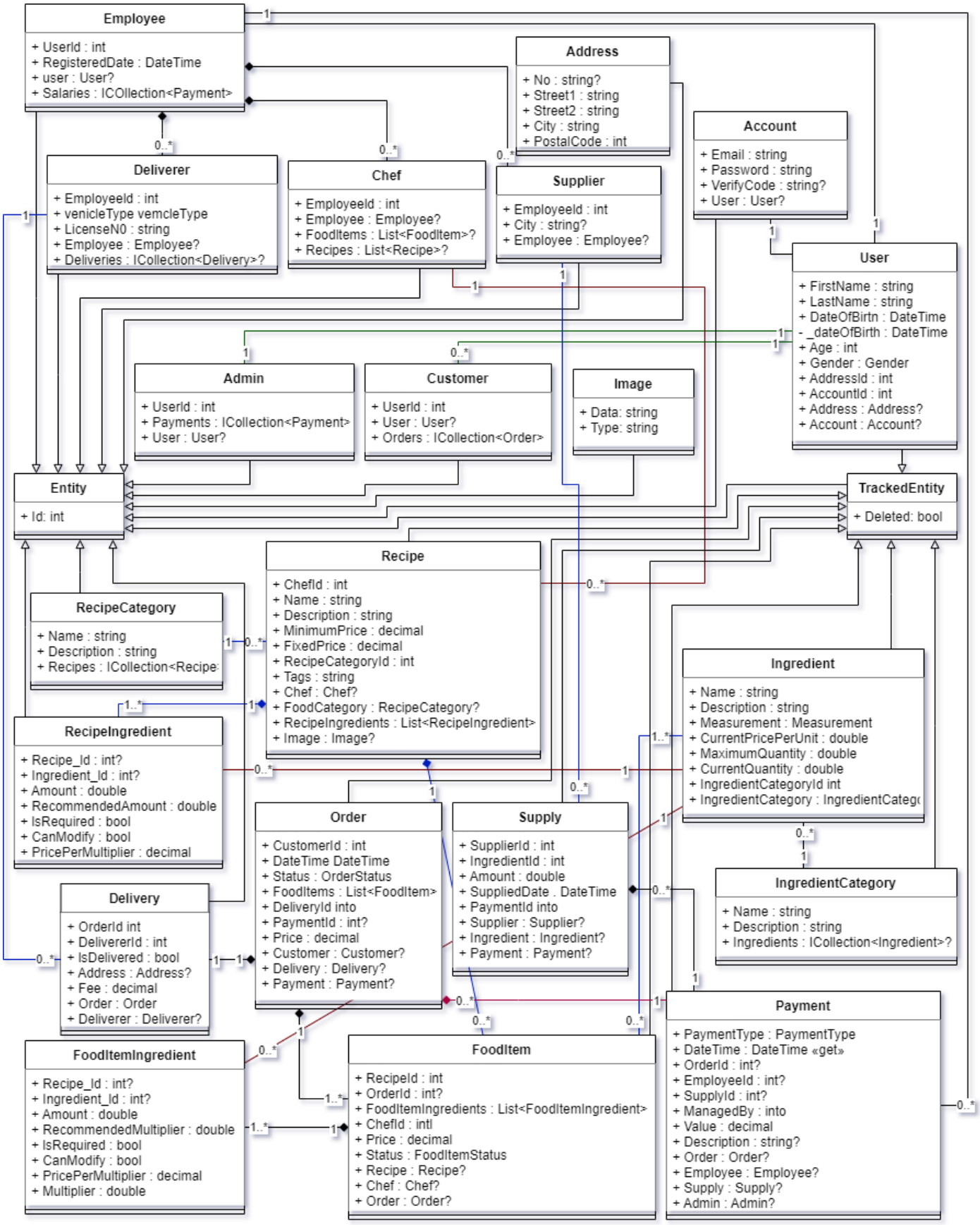


Figure 4. 2Models Class Diagram.

The provided class diagram represents a model for managing each entity depicted in Figure 4.2 within the system. This model is implemented using Entity Framework Core and serves as a class library that can be utilized throughout the entire system.

The Entity class acts as a base class for all entities in the project. It includes a single property called "Id," which serves as the primary key for each entity. Inheriting from this base class allows all entities in the project to access this property easily. This simplifies the implementation of the generic repository pattern, which will be further discussed in the document.

The TrackedEntity class is designed to enable soft delete functionality for any entity class that inherits from it. Soft delete is a pattern used to mark a record as deleted rather than physically removing it from the database. This approach allows deleted records to be easily restored if necessary and enables the tracking of when a record was deleted. The TrackedEntity class includes a Boolean property named "IsDeleted," which is set to true when the entity is deleted. This property can be utilized to filter out deleted records when querying the database. By utilizing the TrackedEntity class, the system can implement soft delete functionality across entities as needed.

Overall, the Entity class simplifies the implementation of the generic repository pattern, while the TrackedEntity class enables soft delete functionality. These classes provide essential tools for efficient and effective database operations within the project.

4.2.2 Data Access Layer (DAL)

After generating the database, a services layer was created to interact with it using Entity Framework which allows using C# OOP methods to handle database operations.

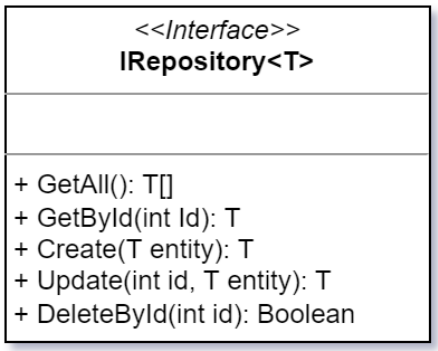


Figure 4. 4 Generic Type Repository

Figure 4. 3 Repository Usage

Repository pattern was used to handle database operations where any entities can implement this interface to do CRUD operations on their respective entity.

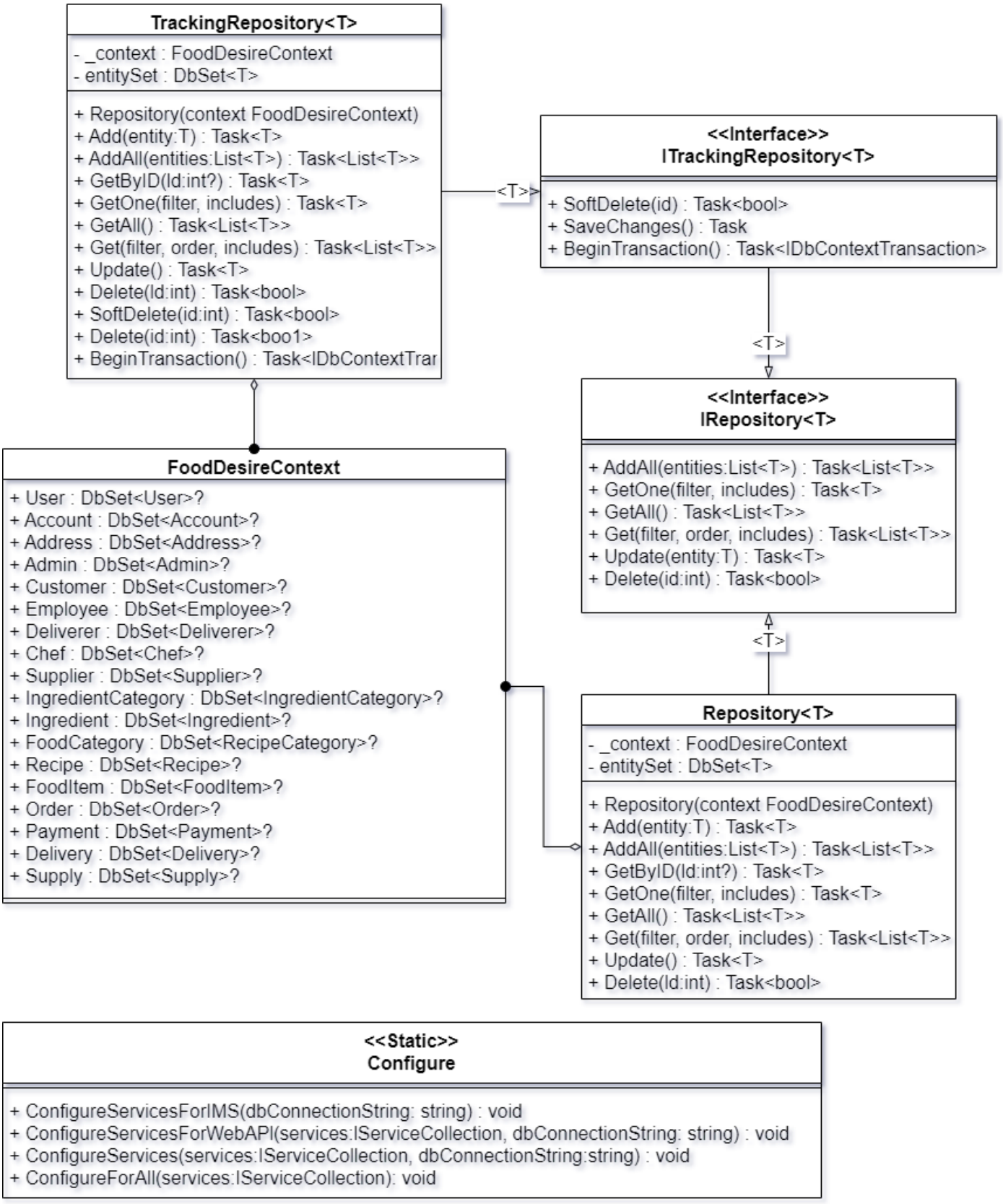


Figure 4. 5 DAL Class Diagram.

The provided class diagram (Figure 4.5) illustrates the structure of the data access layer (DAL) in a food ordering system. The DAL is responsible for managing the interaction between the application and the database. In this diagram, the FoodDesireContext class is specifically

designed for use with Entity Framework. It contains properties for each entity (such as User, Account, Address, Admin, etc.) that can be persisted in the database.

To achieve separation between the data access logic and the business logic of the application, the repository pattern is employed. The IRepository<T> interface serves as an abstraction layer between the data access logic and the business logic. It defines the contract for CRUD (Create, Read, Update, Delete) operations on the entities. The Repository<T> class implements this interface and provides the actual implementation of these operations. These operations are executed on an instance of the DbContext, which represents the database context.

In addition to the IRepository<T> interface, there is also the ITrackingRepository<T> interface. This interface extends IRepository<T> and provides additional methods for soft delete and transaction management. The TrackingRepository<T> class implements this interface and provides the concrete implementation of these extended operations.

The Configure class plays a crucial role in enabling dependency injection and configuring the required services for the application. By using dependency injection, the code achieves loose coupling and enhanced testability. The Configure class defines the mappings between interfaces and their corresponding implementations, allowing the application to configure dependencies at runtime.

4.2.3 Domain Service Layer (Core)

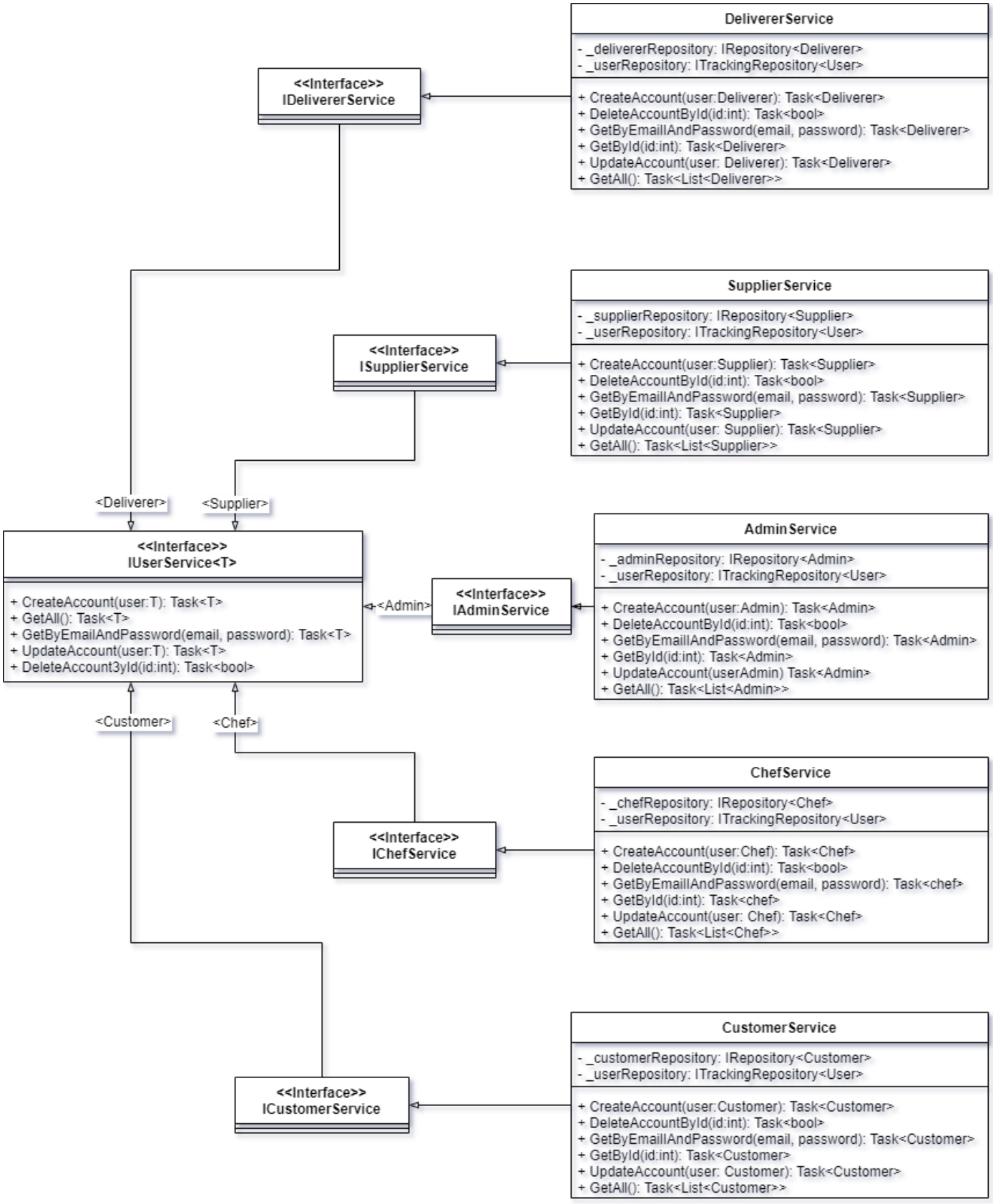


Figure 4. 6 User Services Diagram. (Inheritance only).

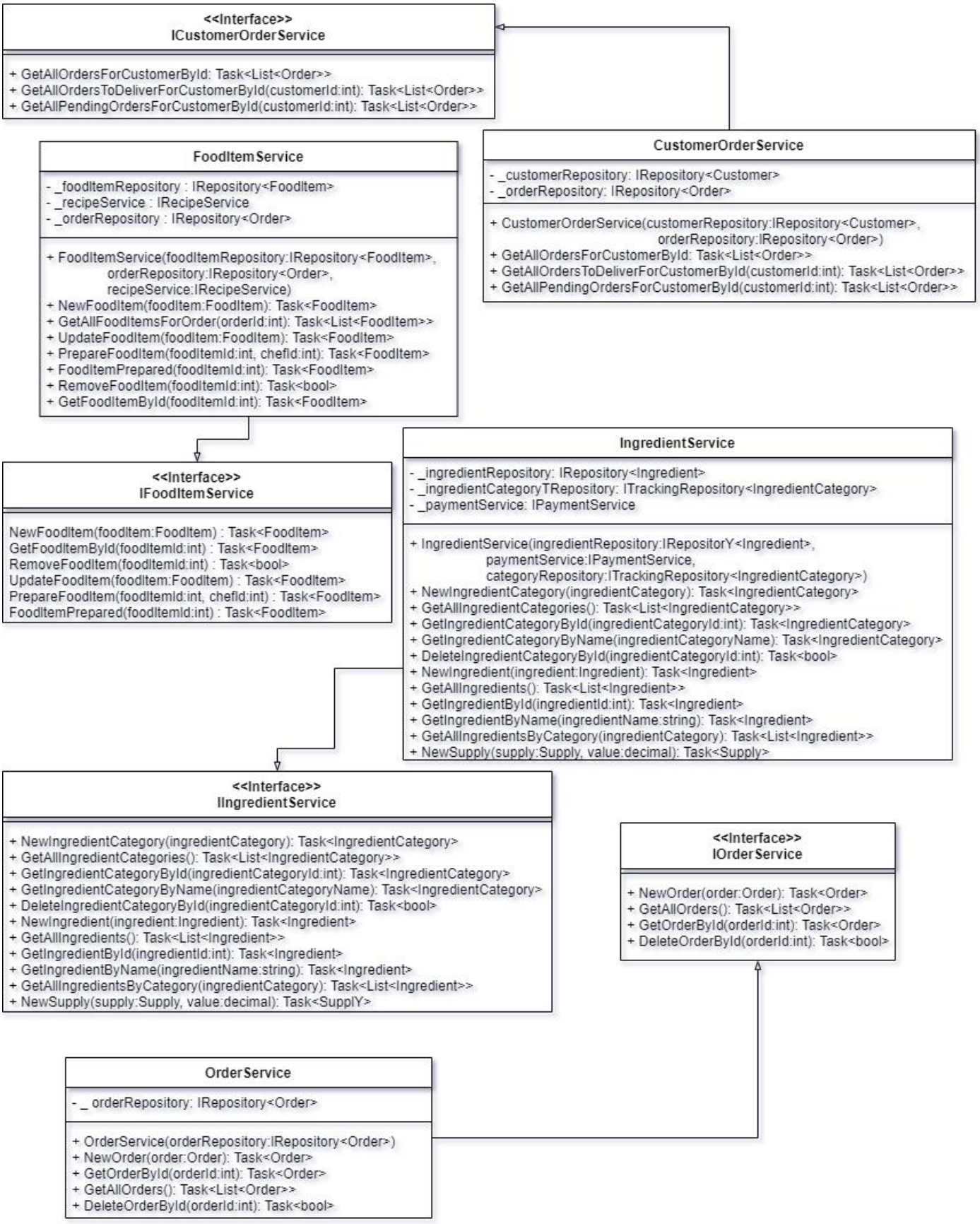


Figure 4. 7 Core Services 1. (Inheritance only).

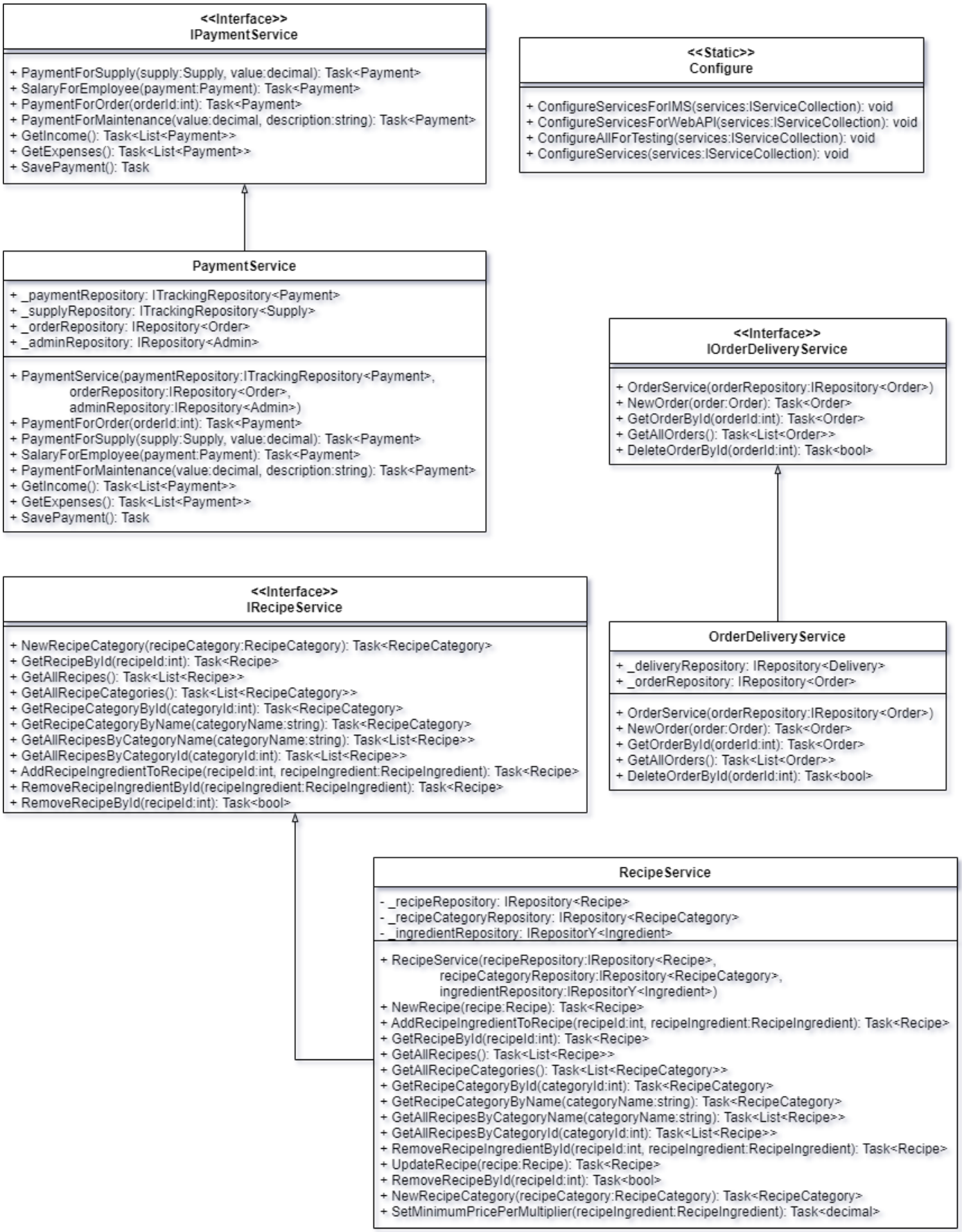


Figure 4. 8 Core Services 2. (Inheritance only).

In the provided class diagrams (Figures 4.6, 4.7, 4.8), various services and their corresponding interfaces are depicted. These services play crucial roles in the system, and each service implements a specific service interface, defining the contract for that service and facilitating loose coupling of code.

The followings are the associations and dependencies between the classes:

- AdminService: Depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Admin entity.
- ChefService: Depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Chef entity.
- CustomerService: Depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Customer entity.
- DelivererService: Depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Deliverer entity.
- SupplierService: Depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Supplier entity.
- FoodItemService: Depends on IRepository and IRecipeService interfaces and has a composition relationship with the FoodItem entity.
- CustomerOrderService: Depends on IRepository interface and has a composition relationship with the Customer and Order entities.
- IngredientService: Depends on IRepository, ITrackingRepository, and IPaymentService interfaces and has a composition relationship with the ingredient entity.
- OrderDeliveryService: Depends on IRepository interface and has a composition relationship with the Delivery entity.
- OrderService: Depends on IRepository interface and has a composition relationship with the Order entity.
- PaymentService: Depends on IRepository, ITrackingRepository, and IPaymentService interfaces and has a composition relationship with the Payment entity.
- RecipeService: Depends on IRepository interface and has a composition relationship with the Recipe, RecipeCategory, and Ingredient entities.

These associations and dependencies demonstrate how the services in the system rely on specific interfaces and entities to perform their tasks. By depending on these interfaces, the services can achieve loose coupling and easily adapt to changes in the underlying data storage or payment functionality.

Furthermore, the Configure class plays a vital role in setting up the dependency injection framework for the application. By utilizing dependency injection, the code achieves loose coupling between components, resulting in increased code reusability and maintainability. The

Configure class enables the mapping of interfaces to their respective implementations, ensuring that dependencies can be easily configured at runtime.

4.2.3 Inventory Management System

After designing the data access layer and services layer, these layers were used in other projects, such as the inventory management system. The inventory system was developed to include all the use cases of the actors depicted in Figure 3.1, namely the Admin, system user, and restaurant.

The main purpose of the inventory system is to manage food items for the web application and allow customers to customize their food items.

The inventory management system encompasses several major interactions, including:

1. Employee Registration: Employees, as depicted in Figure 4.2, are registered in this system.
2. Supply Details Management: The system facilitates the management of supply details.
3. Payment Management: Payment-related operations were handled by the system.
4. Food Item Recipe Creation: Chefs creates recipes for food items using the system.

The inventory system consisted of various views that fulfilled specific functionalities, including:

1. Home View: Displays a summary of the business, providing essential information.
2. Ingredient View: Manages ingredients in the inventory.
3. Supply View: Enables system users to manage supplies and place supply orders.
4. Recipe View: Allows chefs to create food items by formulating recipes.
5. Order View: Allows chefs to see ordered food items and prepare them.
6. Delivery View: Allows deliverers to see completed food items that needs to be delivered and manage them.
7. Employee View: Allows system users to manage employee-related information.
8. Settings View: Shows settings for the application.

It is important to note that the views are role-specific, meaning that certain views are hidden from specific roles. For instance, chefs do not have access to employee-related views as they are irrelevant to their responsibilities.

The Recipe View plays a crucial role within the system, as it serves as the interface through which the restaurant enables web users to customize their food items. Chefs creates recipes by utilizing ingredients available in the inventory. Additionally, chefs can set restrictions on the amount of adjustable ingredients to prevent customers from overusing or overdosing ingredients.

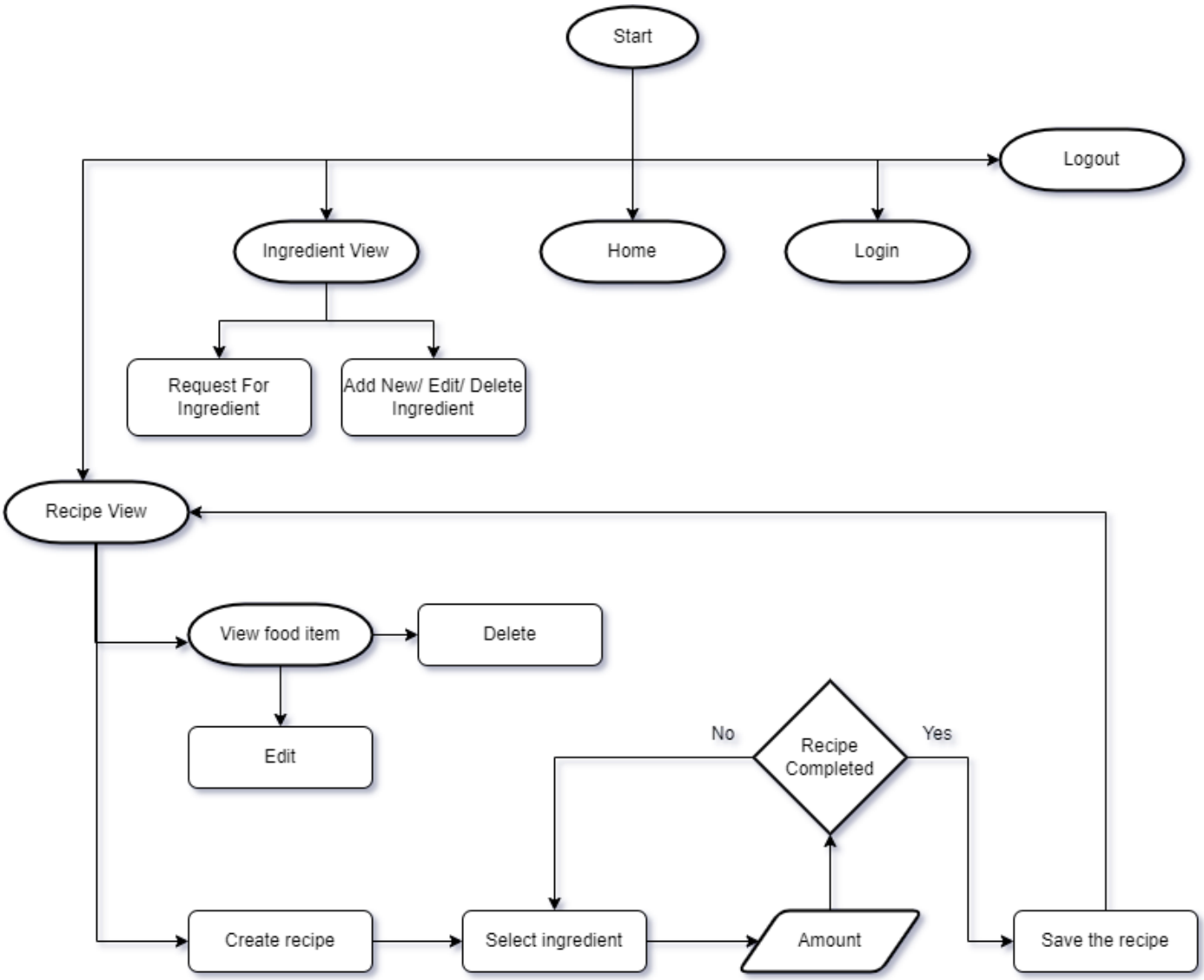


Figure 4. 9 Chef Activity Diagrams.

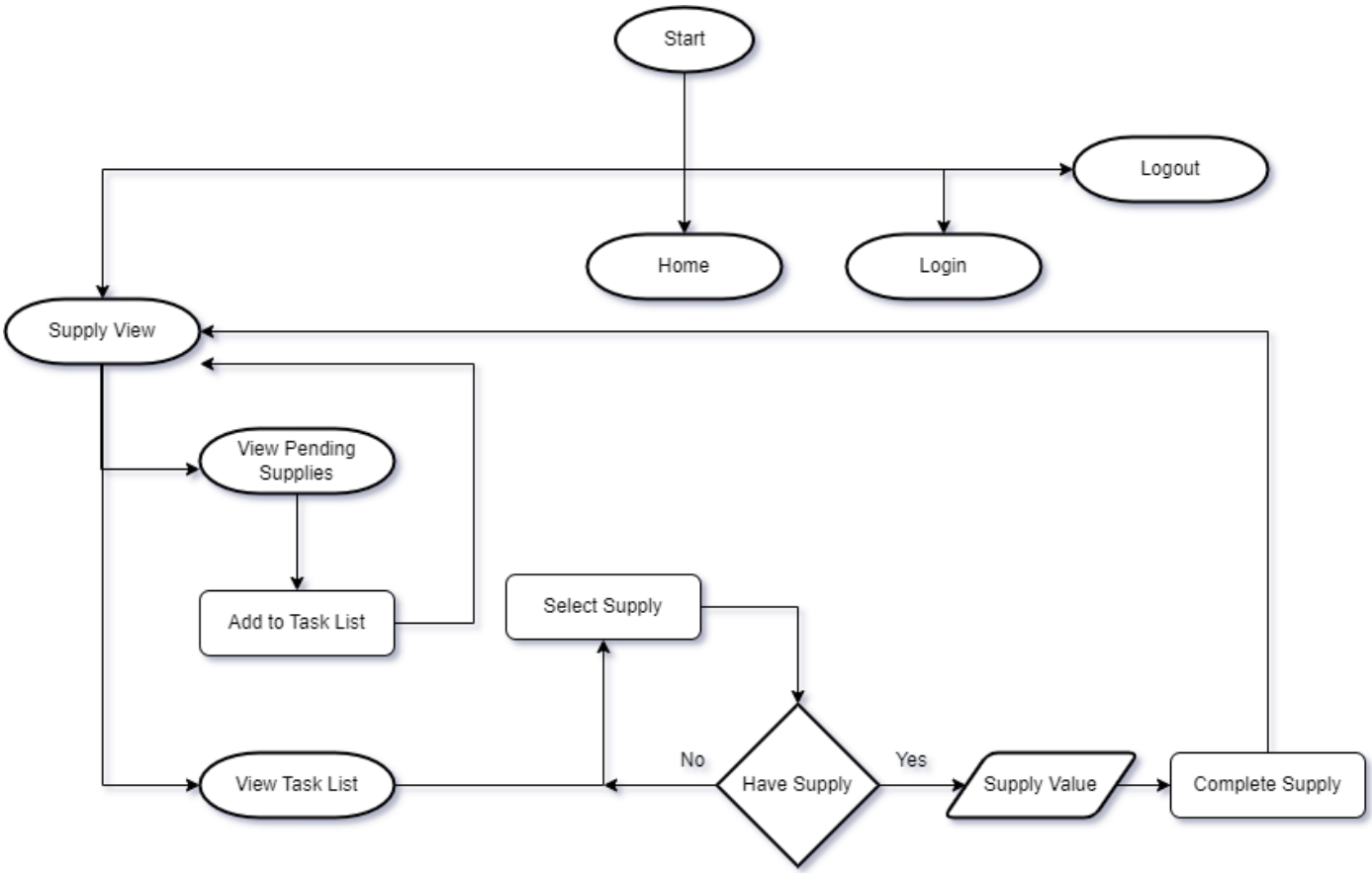


Figure 4. 10 Supplier Activity Diagram.

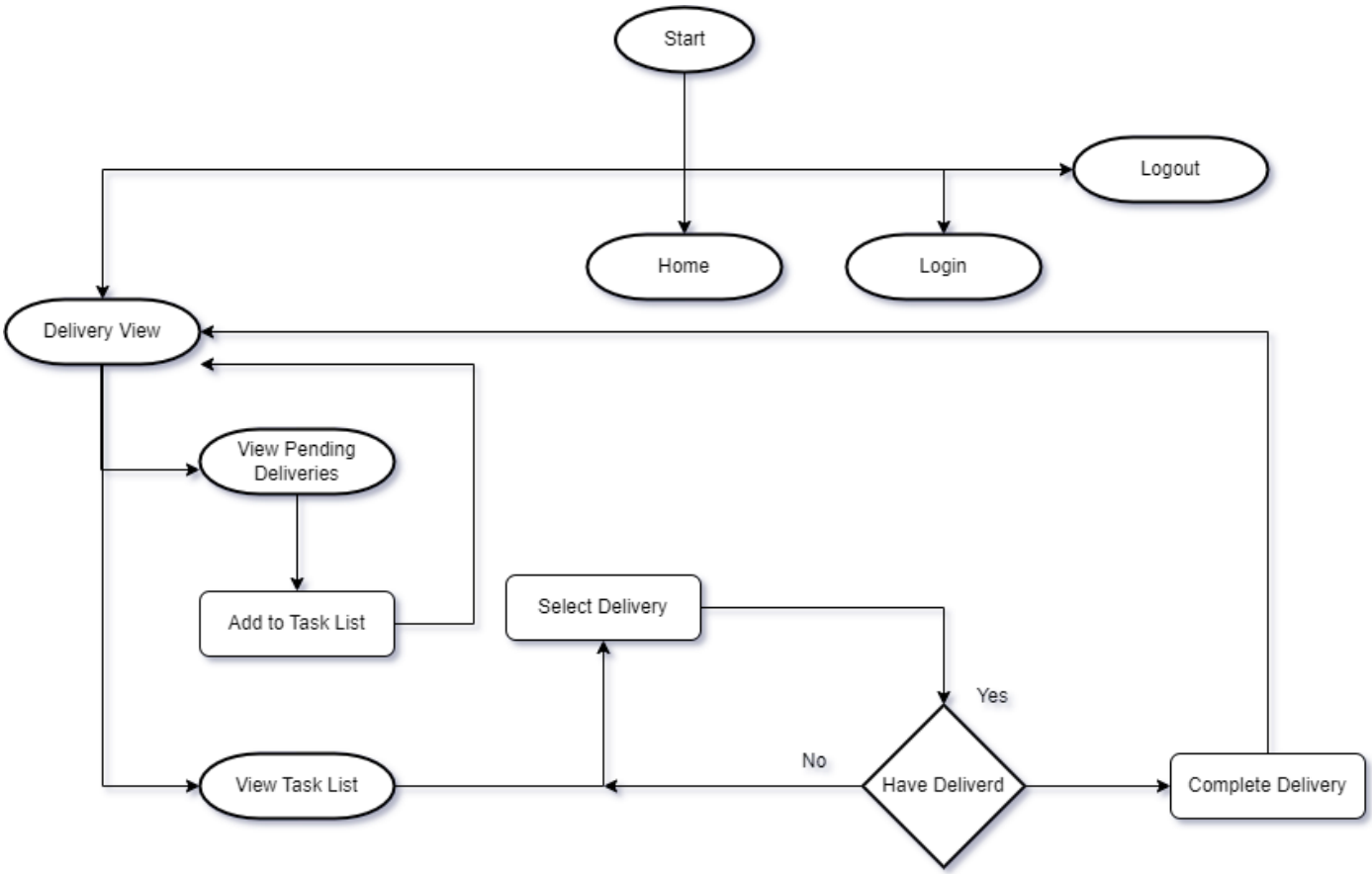


Figure 4. 11 Deliverer Activity Diagram.

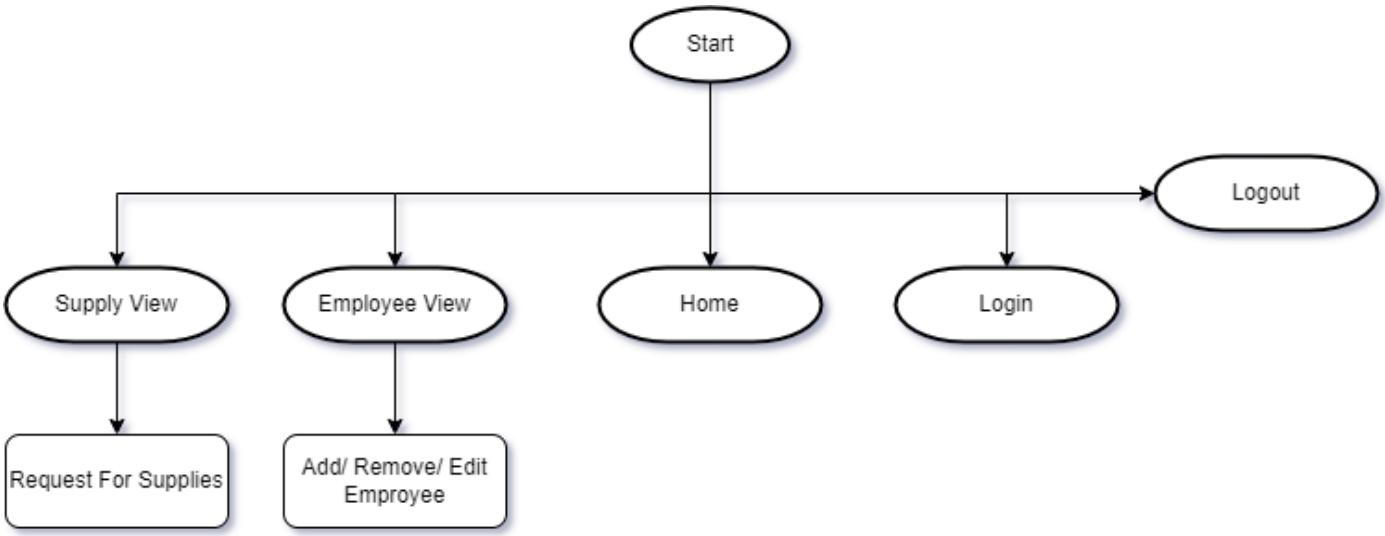


Figure 4. 12 Admin Activity Diagram.

4.2.4 Web Application

The primary objective of developing the web application was to provide an online platform for restaurant customers to conveniently order food through the internet. In this project, we have followed the standard online food ordering system, but we have also incorporated additional features, such as allowing customers to modify existing recipes according to their preferences. The following is an overview of the web application's various views and functionalities:

1. Home: The Home view serves as the landing page for the web application, providing a brief introduction to the restaurant and its offerings. It also includes suggestions recipes to engage customers.

2. Sign In/Sign Up: The Sign In/Sign Up view allows new customers to create an account or existing customers to log in to their accounts. This feature enables personalized experiences, such as saving preferences, viewing order history, and managing payment methods.

3. Menu: The Menu view showcases the restaurant's diverse range of food items. Customers can explore different categories and browse through the available options. Each menu item includes essential details such as a description, price, and any applicable dietary information.

4. Customizing Food Items: The New Food Item view is a pivotal component of our solution. Here, customers can customize their selected food item from the Menu view to suit their preferences. The app provides an intuitive interface that allows customers to add or remove ingredients, adjust spice levels, and make other modifications. This feature enhances the customer experience by providing a personalized dining experience.

5. Cart: The Cart view displays the items that customers have selected for ordering. It provides a summary of the order, including the customized food items, quantities, and total cost. Customers can review their selections, make any final modifications, and proceed to the payment stage.

6. Payment: The Payment view enables customers to securely complete their transactions. The app has integrated PayPal payment gateways to offer a seamless and trusted payment experience.

7. Food Review: The Food Review view allows customers to provide feedback and rate their dining experience. This feature not only helps the restaurant to improve its offerings but also assists other customers in making informed decisions. Customers can leave comments, rate the food quality, service, and overall satisfaction.

8. Account: The Account view provides customers with a centralized location to manage their profiles and preferences. Customers can update personal information, view order history, track deliveries, manage payment methods, and access exclusive offers or loyalty programs.

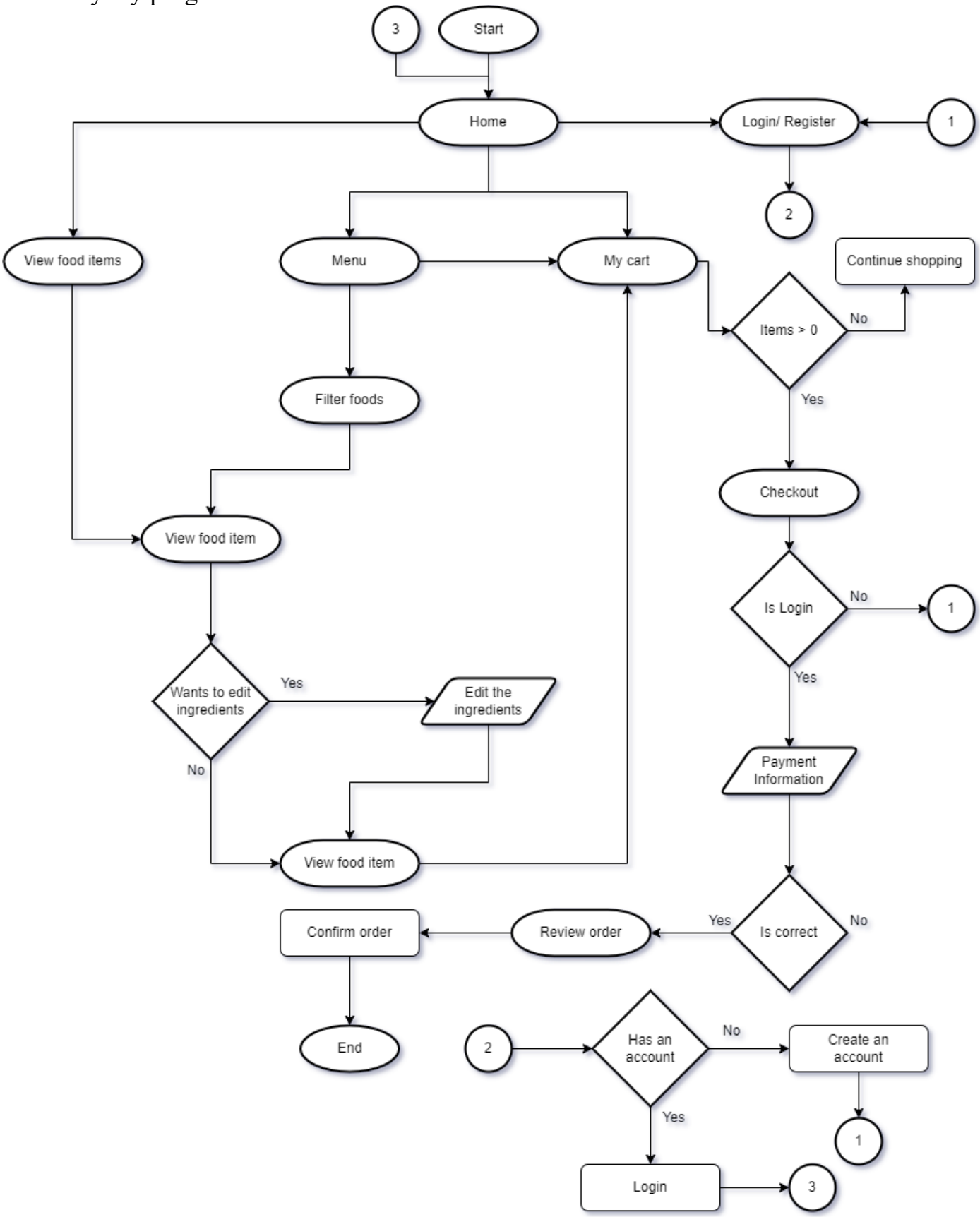


Figure 4. 13 Customer Activity Diagram.

In summary, web application offers an intuitive and user-friendly platform for customers to order food online. The inclusion of the New Food Item view allows customers to customize their orders, tailoring them to their specific preferences. By incorporating features such as personalized accounts, secure payments, and a food review system, it provides a seamless and delightful experience for restaurant customers.

4.2.5 Recommendation System

The system incorporates a recommendation system that suggests food recipes to customers within the web application. The recommendation system is based on collaborative filtering, a popular technique used in personalized recommendation systems.

Collaborative filtering analyzes the behavior and preferences of multiple users to make recommendations. In this context, it examines the historical data of various customers, such as their recipe reviews and ratings. By identifying patterns and similarities in user preferences, the system can suggest recipes that are likely to be of interest to a particular customer.

4.3 Common Design Patterns

The project incorporates two design patterns to ensure an efficient and maintainable architecture. The MVVM (Model View ViewModel) design pattern is used for the Inventory Management System, while the MVC (Model View Controller) design pattern is employed for the web application.

4.3.1 MVVM Design Pattern for the Inventory Management System

The Inventory Management System follows the MVVM design pattern, which promotes loose coupling and separation of concerns.

- The Models component encapsulates the data structures and business logic specific to the Inventory Management System. It allows for reusability across multiple projects without negatively impacting the system's functionality.

- The Views component focuses on the user interface elements of the Inventory Management System. It presents data to the users and captures user interactions. Views remain decoupled from the underlying logic, ensuring flexibility and ease of customization.

- The ViewModels component acts as an intermediary between the Models and Views. It provides the necessary data and functionality for the Views to display and manipulate the system's data. ViewModels separate the business logic from the Views, facilitating independent development and testing.

By implementing the MVVM design pattern, the Inventory Management System achieves maintainability, reusability, and modularity, enabling collaboration among multiple developers.

4.3.2 MVC Design Pattern for the Web Application

The web application component of the project adheres to the MVC design pattern, which emphasizes the separation of concerns and modularity.

- The Models component represents the data structures and business logic specific to the web application. It manages data, performs data manipulation, and interacts with external resources, catering to the web application's requirements.

- The Views component encompasses the user interface elements of the web application. It renders information and captures user interactions, providing a visual representation of data and enabling user engagement.

- The Controllers component acts as the bridge between the Models and Views. Controllers handle user requests, process input data, interact with the Models, and determine which Views to present. They ensure the flow of information and control the behavior of the web application.

By utilizing the MVC design pattern, the web application achieves separation of concerns, enabling independent development and simplified maintenance. The modular structure of MVC enhances code organization, testability, and scalability.

In conclusion, the project successfully integrates the MVVM design pattern for the Inventory Management System and the MVC design pattern for the web application. These design patterns ensure loose coupling, separation of concerns, and collaboration among developers, resulting in a well-structured and maintainable system.

In addition to the MVVM design pattern for the Inventory Management System and the MVC design pattern for the web application, the project incorporates the Generic Repository pattern and the Service Layer pattern. These patterns are used in the Data Access Layer (DAL) and the Core layer to facilitate efficient data access and domain-driven event execution across both distributed systems. Figures 4.4-4.8 desist the design view of these design patterns.

4.3.3 Generic Repository Pattern

The Generic Repository pattern is employed in the DAL to encapsulate the database context and provide a standardized interface for data access. This pattern promotes reusability and maintainability by abstracting the underlying data storage details.

By using the Generic Repository pattern, the DAL can provide a consistent set of operations for accessing and manipulating data, regardless of the specific data storage technology (e.g., relational database, NoSQL database, etc.) employed. The repositories serve as an abstraction layer between the data storage and the Core layer, allowing for decoupling and facilitating easier testing and maintenance.

4.3.4 Service Layer Pattern

The Service Layer pattern is utilized in the Core layer to encapsulate business logic and provide a unified interface for executing domain-driven events. This pattern promotes separation of concerns and ensures that the business logic is centralized and reusable across different components of the project.

The Service Layer acts as an intermediary between the distributed systems (Inventory Management System and web application) and the DAL. It encapsulates the necessary operations and validations related to the domain models and provides a cohesive set of services that can be accessed by the different components.

By employing the Service Layer pattern, the project achieves a modular and scalable architecture. It enables the centralized management of domain-driven events, such as creating or updating inventory items, processing orders, or generating reports. The Service Layer promotes code reuse, testability, and maintainability by encapsulating the complex business logic and keeping it separate from the presentation and data access layers.

In conclusion, the project incorporates the Generic Repository pattern in the DAL to provide a consistent and reusable data access layer, and the Service Layer pattern in the Core layer to encapsulate domain-driven events and promote separation of concerns. These design patterns, along with the MVVM and MVC patterns, contribute to the overall robustness, modularity, and maintainability of the project.

4.4 Testing

The project has implemented a robust testing strategy to ensure the quality and reliability of the system. The testing approach can be outlined as follows:

1. Unit Testing: Unit testing is a fundamental part of the testing process, and it is implemented for different layers of the system. The unit tests focus on testing individual units of code in isolation to ensure their functionality is correct. In this project, unit testing is performed for the following components:
    a. Repository Layer: Unit tests are designed to validate the data access and manipulation operations performed by the repository layer. These tests ensure the integrity of data storage and retrieval processes.

   b. Services Layer: Unit tests are created to test the various services implemented in the system. These tests validate the functionality and behavior of the services, ensuring they perform their intended tasks accurately.

2. Web API: For the web API component, manual testing is conducted using tools such as Postman. Manual testing involves sending requests to the API endpoints and verifying the responses to ensure they align with the expected behavior. This approach allows for interactive and exploratory testing, enabling testers to validate the API's functionality, input validation, error handling, and response formats.

3. Testing Timeline: Unit testing is initiated early in the project timeline as soon as the database is created. This allows for the early detection of any issues or discrepancies and facilitates the implementation of corrective measures promptly. By incorporating testing at an early stage, the development team can identify and resolve bugs more efficiently, resulting in a more robust and stable system.

4. Test Maintenance and Iterative Testing: Testing is an iterative process, and as the project evolves, the test suite is continuously maintained and updated. As new features are added or existing functionalities are modified, corresponding test cases are created or modified accordingly. This iterative approach ensures that the testing efforts stay aligned with the evolving project requirements.

By following these testing practices and methodologies, the project aims to deliver a robust and reliable system that meets the specified requirements and provides a seamless user experience.

4.5 Hosting

The project utilizes Microsoft Azure cloud services for hosting its various components. The database is hosted in Azure SQL Server, which provides a scalable and reliable platform for managing relational data. The ML Model is stored in Azure Blob Storage, which offers a cost-effective and flexible solution for storing large amounts of data. The web app is hosted in Azure Web App Service, which enables rapid deployment and scaling of web applications.

By hosting the project’s components in Azure, the system benefits from the reliability, scalability, and security offered by Microsoft’s cloud platform. This ensures that the system can handle increasing amounts of data and traffic while maintaining high levels of performance and availability.

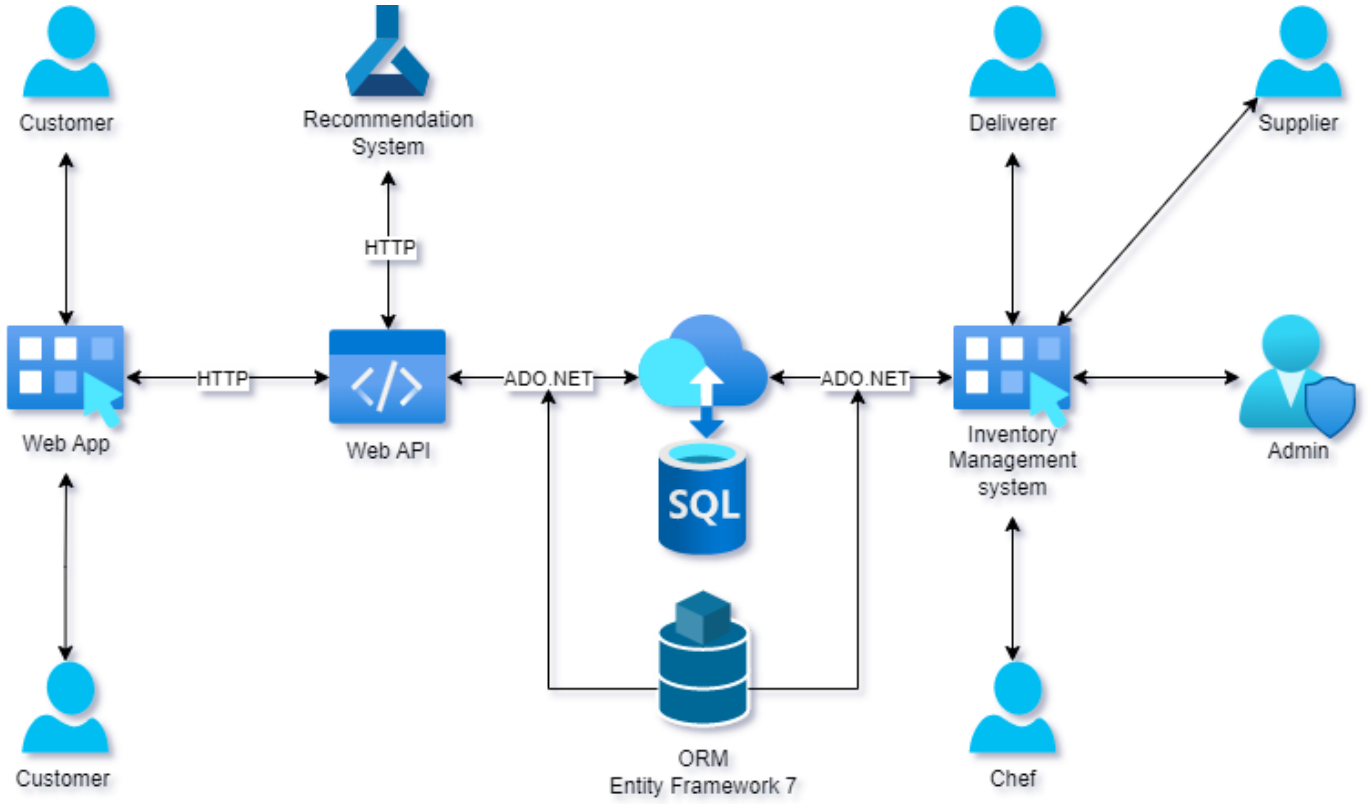


Figure 4. 14 High Level Architecture Diagram.

4.6 Source Control and DevOps Pipeline

The project utilizes GitHub as the source control system to store the entire project codebase. The source code is organized using a mono repository approach, where the core projects, DAL (Data Access Layer), and Service (Core) are stored together in a unified repository. This allows for better code organization, version control, and collaboration among the development team.

In addition to source control, the project incorporates a DevOps pipeline to streamline the software development lifecycle. The DevOps pipeline is responsible for automating various stages, including testing, verification of testing results, deployments, and publishing. The pipeline ensures consistent and reliable delivery of the software, reducing manual effort and mitigating the risk of human error.

The DevOps pipeline includes the following key components:

1. Continuous Integration (CI): The CI process is triggered whenever changes are pushed to the GitHub repository. The pipeline automatically compiles the code, runs unit tests, and performs other static code analysis tasks. This helps identify issues early in the development cycle, ensuring code quality and preventing the introduction of bugs into the main codebase.

2. Testing: As mentioned earlier, the project incorporates a comprehensive testing strategy. The DevOps pipeline runs automated tests, including unit tests, integration tests, and any other defined test suites. The pipeline verifies the testing results against the expected outcomes, helping ensure the functionality and stability of the system.

3. Verification and Validation: Once the testing phase is complete, the pipeline performs verification and validation checks. This may involve additional checks, such as code reviews, security scans, and adherence to coding standards. By enforcing these checks, the pipeline ensures that the code meets the defined quality standards and best practices.

4. Deployment: After the code has been validated, the pipeline facilitates the deployment process. It automates the deployment tasks, such as packaging the application, configuring deployment environments, and deploying to the target servers or cloud platforms. The pipeline ensures consistency and repeatability in the deployment process, reducing the risk of configuration errors and enabling faster and more efficient deployments.

5. Publishing: Once the application is deployed, the DevOps pipeline can handle the publishing process. This may involve publishing the application to an app store, creating release notes, generating documentation, or notifying stakeholders about the new release. Publishing tasks are automated, saving time and effort while ensuring accurate and consistent release artifacts.

By leveraging GitHub as the source control system and implementing a DevOps pipeline, the project benefits from efficient code management, version control, automated testing, and streamlined deployments. These practices promote collaboration, improve code quality, and enhance the overall software development and delivery process.

4.7 Summary

The chapter outlines an overview of the design and development approach for the "Food Desire" online food ordering system. The chapter covers important aspects such as the database design, data access layer, domain service layer, inventory management system, web application, recommendation system, and the use of common design patterns.

It highlights the adoption of design patterns like MVVM (Model-View-ViewModel) for the inventory management system and MVC (Model-View-Controller) for the web application. These patterns promote separation of concerns and enhance code organization.

The chapter also discusses the use of source control with GitHub and the implementation of a DevOps pipeline for testing, deployment, and publishing. These practices ensure version control, automated testing, and streamlined development processes.

In summary, The chapter provides a concise overview of the technical aspects of the "Food Desire" project, including its design patterns, source control, and DevOps pipeline. This chapter serves as a guide to understanding the system's architecture and the methodologies employed in its development.

Chapter 5
DEVELOPMENT

5.1 Introduction

The Food Desire project is an online food ordering system developed using .NET 7. The project consists of several shared libraries, including Models, Data Access Layer (DAL), and Domain Service Layer (Core). These layers serve different purposes and are discussed in more detail below.

- Models: The Models library contains the data structures and business logic specific to the Food Desire system. It allows for reusability across multiple projects without negatively impacting the system's functionality.

- Data Access Layer (DAL): The Data Access Layer (DAL) is responsible for managing interactions between the application and the database. Entity Framework Core is used to design the database and handle CRUD (Create, Read, Update, Delete) operations from repositories.

- Domain Service Layer (Core): The services in the Core layer depend on the repositories in the DAL. These services provide the necessary data and functionality for the Inventory Management System (IMS) Desktop app and the Web App to display and manipulate the system's data.

5.2 Models

The Models library contains the data structures specific to the Food Desire system. These data structures represent the entities in the system, such as customers, orders, food items, etc. In the Food Desire project, Entity Framework Core is used to map these entities to the database.

Entity Framework Core allows developers to use data annotations to configure relationships between entities. Data annotations are attributes that can be applied to classes or properties to provide additional metadata for Entity Framework Core. For example, the [ForeignKey] attribute can be used to specify which property should be used as a foreign key in a relationship.

In the Food Desire project, relationships between entities are configured using data annotations and the Fluent API provided by Entity Framework Core. For example, a one-to-many relationship between a customer and their orders can be configured using the [ForeignKey] attribute on the CustomerId property of the Order entity. This tells Entity Framework Core that the CustomerId property should be used as a foreign key to represent the relationship between a customer and their orders.

```
namespace FoodDesire.Models;
public sealed class Order : TrackedEntity {
    [Required]
    public int CustomerId { get; set; }
    [AllowNull]
    public int? DeliveryId { get; set; }
    [Required]
    public DateTime DateTime { get; set; } = DateTime.Now;
    public OrderStatus Status { get; set; } = OrderStatus.Pending;
    [Column(TypeName = "Decimal(18,2)")]
    public decimal Price { get; set; } = decimal.Zero;

    [ForeignKey(nameof(CustomerId))]
    public Customer? Customer { get; set; }
    [ForeignKey(nameof(DeliveryId))]
    public Delivery? Delivery { get; set; }
    public Payment? Payment { get; set; }
}
```

By using data annotations and the Fluent API, the Models library can configure relationships between entities in a flexible and intuitive way. This allows for easy management of complex data relationships and ensures that the data model accurately represents the domain.

5.3 Data Access Layer (DAL)

The Data Access Layer (DAL) is responsible for managing interactions between the application and the database. In the Food Desire project, Entity Framework Core is used to design the database and handle CRUD (Create, Read, Update, Delete) operations from repositories.

Entity Framework Core is an Object-Relational Mapping (ORM) framework that allows developers to work with data using domain-specific objects and properties, without having to worry about the underlying database details. In the DAL of the Food Desire project, a database context is created using Entity Framework Core. This context represents a session with the database and allows for querying and saving data.

5.3.1 Database Context

The ApplicationDbContext class is a subclass of the DbContext class provided by Entity Framework Core. It represents a session with the database and allows for querying and saving data. In the Food Desire project, the ApplicationDbContext class is used to configure the database connection and define the database schema.

The ApplicationDbContext class includes DbSet properties for each entity in the system. These properties provide access to the data for each entity and allow for performing CRUD (Create,

Read, Update, Delete) operations. The ApplicationDbContext class also includes methods for configuring relationships between entities and customizing the database schema.

```
namespace FoodDesire.DAL.Context;
public class ApplicationDbContext : DbContext {
    public DbSet<User>? User { get; set; }
    public DbSet<Account>? Account { get; set; }
    public DbSet<Admin>? Admin { get; set; }
    public DbSet<Customer>? Customer { get; set; }
    public DbSet<Employee>? Employee { get; set; }
    public DbSet<Deliverer>? Deliverer { get; set; }
    public DbSet<Chef>? Chef { get; set; }
    public DbSet<Supplier>? Supplier { get; set; }
    public DbSet<IngredientCategory>? IngredientCategory { get; set; }
    public DbSet<Ingredient>? Ingredient { get; set; }
    public DbSet<RecipeCategory>? FoodCategory { get; set; }
    public DbSet<Recipe>? Recipe { get; set; }
    public DbSet<Image>? Image { get; set; }
    public DbSet<RecipeReview>? RecipeReview { get; set; }
    public DbSet<FoodItem>? FoodItem { get; set; }
    public DbSet<Order>? Order { get; set; }
    public DbSet<Payment>? Payment { get; set; }
    public DbSet<Delivery>? Delivery { get; set; }
    public DbSet<Supply>? Supply { get; set; }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options: options) { }
}
```

Figure 5. 1 Application Database Context.

5.3.2 Repositories

The repository pattern is implemented in the DAL to encapsulate the database context and provide a standardized interface for data access. This pattern promotes reusability and maintainability by abstracting the underlying data storage details. In the Food Desire project, repositories are used to perform CRUD operations on the entities in the Models library.

```
namespace FoodDesire.DAL.Contracts.Repositories;
public interface IRepository<T> where T : Entity {
    Task<T> Add(T entity);
    Task<List<T>> AddAll(List<T> entities);
    Task<T> GetByID(int? Id);
    Task<List<T>> GetAll();
    Task<T> Update(T entity);
    Task<bool> Delete(int Id);
}
```

Figure 5. 2 Generic Repository.

By using Entity Framework Core and implementing the repository pattern, the DAL provides a consistent set of operations for accessing and manipulating data, regardless of the specific data storage technology (e.g., relational database, NoSQL database, etc.) employed. This allows for loose coupling between the DAL and the rest of the system, making it easier to manage dependencies at runtime.

5.4 Domain Services Layer (Core)

The Domain Service Layer (Core) is responsible for providing the necessary data and functionality for the Inventory Management System (IMS) Desktop app and the Web App to display and manipulate the system's data. The services in the Core layer depend on the repositories in the Data Access Layer (DAL) to perform CRUD (Create, Read, Update, Delete) operations on the entities in the Models library.

Each service in the Core layer is designed to handle a specific set of operations related to a particular entity or group of entities. For example, there may be a CustomerOrderService that handles operations related to customers and order entities such as creating new order for customer, updating order information, or retrieving order data. This service would depend on the CustomerRepository and the OrderRepository in the DAL to perform these operations.

```
namespace FoodDesire.Core.Contracts.Services;
public interface ICustomerOrderService {
    Task<List<Order>> GetAllOrdersForCustomerById(int customerId);
    Task<List<Order>> GetAllOrdersToDeliverForCustomerById(int customerId);
    Task<List<Order>> GetAllPendingOrdersForCustomerById(int customerId);
}
```

Figure 5. 3 Contract for CustomerOrderService.

```
namespace FoodDesire.Core.Services;
public class CustomerOrderService : ICustomerOrderService {
    private readonly IRepository<Customer> _customerRepository;
    private readonly IRepository<Order> _orderRepository;

    public CustomerOrderService(
        IRepository<Customer> customerRepository,
        IRepository<Order> orderRepository
        ) {
        _customerRepository = customerRepository;
        _orderRepository = orderRepository;
    }
    public async Task<List<Order>> GetAllOrdersForCustomerById(int customerId) {…
    public async Task<List<Order>> GetAllOrdersToDeliverForCustomerById(int customerId) {…
    public async Task<List<Order>> GetAllPendingOrdersForCustomerById(int customerId) {…
}
```

Figure 5. 4 Implementation for CustomerOrderService.

The services in the Core layer can also handle more complex database operations that involve multiple entities or relationships. For example, a FoodItemService might need to retrieve data about food items and their associated ingredients. This service would depend on multiple repositories, such as the FoodItemRepository and the IngredientRepository, to perform these operations.

By depending on multiple repositories and handling complex database operations, the services in the Core layer provide a flexible and powerful way to access and manipulate data in the

Food Desire system. This allows for easy management of complex data relationships and ensures that the data model accurately represents the domain.

5.5 Usage Of Dependency Injection

Dependency Injection (DI) is a design pattern that promotes loose coupling between components by allowing dependencies to be injected at runtime. This means that instead of creating dependencies directly within a component, the component receives its dependencies as parameters, typically through its constructor.

In the Food Desire project, Dependency Injection (DI) is used to register and configure services and repositories in both the Inventory Management System (IMS) Desktop app and the Web App. This is done using a static Configure method in the Core and DAL layers, which registers the services and repositories with the DI container.

The DI container is responsible for managing the creation and lifetime of objects. In .NET 7, the built-in DI container is used to register services with different lifetimes, such as transient, scoped, or singleton.

```
using FoodDesire.Core.Services;
using Microsoft.Extensions.DependencyInjection;

namespace FoodDesire.Core;
public static class Configure {

    public static void ConfigureAllForTesting(IServiceCollection services) {
        GetServices(services: services);
    }

    private static void GetServices(IServiceCollection services) {
        services.AddTransient<IOrderService, OrderService>();
... ...
    }
}
```

Figure 5. 5 DI Registration Usage.

For example, the above code snippet, the AddTransient method is used to register the IOrderService and IFoodItemService interfaces with their corresponding implementations. This means that a new instance of these services will be created each time they are requested.

By using Dependency Injection (DI), the Food Desire project can achieve loose coupling between components and make it easier to manage dependencies at runtime. This allows for more flexible and maintainable code, as well as easier testing and development.

5.6 Inventory Management System

The Inventory Management System (IMS) is a desktop application that has been built using WinUI 3, which is a modern UI framework for building Windows applications. The IMS allows restaurant staff to manage inventory, process orders, and perform other tasks related to the Food Desire system.

5.6.1 Design

Inventory Management System (IMS) project is a complex and well-designed system that allows restaurant staff to manage inventory and process orders. The IMS follows the Model-View-ViewModel (MVVM) design pattern and makes use of modern technologies such as WinUI 3 and the MVVM Community Toolkit to provide a user-friendly and interactive front-end.

The IMS project makes use of various design patterns and best practices to ensure a robust and maintainable architecture. These include the use of Dependency Injection to manage dependencies between components, the use of data binding to connect ViewModels and Views, and the use of XAML to design user interfaces.

5.6.1.1 Code-Behind

In the IMS, each main page has a corresponding View, ViewModel, and Service class. For example, the HomePage has a HomeViewModel and a HomePageService. The View is responsible for displaying data and capturing user input, while the ViewModel acts as an intermediary between the View and the Service class.

```
namespace FoodDesire.IMS.Views;
public sealed partial class HomePage : Page {
    public HomeViewModel ViewModel { get; set; }

    public HomePage() {
        ViewModel = App.GetService<HomeViewModel>();
        InitializeComponent();
    }
}
```

Figure 5. 6 ViewModel in a Page.

The HomeViewModel contains properties and methods that are used to bind data to the HomePage View. These properties and methods are responsible for retrieving data from the HomePageService and updating the View with the latest data. The HomePageService, in turn, depends on services from the Core layer to perform data access and manipulation operations.

```
namespace FoodDesire.IMS.ViewModels;
public partial class HomeViewModel : ObservableRecipient, INavigationAware {
    private readonly IHomeService _homeService;
    private readonly INavigationService _navigationService;
    private readonly IMapper _mapper;

    [ObservableProperty]
    private bool _isLoading = true;
    ....
    [ObservableProperty]
    public PlotModel _model = new PlotModel();

    public HomeViewModel(IHomeService homeService, IMapper mapper, INavigationService navigationService) {
        _homeService = homeService;
        _mapper = mapper;
        _navigationService = navigationService;
    }

    public async void OnNavigatedTo(object parameter) {...

    private async Task LoadData() {...

    private async Task LoadInventorySummery() {...

    private async Task LoadToDosCount() {...

    private async Task LoadTop10Recipes() {...

    private async Task loadFinancialData() {...

    private void PlotChart() {...

    public void OnNavigatedFrom() { }

    [RelayCommand]
    public void OnItemClick(RecipeListItemDetail? clickedItem) {...
}
```

Figure 5. 7 ViewModel for HomePage

In this way, the MVVM design pattern promotes separation of concerns and loose coupling between components. The View is responsible for displaying data and capturing user input, while the ViewModel handles data binding and updates. The Service class provides access to data and business logic, allowing for easy management of complex data relationships.

Each ViewModel has a corresponding Service class that depends on services from the Core layer. For example, the HomePage has a HomePageService that depends on services such as IngredientService, PaymentService, OrderService, RecipeService from the Core layer.

```
namespace FoodDesire.IMS.Core.Services;
public class HomeService : IHomeService {
    private readonly IIngredientService _ingredientService;
    private readonly IPaymentService _paymentService;
    private readonly IOrderService _orderService;
    private readonly IRecipeService _recipeService;      You, 2 weeks ago

    public HomeService(
        IIngredientService ingredientService,
        IPaymentService paymentService,
        IOrderService orderService,
        IRecipeService recipeService) {
        _ingredientService = ingredientService;
        _paymentService = paymentService;
        _orderService = orderService;
        _recipeService = recipeService;
    }

    public async Task<InventorySummary> GetInventorySummery() {…

    public async Task<int> GetPendingOrderCount() {…

    public Task<List<Supply>> GetRecentSupply() {…

    public async Task<List<Payment>> GetExpenses() {…

    public async Task<List<Payment>> GetIncomes() {…

    public async Task<List<Recipe>> GetTop10Recipes() {…

    public async Task<int> GetCompletedOrderCount() {…
}
```

Figure 5. 8 Example of Page Services.

In above figure, the HomeService has dependencies on the IngredientService, PaymentService, OrderService, RecipeService interfaces. These dependencies are resolved at runtime using Dependency Injection (DI). This means that instead of creating instances of the Core services directly within the HomePageService, the HomePageService receives its dependencies as parameters, typically through its constructor.

Additionally, the Inventory Management System (IMS) project makes use of the MVVM Community Toolkit, a collection of tools and components that can help developers implement the Model-View-ViewModel (MVVM) design pattern in their applications. The toolkit includes features such as behaviors, commands, converters, and helpers that can simplify the development of MVVM-based applications. In the IMS project, the MVVM Community Toolkit is used to provide additional functionality and simplify the implementation of the MVVM design pattern.

5.6.1.2 Front-View

In the IMS, each View is defined using XAML markup. This markup specifies the layout and appearance of the user interface elements, such as buttons, text boxes, and lists. The XAML markup is then parsed and rendered by the WinUI 3 runtime to create the final user interface.

One of the benefits of using XAML in WinUI 3 applications is that it allows for a clear separation between the user interface design and the underlying logic. This makes it easier for designers and developers to work together on the same project, as they can focus on their respective areas of expertise without interfering with each other's work.

Figure 5. 9 Home View

Overall, the Inventory Management System (IMS) project of the Food Desire system is a well-designed and well-implemented system that provides a valuable tool for restaurant staff to manage inventory and process orders. Its use of modern technologies and best practices ensures a high-quality user experience and a robust and maintainable architecture.

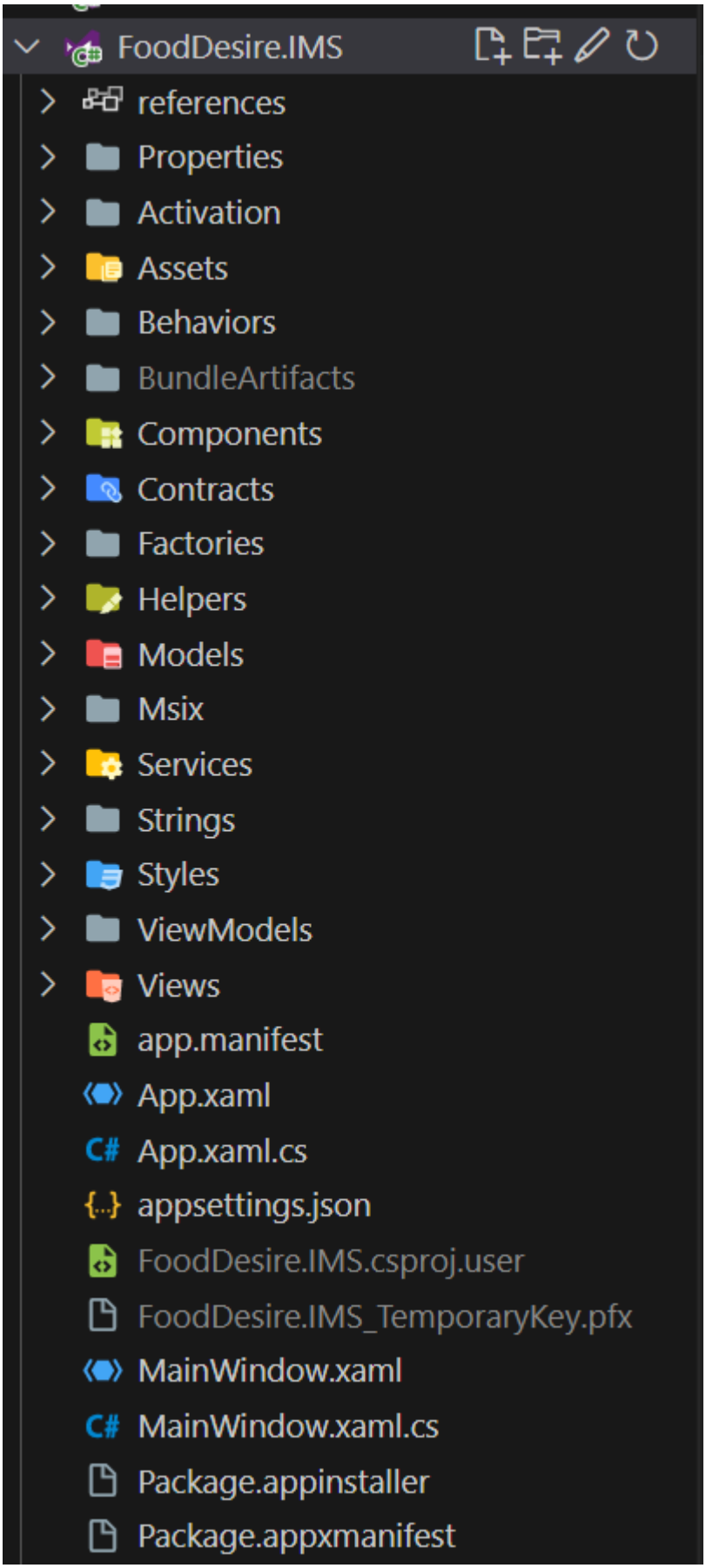


Figure 5. 10 Application Architecture of the IMS.

5.7 Web Application

The Web Application of the Food Desire system provides an online platform for customers to conveniently order food from the restaurant. The application was developed using ASP.NET Core and Blazor, which are modern web development frameworks that enable the creation of fast and responsive web applications.

5.7.1 Web API

The Web API component of the Food Desire system provides a RESTful interface for the web client to interact with the backend services. The Web API was developed using ASP.NET Core, which is a modern web development framework that enables the creation of fast and scalable web APIs.

The architecture of the Web API is like the Inventory Management System (IMS). Just like the IMS's ViewModel has its own page service, the Web API controllers have their own controller service. The controller service implements the core functionality of the system, such as processing orders, generating recommendations, and managing inventory.

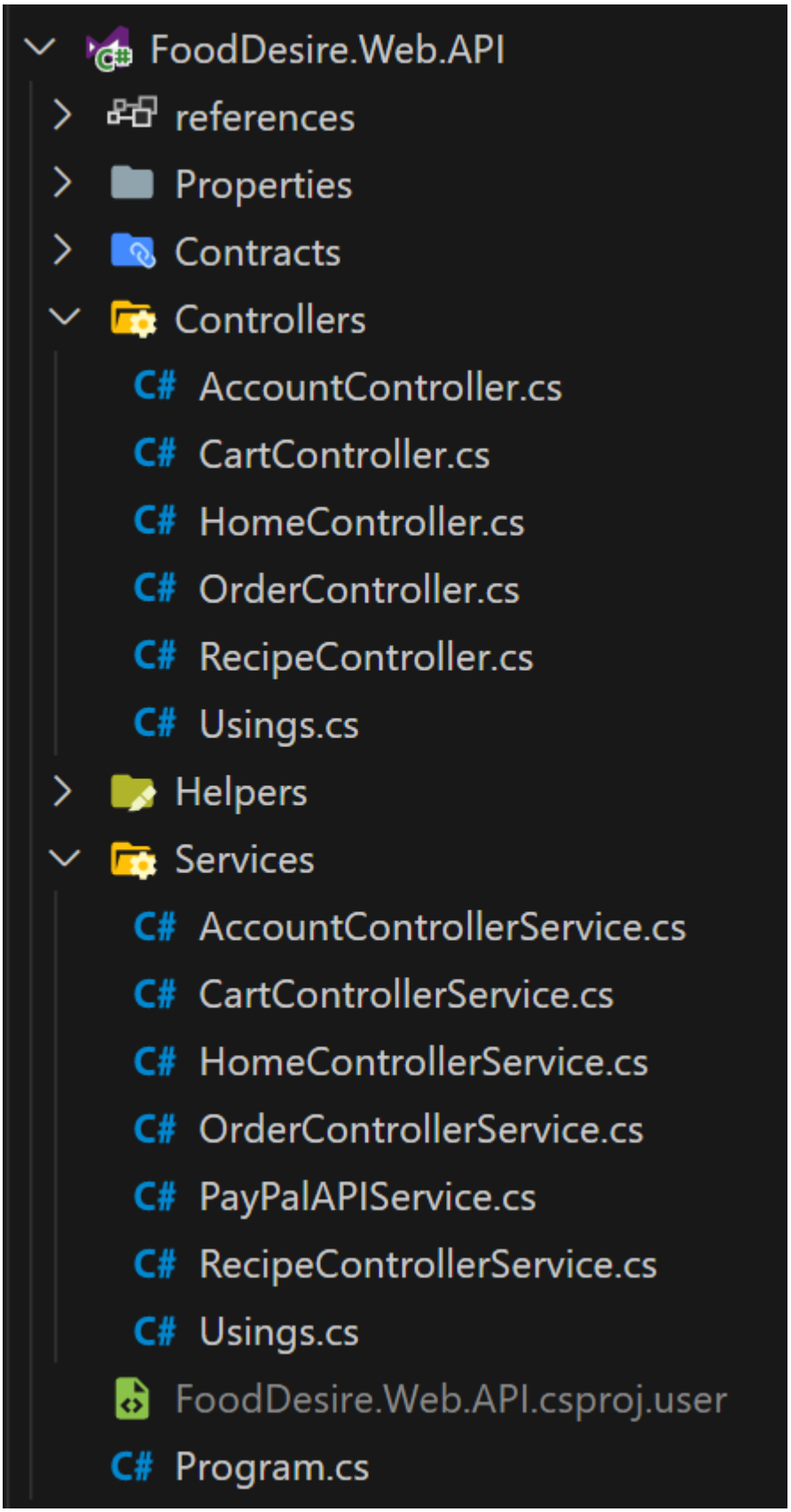


Figure 5. 11 Application Architecture of Web API.

To ensure secure communication between the web client and the Web API, the system uses HTTPS and implements authentication and authorization mechanisms. This ensures that only authorized users can access the system and perform actions.

5.7.2 Web Client

The Web Client component of the Food Desire system provides an online platform for customers to conveniently order food from the restaurant. The web client was developed using Blazor and MudBlazor, which are modern web development frameworks that enable the creation of fast and responsive web applications.

MudBlazor is a popular component library for Blazor that provides a wide range of reusable UI components. The Web Client makes use of these components to create a consistent and visually appealing user interface.

The architecture of the Blazor client project is such that each page has its own page service that calls the Web API. The page service acts as an intermediary between the page and the Web API, providing the necessary data and functionality for the page to display and manipulate the system's data.

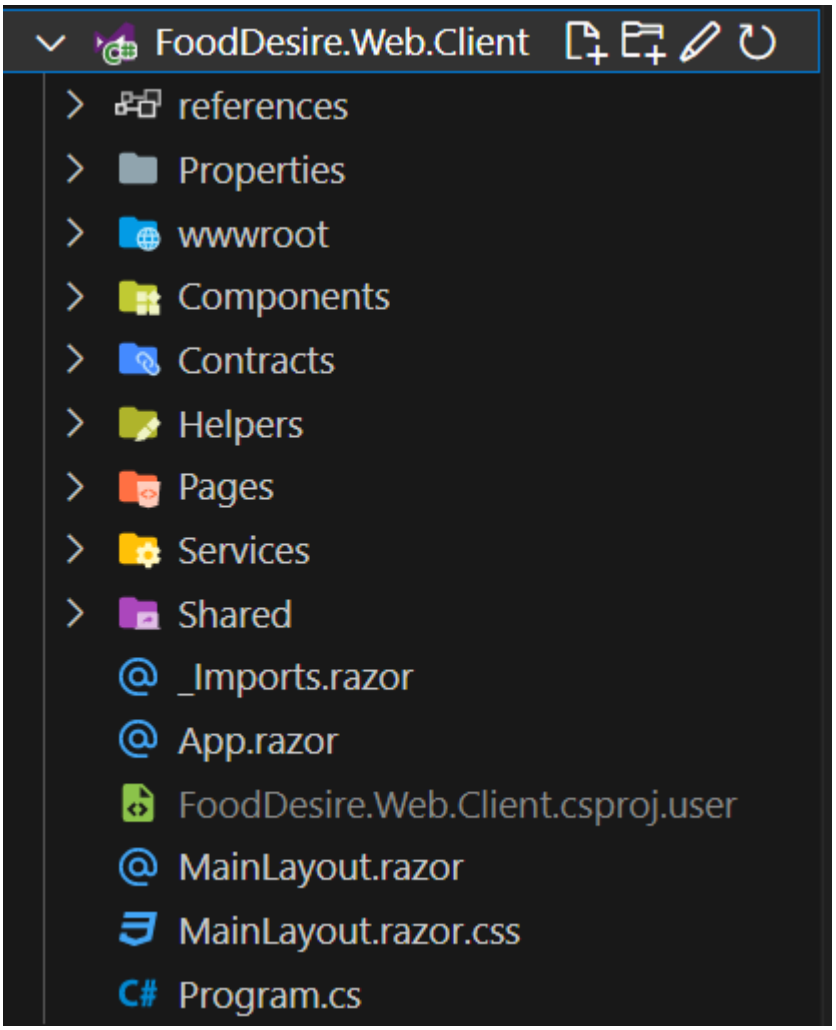


Figure 5. 12 Application Architecture of Wen Client.

The Web Client includes several key features that enhance the user experience. Customers can browse through different food categories and items, customize their orders by changing the amount of ingredients, see the price of orders based on customization, place orders online and pay with various methods, track order status and delivery time, rate and review orders after delivery, and view order history and recommendations.

To ensure a seamless and secure payment experience for customers, the Web Client integrates with PayPal payment gateway. This allows customers to complete their transactions securely using their PayPal accounts or credit cards.

After developing the Web API, it can be merged with the Web Client so that the API can serve the Blazor client output. This means that the client static files can be served from the API.

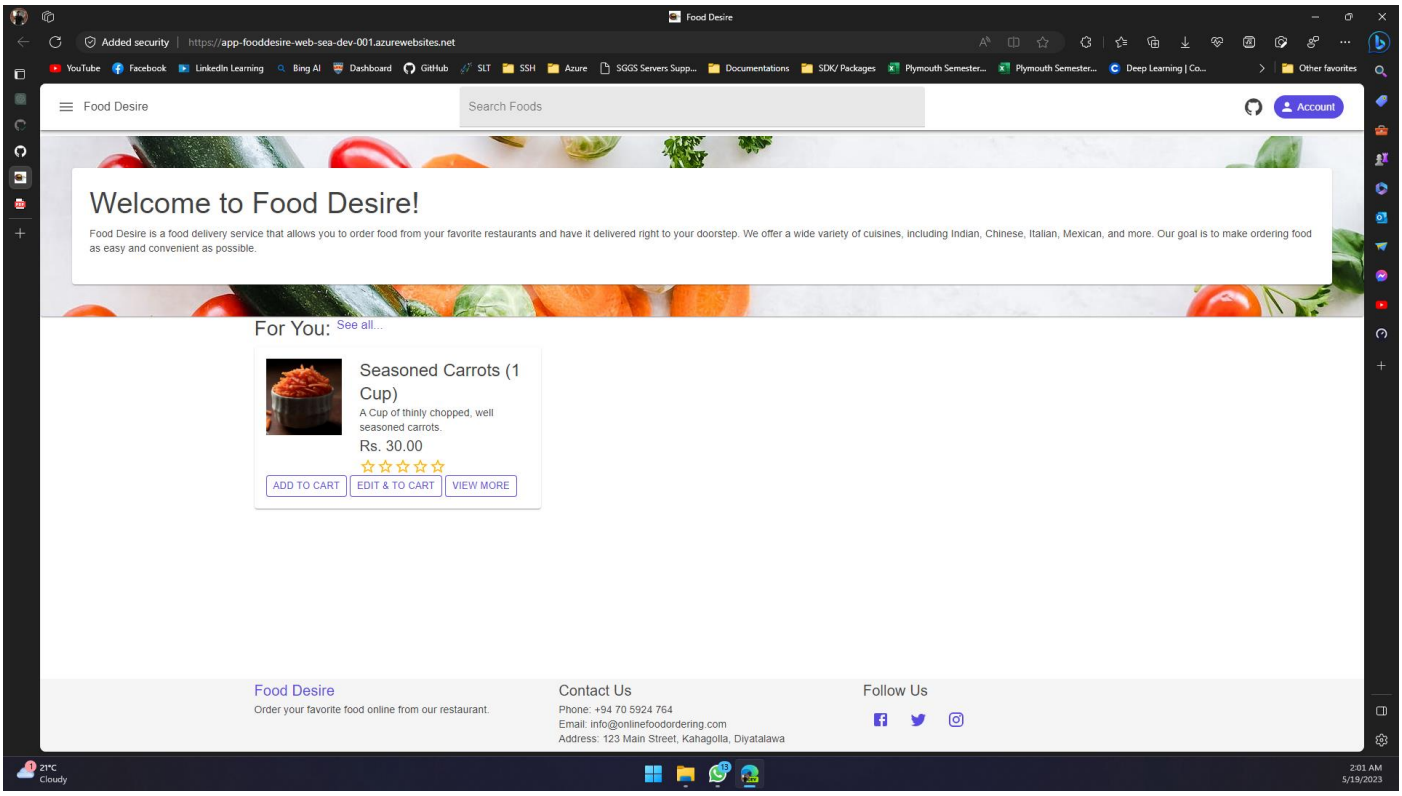


Figure 5. 13 Home Page Web App.

5.8 Recommendation System

The Food Desire system incorporates a recommendation system that suggests food recipes to customers within the web application. The recommendation system is based on collaborative filtering, a popular technique used in personalized recommendation systems.

Collaborative filtering analyzes the behavior and preferences of multiple users to make recommendations. In this context, it examines the historical data of various customers, such

as their recipe reviews and ratings. By identifying patterns and similarities in user preferences, the system can suggest recipes that are likely to be of interest to a particular customer.

The recommendation system was developed using ML.NET, which is a machine learning framework for .NET developers. ML.NET provides a wide range of machine learning algorithms and tools that can be used to build custom machine learning models.

In this project, the recommendation system uses a collaborative filtering algorithm provided by ML.NET. The algorithm analyzes the historical data of customers to identify patterns and similarities in their preferences. Based on this analysis, the algorithm generates personalized recommendations for each customer.

The recommendation system is integrated with the web application, allowing customers to see personalized recommendations when they browse the menu. This enhances the user experience by providing relevant and personalized suggestions, while also benefiting the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue.

Overall, the recommendation system uses ML.NET and collaborative filtering to generate personalized recommendations for customers. By integrating this feature into the web application, the Food Desire system provides an enhanced user experience for restaurant customers.

5.9 Summary

The chapter outlines, the development of the Food Desire system. The system consists of several components, including an Inventory Management System (IMS), a Web API, a Web Client, and a Recommendation System.

The IMS was developed using C#, .NET, and WinUI3, and provides a user interface for managing inventory, employees, suppliers, and recipes. The Web API was developed using ASP.NET Core and provides a RESTful interface for the web client to interact with the backend services. The Web Client was developed using Blazor and MudBlazor and provides an online platform for customers to order food from the restaurant. The Recommendation System was developed using ML.NET and collaborative filtering and generates personalized recommendations for customers based on their preferences and previous orders.

Throughout the development process, various design patterns, tools, and technologies were used to ensure a robust, scalable, and maintainable system. These included the MVVM design pattern for the IMS, the MVC design pattern for the web application, the Generic Repository pattern for data access, and the Service Layer pattern for domain-driven event execution.

In conclusion, the chapter provides an overview of the development of the Food Desire system, including its various components, design patterns, tools, and technologies. By leveraging modern frameworks such as .NET, ASP.NET Core, Blazor, MudBlazor, and ML.NET, the Food Desire system provides a seamless and delightful experience for restaurant customers.

Chapter 6
TESTING

6.1 Introduction

Testing is a crucial part of the software development process, helping to ensure that the system functions correctly and meets the specified requirements. In this project, a robust testing strategy was implemented, including unit testing for the DAL (Data Access Layer) and core layer, as well as manual testing for the Web API.

6.2 Unit Testing

In this project, unit testing was implemented for both the DAL (Data Access Layer) and the core layer using XUnit testing. This approach allowed for the validation of the functionality of the repositories and each core layer service, ensuring that they performed their intended tasks accurately.

XUnit testing is a popular testing framework for .NET that allows developers to write and run unit tests for their code. It provides a range of features and tools that make it easy to create, organize, and execute tests, helping to improve the quality and reliability of the codebase.

In this project, XUnit testing was used to write unit tests for both the DAL and core layer. For the DAL, tests were written to validate the data access and manipulation operations performed by the repository layer. These tests ensured that the repositories were able to correctly store, retrieve, and manipulate data in the database.

For the core layer, tests were written to validate the functionality of each service. These tests checked that the services were able to perform their intended tasks, such as managing inventory, processing orders. By writing and running these tests, any issues or bugs could be caught early in the development process, allowing them to be fixed before they could cause problems.

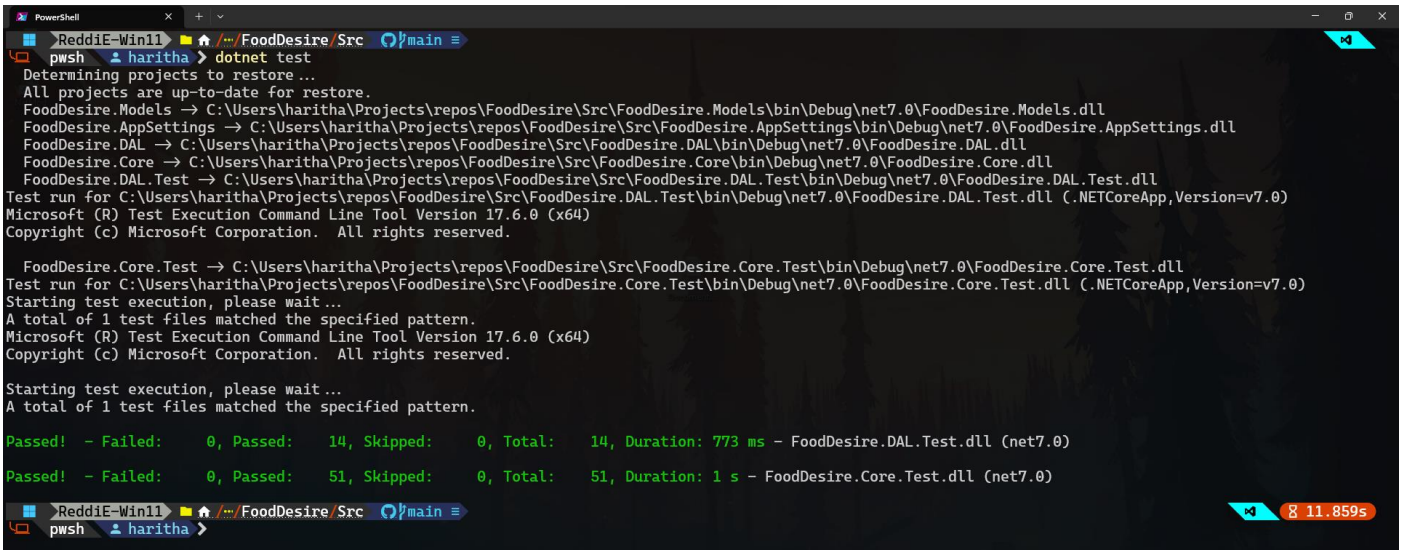


Figure 6. 1 Unit Test Result (Local)

Overall, implementing unit testing using XUnit testing allowed for an improvement in the quality and reliability of the codebase. By writing and running tests for both the DAL and core layer, it was ensured that the repositories and services were functioning correctly, helping to deliver a robust and reliable system.

6.3 API Testing

In addition to unit testing, this project also implemented API testing to ensure the functionality and reliability of the Web API. API testing involves sending requests to the API endpoints and verifying the responses to ensure that they align with the expected behavior.

In this project, manual testing was performed using Swagger UI. Swagger UI is a tool that allows for interactive and exploratory testing of APIs. It provides a user-friendly interface for sending requests to the API endpoints and viewing the responses in real-time. This allowed testers to validate the functionality of each endpoint, checking that they returned the expected responses for different inputs.

Using Swagger UI, testers were able to explore different scenarios and test cases, verifying that the API behaved correctly under different conditions. This helped to catch any issues or bugs early in the development process, allowing them to be fixed before they could cause problems.

Overall, implementing manual API testing using Swagger UI helped to improve the quality and reliability of the Web API. By interactively testing each endpoint, any issues or bugs could be caught early in the development process, allowing them to be fixed before they could cause problems.

Disclaimer: Test cases for Web API can be seen in Appendices: Test cases and Results: Web API Test cases,

6.4 Summary

In summary, Chapter 6 discusses the various testing methods used to ensure the quality and reliability of the system, including unit testing, manual testing, and testing for usability, performance, and security.

Chapter 7
DEVOPS PIPELINE & PUBLISHING

7.1 Introduction

The development and deployment of the Food Desire application involves the use of a DevOps pipeline to automate and streamline the process. This chapter discusses the implementation of this pipeline using GitHub Actions, including automated testing, deployment, and publishing.

7.2 GitHub Actions

To implement a DevOps pipeline using GitHub Actions, a repository was created on GitHub to host the code. Custom workflows were then defined using YAML files, which specify the steps that should be executed whenever certain events occur.

For example, a workflow was created that is triggered whenever code is pushed to the repository. This workflow includes several steps, such as checking out the code, setting up the necessary dependencies, building the code, and running unit tests. By defining these steps in the workflow file, it can be ensured that they are automatically executed whenever code is pushed to the repository.

In addition to running unit tests, other workflows were also created to automate tasks such as deploying the application to different environments and publishing it to production. These workflows were triggered by different events, such as when a pull request was merged or when a new release was created.

7.2.1 Inventory Management Deployment

a workflow was created that builds the application, packages it as an MSIX file, and deploys it to the release in the GitHub Repository where it can be downloaded and installed in the local business environment. It is a GitHub Actions workflow for building, signing, and packaging a WinUI 3 MSIX desktop application built on .NET. This workflow is triggered by events such as when code is pushed to the repository or when a new release is created.

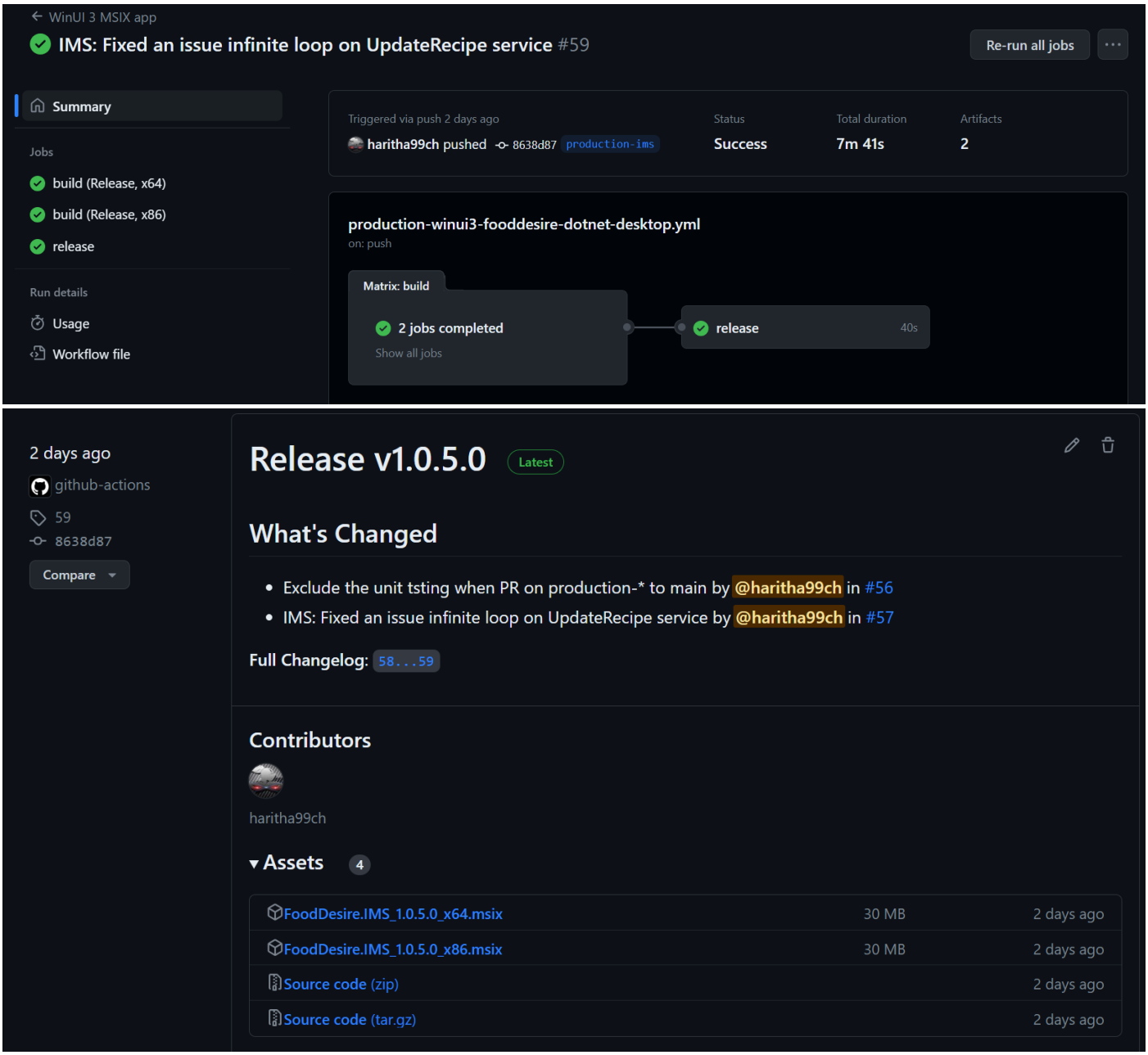


Figure 7. 1 IMS Automated Deployment Results

7.2.2 Web Application Deployment

A workflow was also created that builds the application and deploys it to an Azure App Service. This workflow was also triggered by events such as when code was pushed to the repository or when a new release was created.

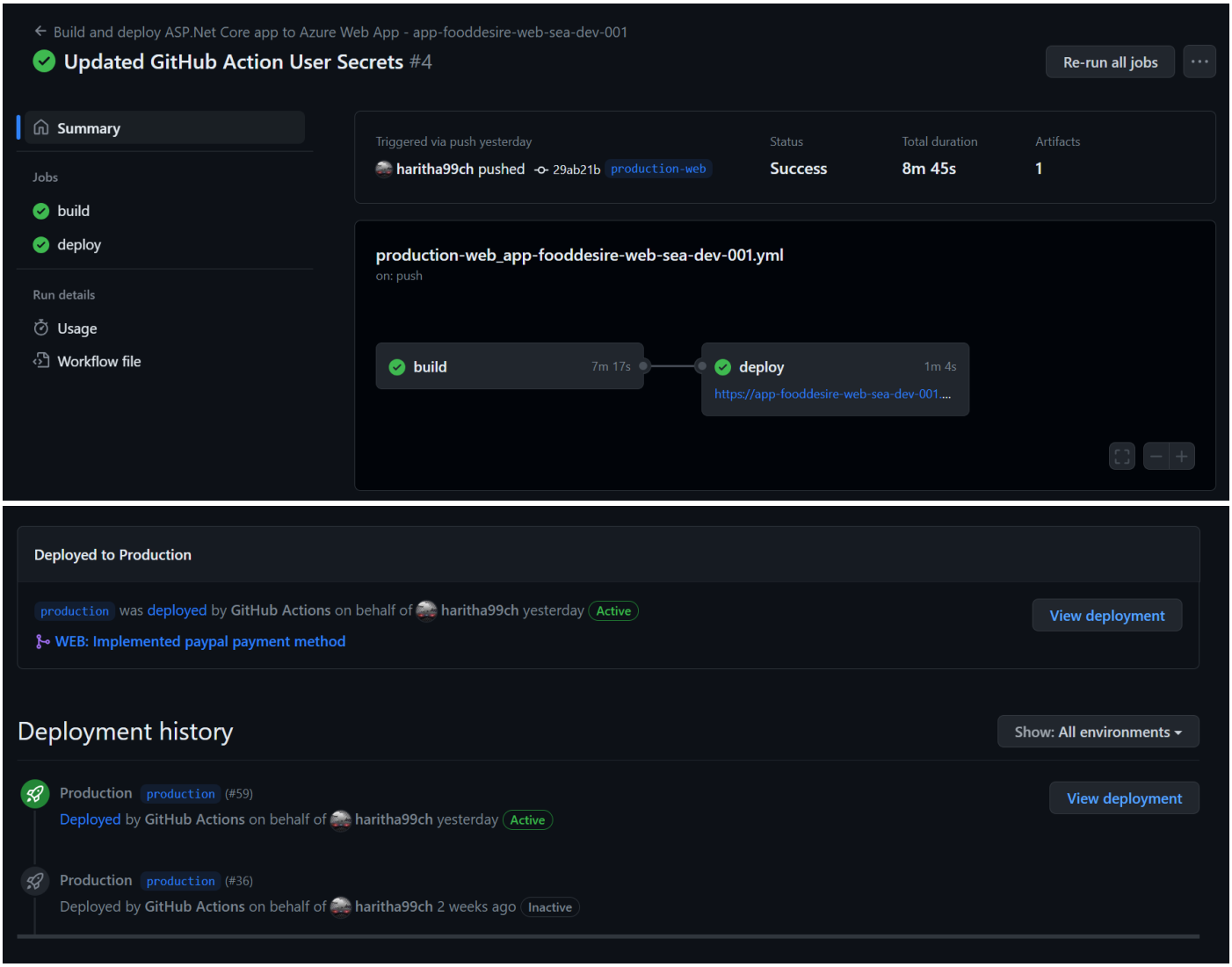


Figure 5. 14 Web App Deployment Results

7.3 Summary

Overall, by using GitHub Actions to automate the deployment of both the IMS MSIX and the web app, it was possible to streamline the process and ensure that these applications were always up-to-date and available for use.

END-PROJECT REPORT

The Food Desire project aimed to develop an online food ordering system for a single restaurant that allows customers to customize their meals by adjusting the amount of ingredients in each meal item. Additionally, a recommendation system was implemented to suggest meal items to users based on their preferences and previous orders, using a collaborative filtering machine learning algorithm. To support this feature, an inventory management system was also developed for the restaurant to manage ingredients and recipes.

The project followed a structured approach, starting with requirements gathering, where the needs, expectations, and constraints of the stakeholders were identified. This was followed by the design phase, where the system architecture and user interface were developed. The implementation phase involved the development of the various components of the system, including the web-based interface, customization feature, recommendation system, and inventory management system. The testing phase ensured that the system met the specified requirements and functioned correctly. Finally, the evaluation phase assessed the effectiveness of the system in meeting its objectives.

The key deliverables of this project included a user-friendly web-based interface for customers to browse and order food from the restaurant, a customization feature that allows customers to adjust the amount of ingredients in each meal item, a recommendation system that suggests meal items to users based on their preferences and previous orders, an inventory management system for the restaurant to manage ingredients and recipes, and a comprehensive final report detailing the design, implementation, testing, and evaluation of the Food Desire online food ordering system and its associated inventory management system.

Overall, it can be concluded that the Food Desire project successfully developed an online food ordering system that enhances the customer experience by providing more control and flexibility over their orders while also benefiting the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue. The project demonstrated effective planning, execution, and delivery of a complex software system.

PROJECT POST-MORTEM

The project was successful in achieving its objectives and delivering a functional and user-friendly online food ordering system. The system provides customers with more control and flexibility over their orders while also benefiting the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue.

However, like any complex software project, there were challenges faced during development. One challenge was ensuring that the system could handle concurrent requests and provide high performance while also maintaining high security by encrypting sensitive data. Another challenge was developing a recommendation system as a cold start that could provide personalized and relevant suggestions to users while also benefiting the restaurant.

In conclusion, the Food Desire project demonstrated effective planning, execution, and delivery of a complex software system. While there were challenges faced during development, these were addressed and overcome through effective problem-solving and collaboration among the project team. The result is a functional and user-friendly online food ordering system that enhances the customer experience and benefits the restaurant.

SUMMARY

The Food Desire project aimed to develop an online food ordering system for a single restaurant that allows customers to customize their meals by adjusting the amount of ingredients in each meal item. Additionally, a recommendation system was implemented to suggest meal items to users based on their preferences and previous orders, using a collaborative filtering machine learning algorithm. To support this feature, an inventory management system was also developed for the restaurant to manage ingredients and recipes.

Overall, the project was successful in achieving its objectives and delivering a functional and user-friendly online food ordering system. The system provides customers with more control and flexibility over their orders while also benefiting the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue.

The project followed a structured approach, starting with requirements gathering, followed by design, implementation, testing, and evaluation. The final deliverables included a user-friendly web-based interface for customers, a customization feature, a recommendation system, an inventory management system, and a comprehensive final report.

In conclusion, the Food Desire project demonstrated effective planning, execution, and delivery of a complex software system. The result is a functional and user-friendly online food ordering system that enhances the customer experience and benefits the restaurant.

REFERENCES.

Ahrens, S. (2012) ‘Recommender Systems’, in.

Daim, T.U. *et al.* (2013) ‘Exploring technology acceptance for online food services’, *Int. J. Bus. Inf. Syst.*, 12, pp. 383–403.

Fujita, T., Shimada, H. and Sato, K. (2014) ‘Self-ordering system of restaurants for considering allergy information’, *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pp. 179–184.

Goffe, L. *et al.* (2021) ‘Appetite for Disruption: Designing Human-Centred Augmentations to an Online Food Ordering Platform’, in *34th British Human Computer Interaction Conference Interaction Conference, BCS HCI 2021*. BCS Learning and Development Ltd., pp. 155–167. Available at: https://doi.org/10.14236/ewic/HCI2021.16.

(n.d.). (no date) *Unsupervised Machine Learning*, *IBM*. Available at: https://www.javatpoint.com/unsupervised-machine-learning (Accessed: 15 May 2023).

R., A. *et al.* (2017) ‘Online Food Ordering System’, *International Journal of Computer Applications*, 180(6), pp. 22–24. Available at: https://doi.org/10.5120/ijca2017916046.

Ratto, F. *et al.* (2008) ‘A numerical approach to quantify self-ordering among self-organized nanostructures’, *Surface Science*, 602, pp. 249–258.

Sarwar, B.M. *et al.* (2001) ‘Item-based collaborative filtering recommendation algorithms’, in *The Web Conference*.

APENDIXES

Source code

haritha99ch/FoodDesire: PUSL3119 Computing Project (github.com)

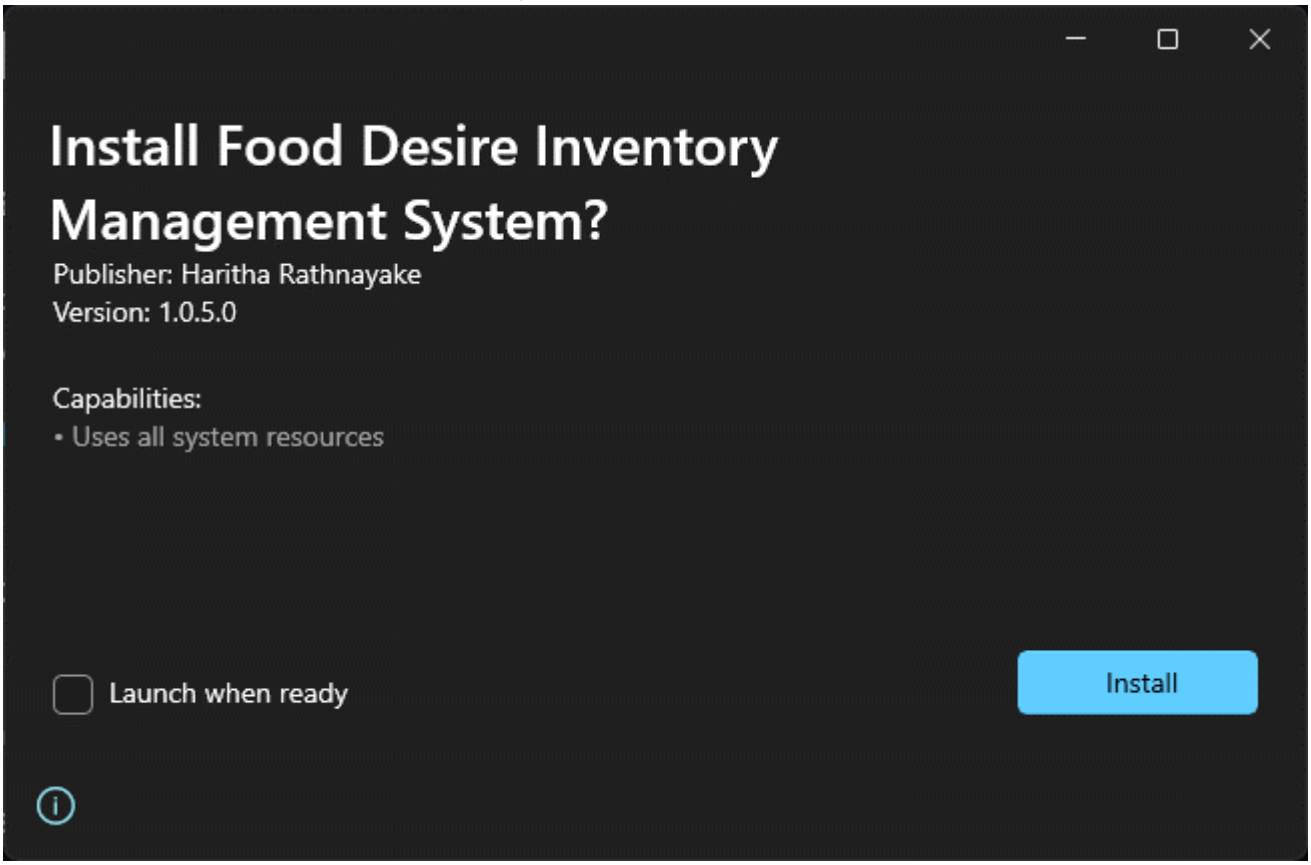
Web Application

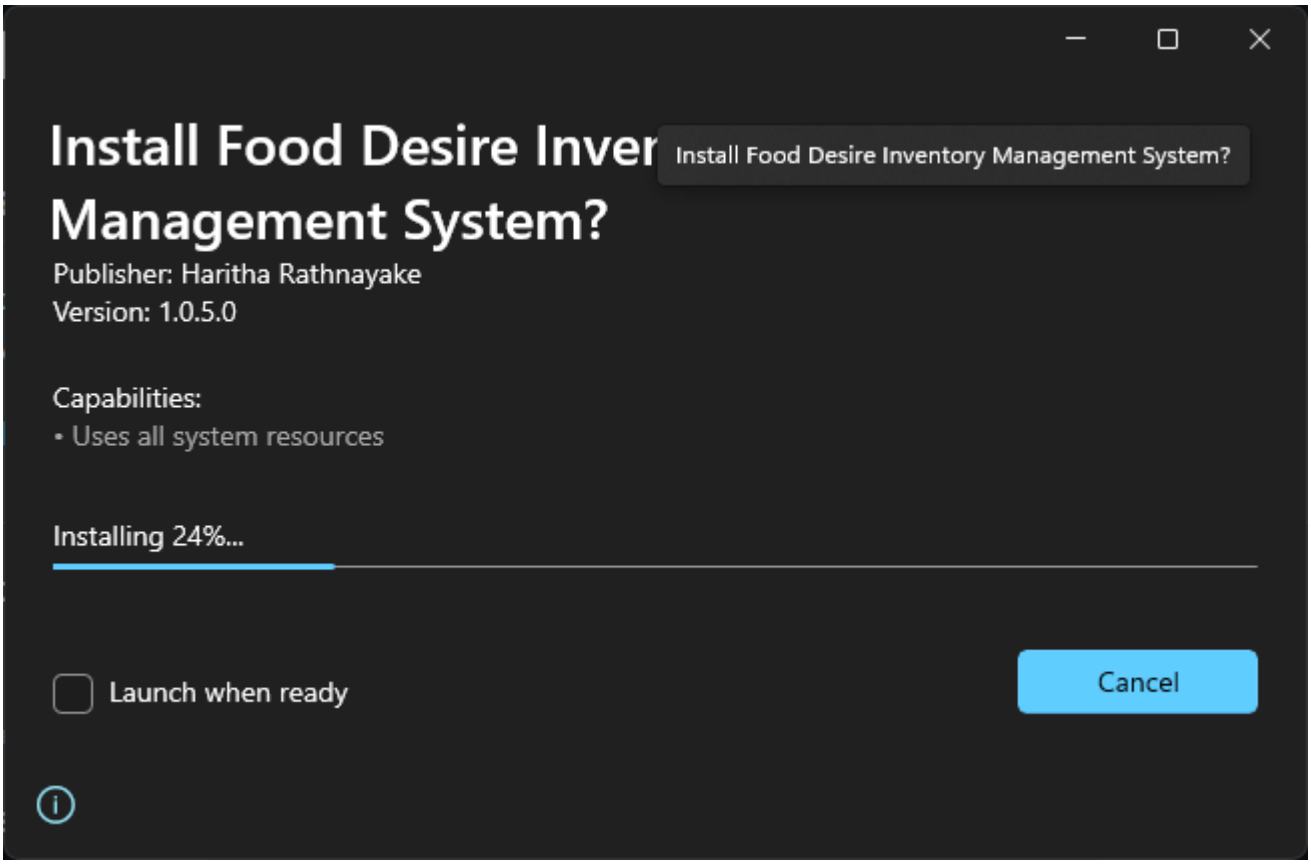
Food Desire (app-fooddesire-web-sea-dev-001.azurewebsites.net)

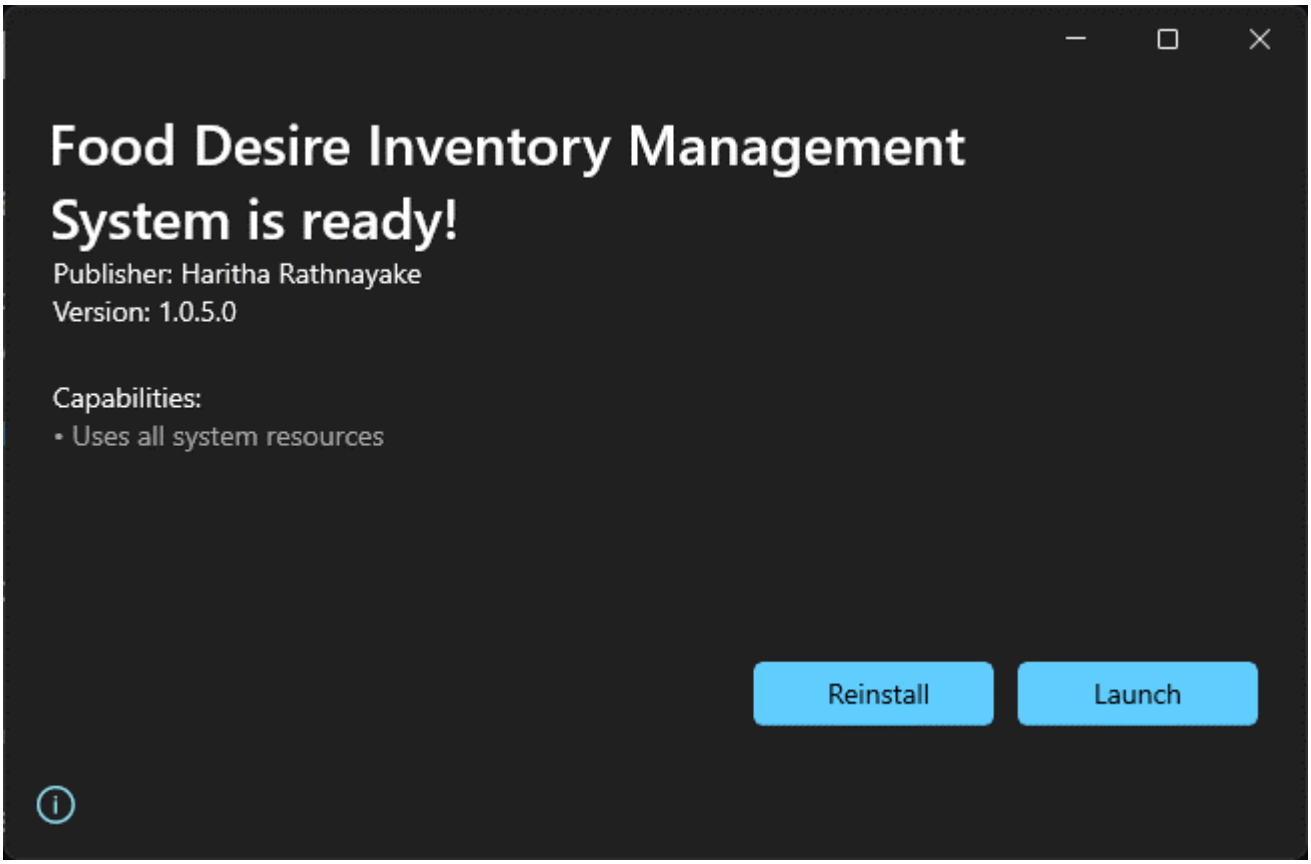
User Guide

Inventory Management System

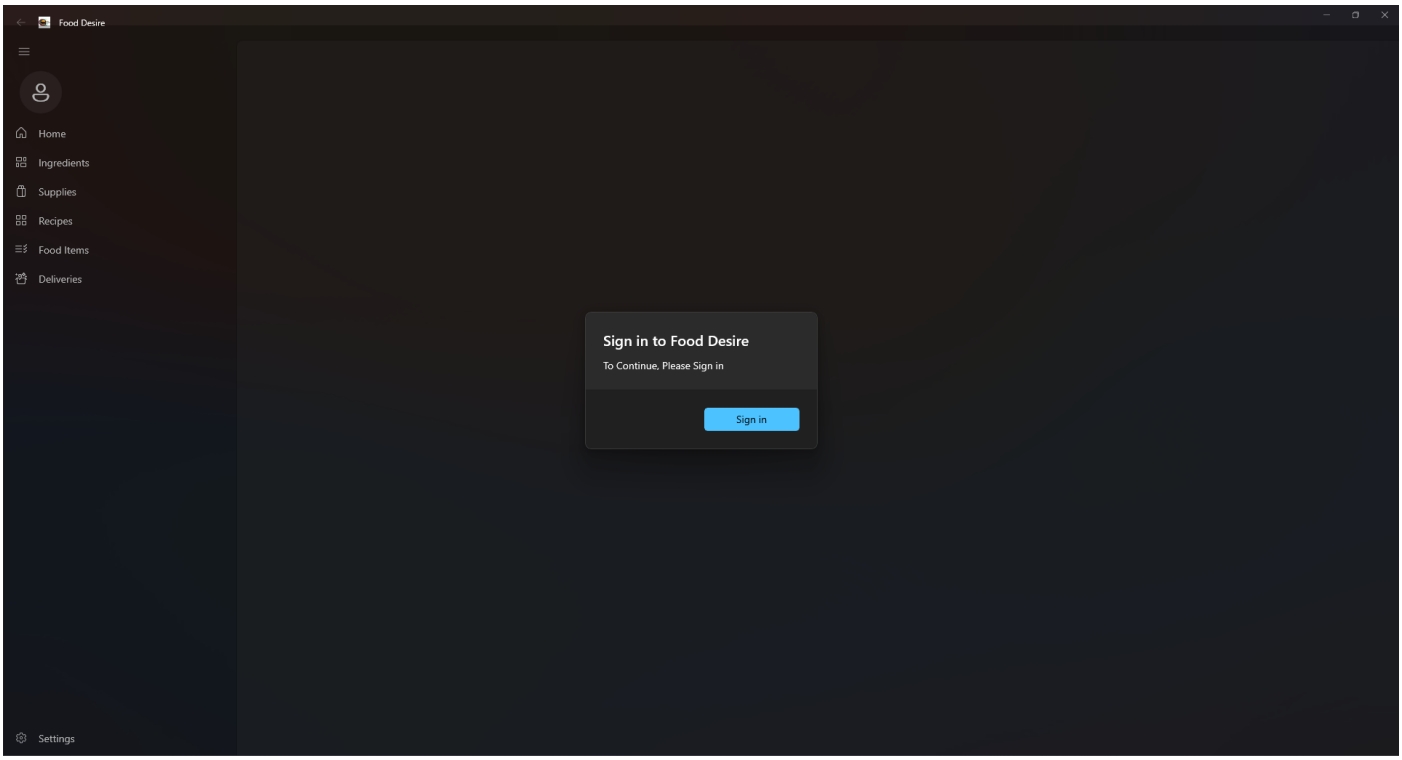
1. Download and Install: The MSIX file for the Inventory Management System can be downloaded from the project's GitHub repository releases tab. Once downloaded, the file needs to be installed on your device.

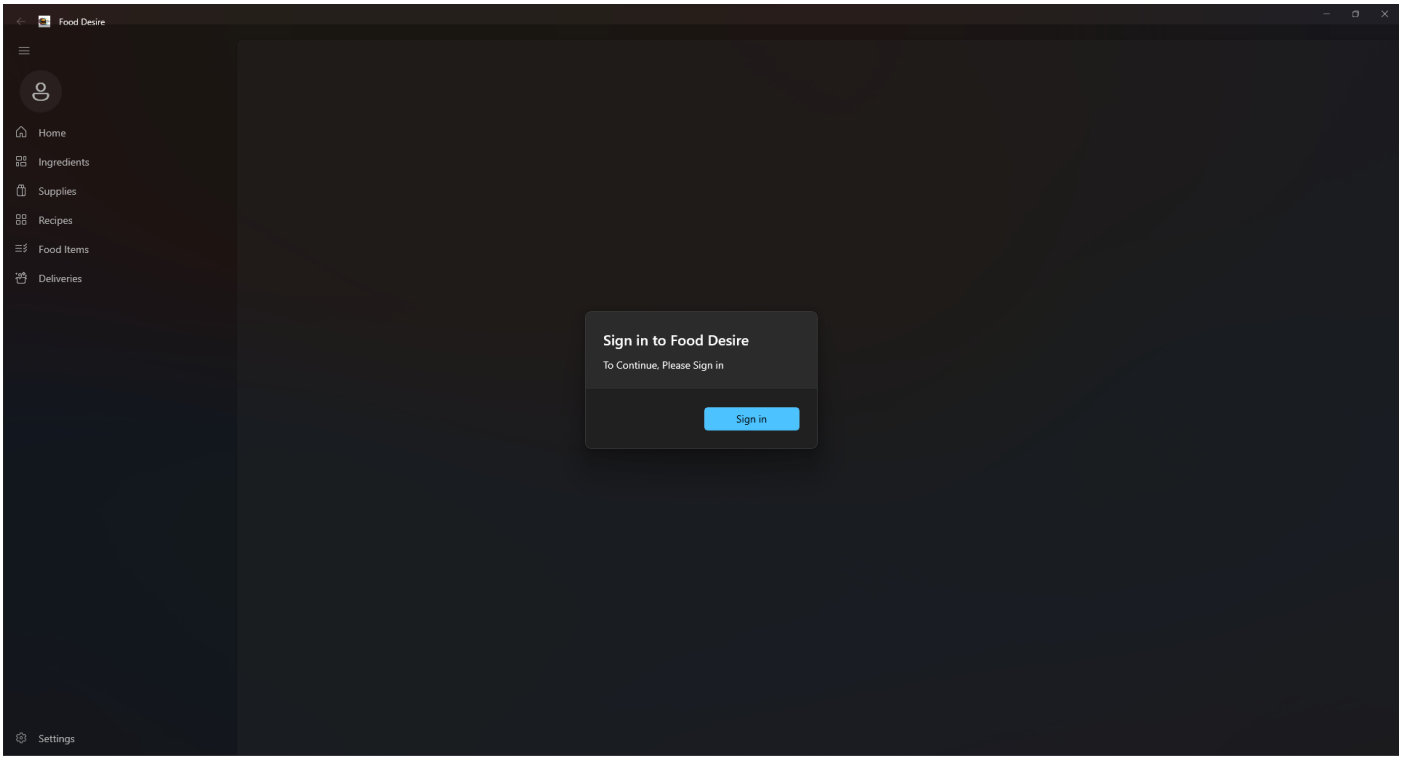






2. Initial Setup: After installation, the admin will be provided with a Microsoft account to sign into the app for the initial setup.





3. Employee Registration: The first step in using the Inventory Management System is to register employees. This can be done by navigating to the Employee View and entering the employee's information, including their name, role, and contact details.

4. Supply Details Management: The system allows users to manage supply details, including adding new suppliers and placing supply orders. This can be done by navigating to the Supply View and entering the necessary information.

5. Payment Management: The system also allows users to manage payments, including paying for supplies and employee salaries. This can be done by navigating to the Payment View and entering the necessary information.

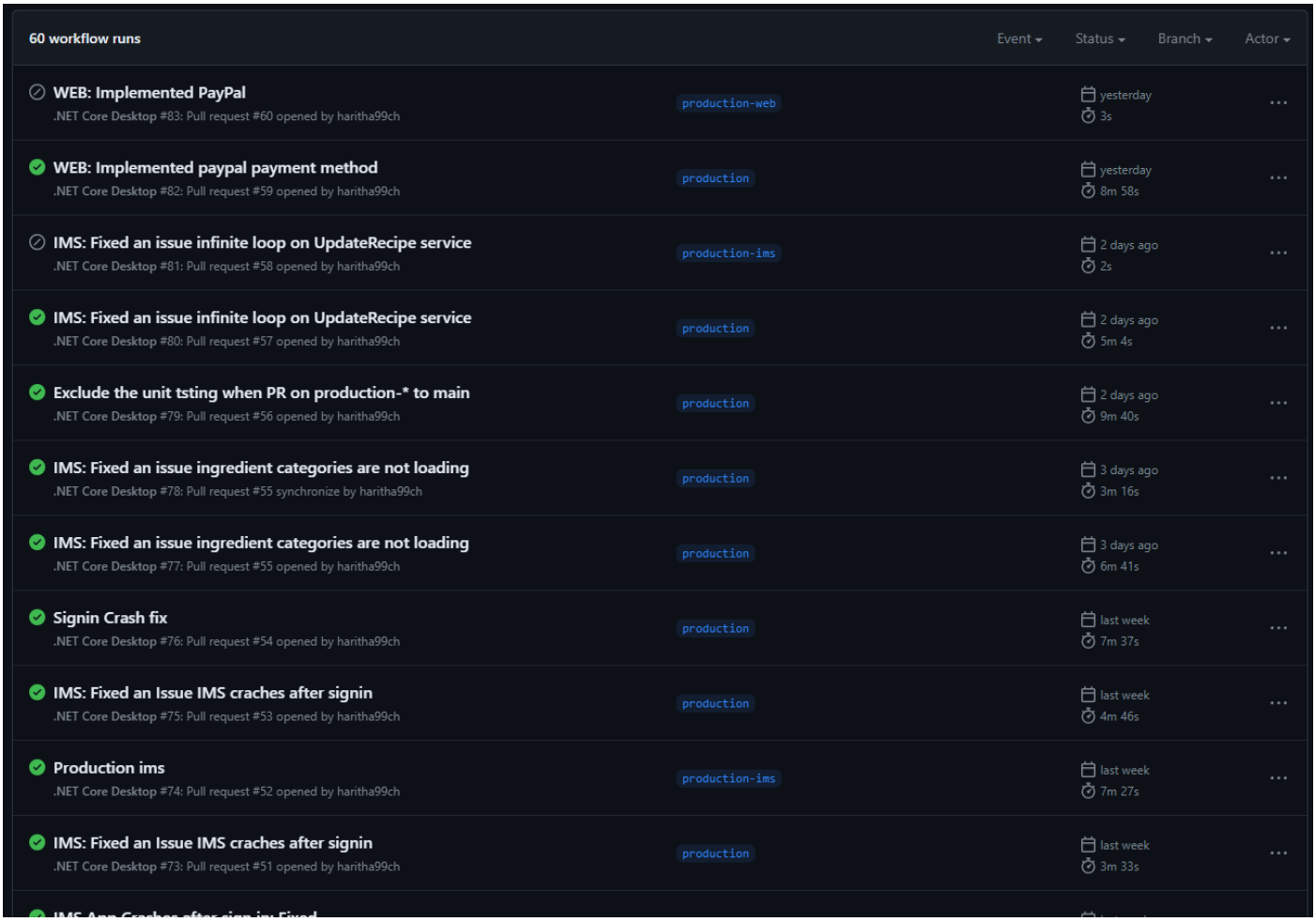
6. Food Item Recipe Creation: Chefs can use the system to create recipes for food items using ingredients available in the inventory. This can be done by navigating to the Recipe View and entering the necessary information, including ingredient quantities and cooking instructions.

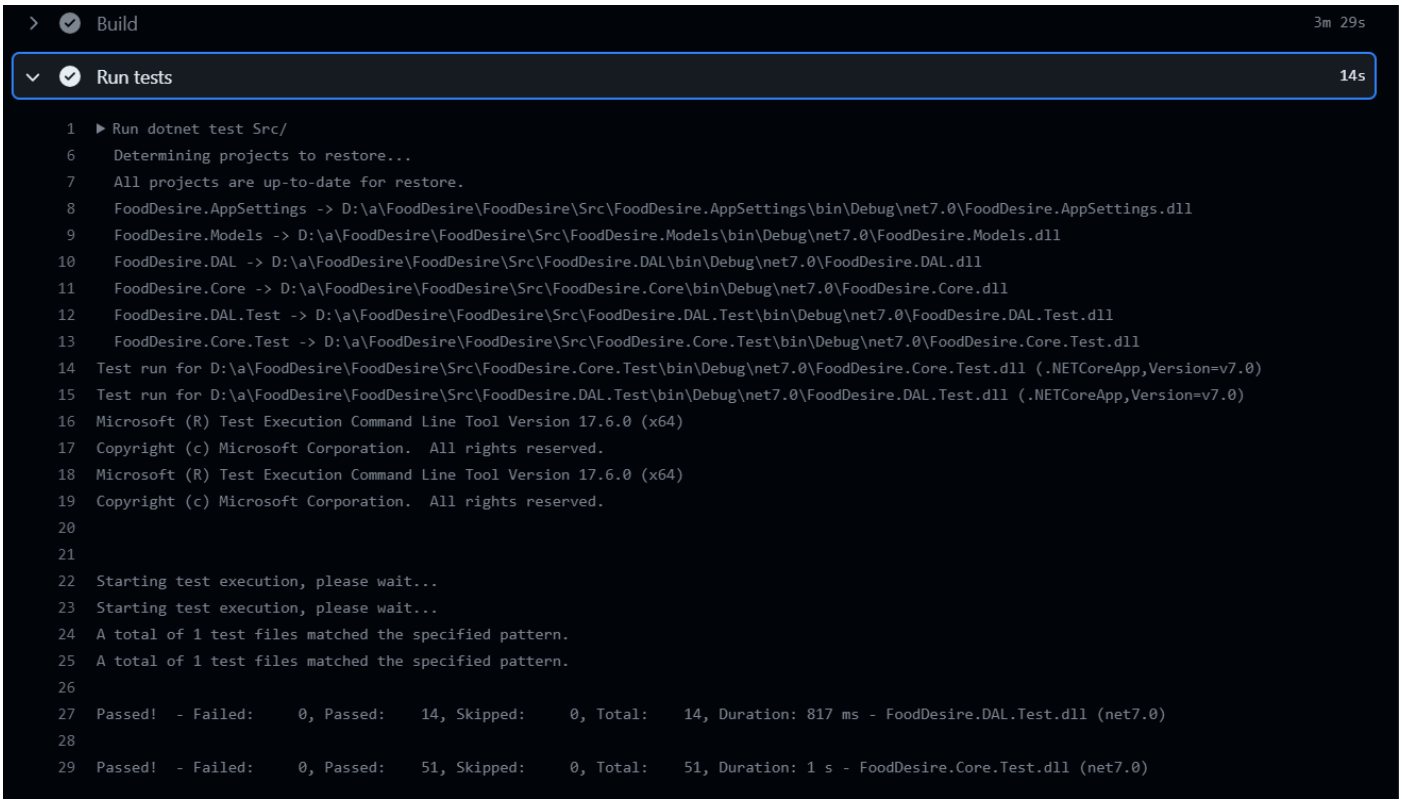
7. Order Management: The system allows chefs to see ordered food items and prepare them accordingly. This can be done by navigating to the Order View and viewing the list of pending orders.

8. Settings: The Settings View allows users to manage various settings for the application, including language preferences and notification settings.

Test Cases and Results

CD/CI Automated Test Results.




Build
3m 29s
Run tests
14s
Run dotnet test Src/
Determining projects to restore...
All projects are up-to-date for restore.
FoodDesire.AppSettings -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.AppSettings\bin\Debug\net7.0\FoodDesire.AppSettings.dll
FoodDesire.Models -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.Models\bin\Debug\net7.0\FoodDesire.Models.dll
FoodDesire.DAL -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.DAL\bin\Debug\net7.0\FoodDesire.DAL.dll
FoodDesire.Core -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.Core\bin\Debug\net7.0\FoodDesire.Core.dll
FoodDesire.DAL.Test -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.DAL.Test\bin\Debug\net7.0\FoodDesire.DAL.Test.dll
FoodDesire.Core.Test -> D:\a\FoodDesire\FoodDesire\Src\FoodDesire.Core.Test\bin\Debug\net7.0\FoodDesire.Core.Test.dll
Test run for D:\a\FoodDesire\FoodDesire\Src\FoodDesire.Core.Test\bin\Debug\net7.0\FoodDesire.Core.Test.dll (.NETCoreApp,Version=v7.0)
Test run for D:\a\FoodDesire\FoodDesire\Src\FoodDesire.DAL.Test\bin\Debug\net7.0\FoodDesire.DAL.Test.dll (.NETCoreApp,Version=v7.0)
Microsoft (R) Test Execution Command Line Tool Version 17.6.0 (x64)
Copyright (c) Microsoft Corporation. All rights reserved.
Microsoft (R) Test Execution Command Line Tool Version 17.6.0 (x64)
Copyright (c) Microsoft Corporation. All rights reserved.
Starting test execution, please wait...
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
A total of 1 test files matched the specified pattern.
Passed! - Failed: 0, Passed: 14, Skipped: 0, Total: 14, Duration: 817 ms - FoodDesire.DAL.Test.dll (net7.0)
Passed! - Failed: 0, Passed: 51, Skipped: 0, Total: 51, Duration: 1 s - FoodDesire.Core.Test.dll (net7.0)

API Testing Cases

Account Controller.

Test Case	Column2	Steps	Input Data	Expected Outcome
TC001	Index - Valid user	1. Send a GET request to /api/Account/Index. 2. Ensure the user is authenticated.		The API should return the account information for the valid user.
TC002	Index - Invalid user (missing email)	1. Send a GET request to /api/Account/Index. 2. Ensure the user is not authenticated.	-	The API should return a BadRequest response indicating an invalid user.
TC003	Index - User not found	1. Send a GET request to /api/Account/Index. 2. Ensure the user is authenticated. 3. Use a non-existing email for the authenticated user.	-	The API should return a BadRequest response indicating that the user doesn't exist.
TC004	SignUp - Successful signup	1. Send a POST request to /api/Account/SignUp. 2. Provide valid user data in the request body.	User object with valid data	The API should create a new account for the user and return a Customer object.
TC005	SignUp - Invalid user data	1. Send a POST request to /api/Account/SignUp. 2. Provide incomplete or invalid user data in the request body.	User object with incomplete/invalid data	The API should return a BadRequest response if the provided user data is incomplete or invalid.
TC006	SignUp - Existing email	1. Send a POST request to /api/Account/SignUp. 2. Provide an email that is already registered in the system.	User object with existing email	The API should return a BadRequest response if the provided email is already registered.
TC008	SignIn - Incorrect credentials	1. Send a POST request to /api/Account/SignIn. 2. Provide incorrect email and/or password in the request body.	SignIn object with incorrect email and/or password	The API should return a BadRequest response indicating incorrect credentials.
TC009	Edit - Successful account update	1. Send a PATCH request to /api/Account/Edit. 2. Provide valid customer data in the request body. 3. Ensure the user is authenticated.	Customer object with valid data	The API should update the user's account information and return the updated Customer object.
TC010	Edit - Invalid customer data	1. Send a PATCH request to /api/Account/Edit. 2. Provide incomplete or invalid customer data in the request body. 3. Ensure the user is authenticated.	Customer object with incomplete/invalid data	The API should return a BadRequest response if the provided customer data is incomplete or invalid.

TC011	Edit - User not found	1. Send a PATCH request to /api/Account/Edit. 2. Provide valid customer data in the request body. 3. Ensure the user is authenticated. 4. Use a non-existing email for the authenticated user.	Customer object with valid data	The API should return a BadRequest response indicating that the user doesn't exist.
TC012	Delete - Successful account deletion	1. Send a DELETE request to /api/Account/Delete. 2. Ensure the user is authenticated.	-	The API should delete the user's account and return true indicating success.
TC013	Delete - User not found	1. Send a DELETE request to /api/Account/Delete. 2. Ensure the user is authenticated. 3. Use a non-existing email for the authenticated user.	-	The API should return a BadRequest response indicating that the user doesn't exist.

Cart Controller

Test Case	Column2	Steps	Input Data	Expected Outcome
TC001	Index - Valid user	1. Send a GET request to /api/Cart/Index. 2. Ensure the user is authenticated.	-	The API should return the pending order for the authenticated user.
TC002	Index - Invalid user (missing user ID)	1. Send a GET request to /api/Cart/Index. 2. Ensure the user is not authenticated.	-	The API should return a BadRequest response indicating that the user could not be found.
TC003	Details - Valid order	1. Send a GET request to /api/Cart/Details/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated.	Valid order ID	The API should return the details of the specified order if the user is authorized to view it.
TC004	Details - Invalid user (missing user ID)	1. Send a GET request to /api/Cart/Details/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is not authenticated.	Valid order ID	The API should return a BadRequest response indicating that the user could not be found.
TC005	Details - Unauthorized user	1. Send a GET request to /api/Cart/Details/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated. 3. Use an order ID that doesn't belong to the authenticated user.	Order ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to view the order.
TC006	CreateOrder - Successful order creation	1. Send a POST request to /api/Cart with a valid order object in the request body.	Valid Order object	The API should create a new order and return the created order object.
TC007	Pay - Successful payment	1. Send a PATCH request to /api/Cart/Pay/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated.	Valid order ID	The API should process the payment for the order and return a success message or relevant payment information.
TC008	Pay - Invalid user (missing user ID)	1. Send a PATCH request to /api/Cart/Pay/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is not authenticated.	Valid order ID	The API should return a BadRequest response indicating that the user could not be found.
TC009	Pay - Unauthorized user	1. Send a PATCH request to /api/Cart/Pay/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated. 3. Use an order ID that doesn't belong to the authenticated user.	Order ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to pay for the order.

TC010	CompletePayment - Successful payment completion	1. Send a PATCH request to /api/Cart/CompletePayment with a valid order object in the request body.2. Ensure the user is authenticated.	Valid Order object	The API should update the order's payment status and return the updated order object.
TC011	CompletePayment - Invalid user (missing user ID)	1. Send a PATCH request to /api/Cart/CompletePayment with a valid order object in the request body. 2. Ensure the user is not authenticated.	Valid Order object	The API should return a BadRequest response indicating that the user could not be found.
TC012	CompletePayment - Unauthorized user	1. Send a PATCH request to /api/Cart/CompletePayment with a valid order object in the request body. 2. Ensure the user is authenticated. 3. Use an order ID that doesn't belong to the authenticated user.	Order object with an ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to complete payment for the order.
TC013	Update - Successful order update	1. Send a PATCH request to /api/Cart/Update with a valid order object in the request body. 2. Ensure the user is authenticated.	Valid Order object	The API should update the order and return the updated order object.
TC014	Update - Invalid user (missing user ID)	1. Send a PATCH request to /api/Cart/Update with a valid order object in the request body. 2. Ensure the user is not authenticated.	Valid Order object	The API should return a BadRequest response indicating that the user could not be found.
TC015	Update - Unauthorized user	1. Send a PATCH request to /api/Cart/Update with a valid order object in the request body. 2. Ensure the user is authenticated. 3. Use an order ID that doesn't belong to the authenticated user.	Order object with an ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to update the order.
TC016	Cancel - Successful order cancellation	1. Send a DELETE request to /api/Cart/Cancel/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated.	Valid order ID	The API should cancel the order and return true indicating success.
TC017	Cancel - Invalid user (missing user ID)	1. Send a DELETE request to /api/Cart/Cancel/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is not authenticated.	Valid order ID	The API should return a BadRequest response indicating that the user could not be found.

TC018	Cancel - Unauthorized user	1. Send a DELETE request to /api/Cart/Cancel/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated. 3. Use an order ID that doesn't belong to the authenticated user.	Order ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to cancel the order.
TC019	GetFoodItems - Successful retrieval	1. Send a GET request to /api/Cart/GetFoodItems/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is authenticated.	Valid order ID	The API should return a list of food items associated with the order if the user is authorized to view them.
TC020	GetFoodItems - Invalid user (missing user ID)	1. Send a GET request to /api/Cart/GetFoodItems/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is not authenticated.	Valid order ID	The API should return a BadRequest response indicating that the user could not be found.
TC020	GetFoodItems - Invalid user (missing user ID)	1. Send a GET request to /api/Cart/GetFoodItems/{orderId} where {orderId} is a valid order ID. 2. Ensure the user is not authenticated.	Valid order ID	The API should return a BadRequest response indicating that the user could not be found.
TC022	RemoveFoodItem - Successful removal	1. Send a DELETE request to /api/Cart/RemoveFoodItem/{foodItemId} where {foodItemId} is a valid food item ID. 2. Ensure the user is authenticated.	Valid food item ID	The API should remove the specified food item from the order and return true indicating success.
TC023	RemoveFoodItem - Invalid user (missing user ID)	1. Send a DELETE request to /api/Cart/RemoveFoodItem/{foodItemId} where {foodItemId} is a valid food item ID. 2. Ensure the user is not authenticated.	Valid food item ID	The API should return a BadRequest response indicating that the user could not be found.
TC024	RemoveFoodItem - Unauthorized user	1. Send a DELETE request to /api/Cart/RemoveFoodItem/{foodItemId} where {foodItemId} is a valid food item ID. 2. Ensure the user is authenticated. 3. Use a food item ID that doesn't belong to the authenticated user's order.	Food item ID of a different user	The API should return a BadRequest response indicating that the user is not authorized to remove the food item.

Home Controller

Test Case	Column2	Steps	Input Data	Expected Outcome
TC001	Index - Get top 10 recipes	1. Send a GET request to /api/Home/Index.	None	The API should return the top 10 recipes from _homeControllerService.
TC002	Index - Get predicted recipes for authenticated user	1. Send a GET request to /api/Home/Index.	User ID in claims	The API should return the predicted recipes for the authenticated user based on their user ID.
TC003	Index - No recipes available	1. Send a GET request to /api/Home/Index.	None	The API should return an empty list of recipes.

Order Controller

Test Case	Column2	Steps	Input Data	Expected Outcome
TC001	Index - Get orders for an authenticated user	1. Send a GET request to /api/Order/Index.	User ID in claims	The API should return a list of orders for the authenticated user based on their user ID.
TC002	Index - Unauthorized access	1. Send a GET request to /api/Order/Index.	No user ID	The API should return a BadRequest response with the message "Could not find user!" since no user ID is available.
TC003	Detail - Get order details for an authenticated user	1. Send a GET request to /api/Order with the orderId parameter.	User ID in claims	The API should return the order details if the order belongs to the authenticated user.
TC004	Detail - Unauthorized access	1. Send a GET request to /api/Order with the orderId parameter.	No user ID	The API should return a BadRequest response with the message "Could not find user!" since no user ID is available.
TC005	Detail - Order not found	1. Send a GET request to /api/Order with a non-existing orderId parameter.	User ID in claims	The API should return a BadRequest response with the message "Order not found" since the order does not exist.
TC006	Detail - Unauthorized access to another user's order	1. Send a GET request to /api/Order with the orderId parameter of another user's order.	User ID in claims	The API should return an Unauthorized response with the message "You are not authorized to view this" since the order does not belong to the authenticated user.

Recipe Controller

Test Case	Column2	Steps	Input Data	Expected Outcome
TC001	Index - Get all recipes	1. Send a GET request to /api/Recipe.	-	The API should return a list of all recipes available.
TC002	Index - Get recipes by search keyword	1. Send a GET request to /api/Recipe with the search parameter set to a valid search keyword.	Valid search keyword	The API should return a list of recipes that match the search keyword.
TC003	Index - No search keyword provided	1. Send a GET request to /api/Recipe without providing a search keyword.	Empty search keyword	The API should return a list of all recipes available since no search keyword is provided.
TC004	Details - Get recipe details	1. Send a GET request to /api/Recipe/{recipeId} with a valid recipeId.	Valid recipe ID	The API should return the details of the specified recipe.
TC005	AddToCart - Add food item to the cart	1. Send a POST request to /api/Recipe/AddToCart with a valid FoodItem object containing all required data.2. Include the authentication token in the request header.	Valid FoodItem object	The API should add the food item to the cart and return the added FoodItem object.
TC006	AddToCart - Unauthorized access to add food item to another user's cart	1. Send a POST request to /api/Recipe/AddToCart with a valid FoodItem object containing all required data.2. Include the authentication token of a different user in the request header.	Valid FoodItem object	The API should return a BadRequest response with the message "You are not authorized to perform this action!" since the user is not authorized to add food items to another user's cart.
TC007	Reviews - Get reviews for a recipe	1. Send a GET request to /api/Recipe/Reviews with the recipeId parameter set to a valid recipe ID.	Valid recipe ID	The API should return a list of reviews for the specified recipe.
TC008	Review - Add a review for a recipe	1. Send a POST request to /api/Recipe/Review with a valid RecipeReview object containing all required data.2. Include the authentication token in the request header.	Valid RecipeReview object	The API should add the review for the recipe and return the added RecipeReview object.
TC009	Review - Unauthorized access to add a review	1. Send a POST request to /api/Recipe/Review with a valid RecipeReview object containing all required data.2. Include the authentication token of a different user in the request header.	Valid RecipeReview object	The API should return a BadRequest response with the message "You are not authorized to perform this action!" since the user is not authorized to add a review for the recipe.

TC010	Review - Add a review without authentication	1. Send a POST request to /api/Recipe/Review with a valid RecipeReview object containing all required data, without including the authentication token in the request header.	Valid RecipeReview object	The API should return an Unauthorized response since the user is not authenticated.

Project Initiation Document

PUSL3119 Computing Individual Project

Project Initiation Document (PID)

Food Desire
An Online Food Ordering System

Supervisor: Mr. Pramudya Thilakarathne
Name: Rathnayake M Rathnayake
Plymouth Index Number: 10747887
Degree Program: BSc (Hons) Computer Science

Table of Contents

TABLE OF FIGURE

Chapter One
INTRODUCTION

1.1 BACKGROUND

Now days it is possible to get anything to doorsteps by ordering goods via the internet, faster and secure. The most important thing is the time that is spared. Because of this people do not want to spend time going to buy the goods by themselves.

Most of the fast-food restaurants in the world are adopting to the online food ordering. This is the latest trend of services that can offered by restaurant. This method allows customers to order food online and delivered to the customer. With an implementation of electronic payment system this method can be achieved. So, from a system like this will enable customers to go online and order their food by paying from their cards.

Because of the technologies that are associated with the food industry many opportunities are coming up due to the amount of internet usage. So many business industries value the use of internet to promote their business. One of such opportunities is an online food ordering system. Many fast-food restaurants have chosen this business opportunity to focus more on food preparation and faster delivery.

1.2 PROBLEM STATEMENT

Because of the food industry is expanding, people are looking for many more ways to improve the current ways. One such way is the arability of customizing the food as they prefer. But this is somewhat limited to fast food industries such as Pizza restaurants, even though it cannot be customize as expected. Customers only be able to browse through the menu and select the meal and order. Customers are restricted to order the meal as it is. Customers cannot see the amount of ingredients that they would like to consider. No allowing customers to consider about the ingredient in their mela is a big disadvantage in the online food ordering system.

1.3 CURRENT STATE OF THE SYSTEM

The online food ordering system sets up a food menu online and customers can easily place the order as per they like. Also, the online customers can easily track their orders(R. *et al.*, 2017). Then the restaurant delivery employees can deliver the meal to the customers. This system also provides a feedback system in which user can rate the food items. Also, the proposed system can recommend hotels, food(R. *et al.*, 2017). This will allow the transparency of the restaurant, so more customers can pay attention to the. The payment can be made online or cash or pay-on-delivery system. For more secured ordering separate accounts are maintained for each user by providing them an ID and a password(R. *et al.*, 2017).

Most of the current online food ordering systems has most secure ways to make payment through online and place order for customers' orders. Theses system may include booth mobile and web applications improve the user experience.

1.2 SOLUTION

The proposed solution is to improve customer experience by adding a feature to the existing online food ordering system that enable customers to see most of the ingredients of their meal and edit amount of ingredient as customers prefer. Customers can browse through the meal items and select a meal that they prefer and increase or decrease the amount of ingredients that contains in the meal. To enable this feature, the inventory management system of the restaurant also needs to be modified. It is up to the restaurant end to make a list of ingredients for each meal items. So, the restaurant end experience also needs an update.

This solution will update both customer and restaurant end to enable this feature. So, the restaurant end can provide which ingredients are used in the meal to the customer, and customers can edit the ingredients as they prefer. Customer will be able to select their preferred meal item, before going to the checkout or cart, customer will be promoted with a page that includes the meal information. Customer can choose either default meal item that is provided by the restaurant or edit the amount of ingredients that contains in the meal. As for an example, if the customer would like to have more "Chicken pieces" in their Fried rice, customer can increase the amount of chicken pieces from the ingredient list.

1.3 PROJECT OUTCOME

Implementing this solution in a restaurant will allow their customers to value the food they are eating more than the existing food ordering system. As customer knows that they are paying. This will also enable the transparency between the customer and the food as customer knows what they are having for their meal. This will also be a big health benefit for the customers because they can be aware of the ingredients the are having. Depend on their health needs, customers can enjoy their favorite meals by limiting the amount of ingredient they should not be having too much.

This solution comes up with an upgrade for the inventory management system for the restaurant as well. Restaurant will be able create recipes from the inventory management system directly by accessing the ingredient categories. When a meal is being prepared, the system will update the amount of ingredients that has been consumed depends on the order that received. This will improve the inventory management experience for the restaurant.

Since customer pays for meal item depend on the amount of ingredients for each meal, will benefits booth financial and food reserve, because both customer and restaurant end will save money and limit the food wastage.

Chapter Two
PROJECT OBJECTIVES

This system is for an imaginary restaurant called “Food Desire”. As the name implies, the customer should be able to customize their meal as they desire. To implement the solution the system must be a brand-new system. An inventory management system is mandatory to deliver the system. So, this project will be a distributed system. A recommendation system for the customer, will also be developed.

2.1 RESTAURANT MANAGEMENT SYSTEM

The restaurant management system will contain multiple functionalities. One of the most important functionalities is the inventory management system that enable the main goal of the solution, the availability of allowing customers to edit the amount of ingredient for their meal. Restaurant can manage the recipe from the same system instead of login into the system that customers login and enables the ingredient modification feature by providing the ingredients list. Restaurant should be able to give a threshold limit to avoid users to overdose ingredients. As an example, if a customer increases the amount of sault more than usual, the taste would be terrible. So, the restaurant can limit amount of ingredients for each one. This system will also monitor each ingredient categories in the inventory and update for each order that received for the restaurant. This will help the restaurant to manage every ingredient manually. This functionality will benefit in financial and time management,

Since this system is an enhanced system of an existing food ordering system, this will include all the functionalities of an ordinary online food ordering system. So, there will be multiple type of employees that work for the restaurant, such as chefs, delivery persons, supply persons and system admins who handles this system. So having a feature to manage these employees in the same system will be great opportunity to enhance the system furthermore. All the employees will be categorized into roles and give access to certain features. This will allow employees to work more efficiently.

This system will be developed by using .NET and this will be a desktop application. Desktop platform is chosen, because it is easier to manage by the chef and other general employees as well. .NET is a very optimized developing platform, and it is the most suitable platform to develop this application. WinUi3 will be the framework to develop the application as this is the most modest and newest framework developed on .NET platform. So, the system will be modern looking and well up to dated application.

2.2 WEB APPLICATION

A web application will be developed for the customer to order their meal as a usual online food ordering system. A new feature will be added to the proposed system that customers able to customize their meal as they desire. Customers will be able to register and login to the system via web browser and start browsing through menu items. Once the customer selects a meal item, they will be prompted a partial page that shows the ingredients that contain in the meal. As for an example, “Chicken Fried Rice” will contain the following key ingredients.

- Rice
- Chicken pieces.
- Garlic
- Ginger
- Onions
- Type of sauces (soya/ fish)
- Green garnish

Once the list of ingredients is displayed to the customer, they will be able to edit the amount. The unit of amount can be differed to another. Customer can procced in either ways, order the default meal that the restaurant have already made or edit the amount as they want and order the meal. So, if the customer does not want to be bother by the ingredients, they can just order the default meal. Customers are not restricted to order in one way. With this implement, customers have freedom to order their meal as they want.

This web application will also be developed using .NET because the same share libraries that will developed to use in the desktop application can be used in this web application as well. .NET MVC with blazorUI framework will be used to develop this web application as will contains web components that has complex

functionalities. With blazorUi it can be overcome by individually program each component

2.3 RECOMMENDATION SYSTEM

A recommendation system will also be implemented for the web application to recommend users with meal items that they may like, depend on the data is collected by each user. Implementation of this system will enhance the user experience by suggesting meal items that they might prefer instead the default meals list in the home page. A collaborative filtering Machine Ai will be used to develop this system.

Having a recommendation system in the food ordering system will benefit both customer and the restaurant as customer will be suggested more meal items that they might like, and the restaurant get more orders than usual.

Chapter Three
BUSINESS CASES

3.1 BUSINESS NEEDS

The modern business industries are involving with the modern technology. Started by ordering goods by telephone, now it is at state that customers can click few buttons to order their needs People can order anything online via internet any time and get them delivered to the doorstep. Having such system in food industry will defiantly improve the industry as well. So, improving these kinds of systems only makes this system more efficient and more valuable.

The ability to customize customers' meals is a good benefit for both customer and restaurant. Because of the uniqueness of this improvement will benefit in restaurant business, as more and more customers will pay attention to the restaurant more than ever. Since customers can customize their meals according to their taste is a great opportunity that the customers can get.

The proposed improvement will have the following features for the restaurant.

1. Have a good understand of the customer's taste.
2. Get more customer as the restaurant allowing customers to customize the recipes.
3. Ability to monitor individual ingredients that are consumed more.
4. Having a separate application to manage restaurant and the inventory.
5. Manage employees and work with them in the same system.

As for the customers,

1. Transparency of the meals' customers having.
2. Make the meal in customer style.
3. Recommend more meal items as customers order depends on the ingredients.

3.2 BUSINESS OBJECTIVES

Since online food ordering systems are in most of the business industries for long time, improving the system according to their customer needs is a great opportunity. As people seeking more ideas this solution is a huge improvement. Having such a system will make a new connection between customers and the restaurant. Customers will have the freedom to have their favorite meal according to their taste.

When customer select a meal, they will see the list of ingredients that are contained in the meal. So, the customer knows what kind of things they are eating. So, this is a good opportunity to promote how trust full the restaurant is. This will bring online food ordering for the people who are concerned about the food they are eating. Meals ordered online for home delivery are typically less healthy than home-made meals, potentially contributing to weight gain(Dana *et al.*, 2021). Having an online food ordering system such, will allow customers to order food by limiting the ingredients that they should not have too much.

When it comes to the recommendation system, most of the e commers businesses have already implemented this as well as food business. The improvement of the proposed system is that the recommendation system can gather more data by analyzing the amount of ingredients in the food items. So, this is a more efficient way to improve the recommendation system, and this will benefit both restaurant and the customer.

Chapter Four
LITERATURE REVIEW

4.1 INTRODUCTION

Ordering systems can be identified as a set of detailed instructions that are used to handle an ordering process. It could either computerized or manual. Self-ordering means that the customer can order food by themselves.

Self-ordering system can be identified as either computerized or manual system. In computerized system, the system can track the orders that being placed by the customer.

4.2 SELF-ORDERING

This is the main point that Online ordering comes to the industries. Including various types of methods like such as internet, self-ordering systems have become a success in the industry. The evaluability of having such ordering technology is proven by many researchers.

(Ratto *et al.*, 2008)suggests that self-ordering systems are valuable because they can reach a state of maximal overall 'success'. (Gao and Su, 2018)found that self-order techniques can significantly reduce waiting cost and increase demand. People hate waiting to order their needs for so long. It is very time consuming. So, people prefer something like “do it by themselves” Therefore they like self-ordering technology. It can be also done by a text massage, email, telephone call and internet.

(Thomas and Davis, 1978)found that a traditional ordering technique can be replaced by timesaving ordering system. All these findings suggest that self-ordering systems can save businesses money by reducing costs and increasing efficiency. Because of these techniques people can order anything faster and get them delivered to their doorstep. The only waiting cost is wait for the goods to be delivered. In the business prospective, more order will receive for the business as self-ordering does not need to be monitored like manual ordering. This will improve the demand for the business.

When it comes to the online-food ordering, there are more benefits than the common business can be get in the food industry. This is mainly focus on the Internet. (Fujita, Shimada and Sato, 2014) found that the self-ordering system of restaurants can be personalized using the customer's user information. This research also suggests that self-ordering systems may be helpful in the food industry. So, implementing this on the internet/ online completely will improve the entire business.

4.3 ONLINE FOOD ORDERING

This is the most modern method to implement a self-ordering system for a restaurant. online food ordering is a suitable way for customers to order food. (R. *et al.*, 2017)suggests that online food ordering systems are efficient and easy to use. Customers can browse through the food items that the restaurant provide and order the food with few steps. (Daim *et al.*, 2013)found that customers concern the efficiency of the website, speed of response duration on online services, and the quality of the personnel to be the most important factors when choosing an online food service.

But there are not any major improvements that have been made up to today. The system can be upgraded in many ways. (Zulkarnain *et al.*, 2015)found that website quality and service quality are key success factors of online food ordering services. (Goffe *et al.*, 2021) suggests that online food ordering platforms need to be more human centered. Customers should have been given more freedom when they order their food.

So, therefore the solution in this project will help to improve the existing food ordering system.

4.4 AI AND MACHINE LEARNING IN E-COMMERCE

Artificial Intelligence means the simulation of human intelligence that is programed to think like a human being. Simply, that's mean "Programming the

Unprogrammable". The Core components of Artificial Intelligence are Machine learning techniques.

So, there are many useful scenarios that Ai and machine learning can be used for E-commerce. E-commerce is a business model that is used to exchange goods and services by the means of internet or other computerized systems. So, any kind of computerized system can include an Ai model.

The most common AI model that is used in E-commerce domain is recommendation system. (Ahrens, 2012)found that recommendation systems keep customers on the business site longer, so they interact with more products, and it suggests more products to customer, so they are likely to purchase or engage with. Recommendation systems are a model of artificial intelligence that are used to predict what users may want to buy or watch. They are used to recommend more items to users based on their past behavior.

(Sarwar *et al.*, 2001)found that item-based collaborative filtering recommendation algorithms provide dramatically better performance than user-based algorithms. This is the algorithm that is initially decided to use in the solution of this project.

4.5 SUPERVISED MACHINE LEARNING

To understand about recommendation system, having a good understanding on supervised machine learning will help more. (Singh, Thakur and Sharma, 2016)Supervised learning is a process of using known data to predict unknown data. It is also a process of using known data to classify unknown data. Supervised learning is a branch of machine learning that is in use in almost all fields. (Mishra, 2019) Semi-supervised learning is a learning pattern that is agitated by the study of how computers learn as human of both labeled and unlabeled data.

When it comes to label and unlabeled data, are two methods which are used to train an AI. Labeled data means, that the machine learning model knows the values to be predicted so the machine learning model can train itself by modified the model's variables for each new observation.

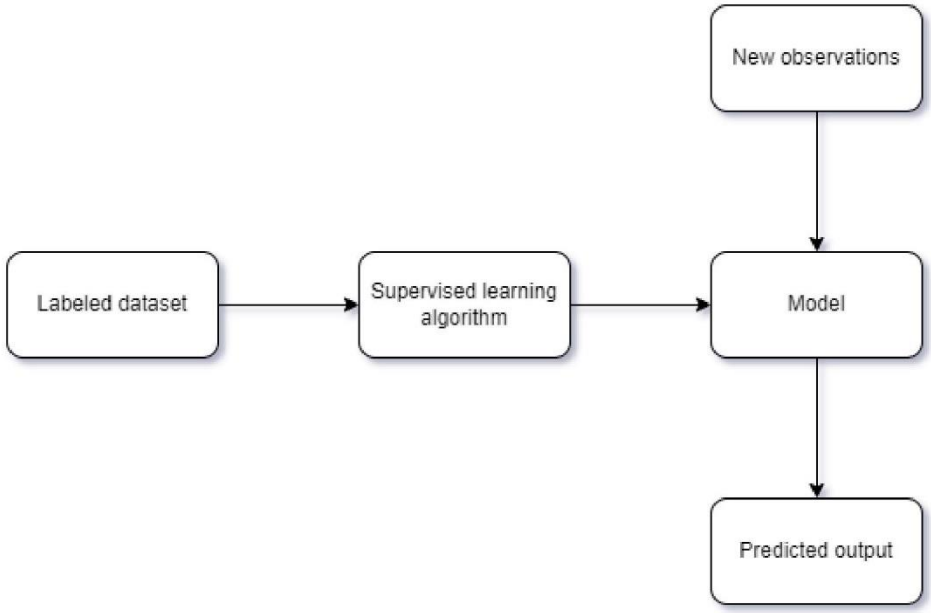


Figure 4. 1 Supervised ML

Supervised machine learning models can be categorized in to two which are Classification model and Regression models. Classification models predict the outputs in binary form such “Yes”, “No”. Regression models are used for problems that predict variables like a number, salary, or weight. So, it is used to predict numerical outputs based on the observations.

Chapter Five
METHOD OF APPROACH

5.1 INTRODUCTION

The proposed project is a distributed system that contains multiple projects. To develop a distributed system, keeping good design practices can help the development in many ways. The system has an inventory system and a web application. The web application is mostly depending on the inventory system as, the contents that rendered by the web application is dependent on the inventory system. Booth of the system will share multiple libraries to avoid any duplication coding. The project will follow a good application architecture practice to achieve a perfect system.

The entire project is built and deployed on the .NET framework. NET is a developer platform with tools and libraries for building any type of app, including web, mobile, desktop, games, IoT, cloud, and microservices. Throughout the development many share libraries and components will be used. To make the sharable component compatible with another, deciding to develop on the same platform is a good approach.

5.2 DESIGN

This distributed system has multiple actors so there are unique types of interactions for each actor.

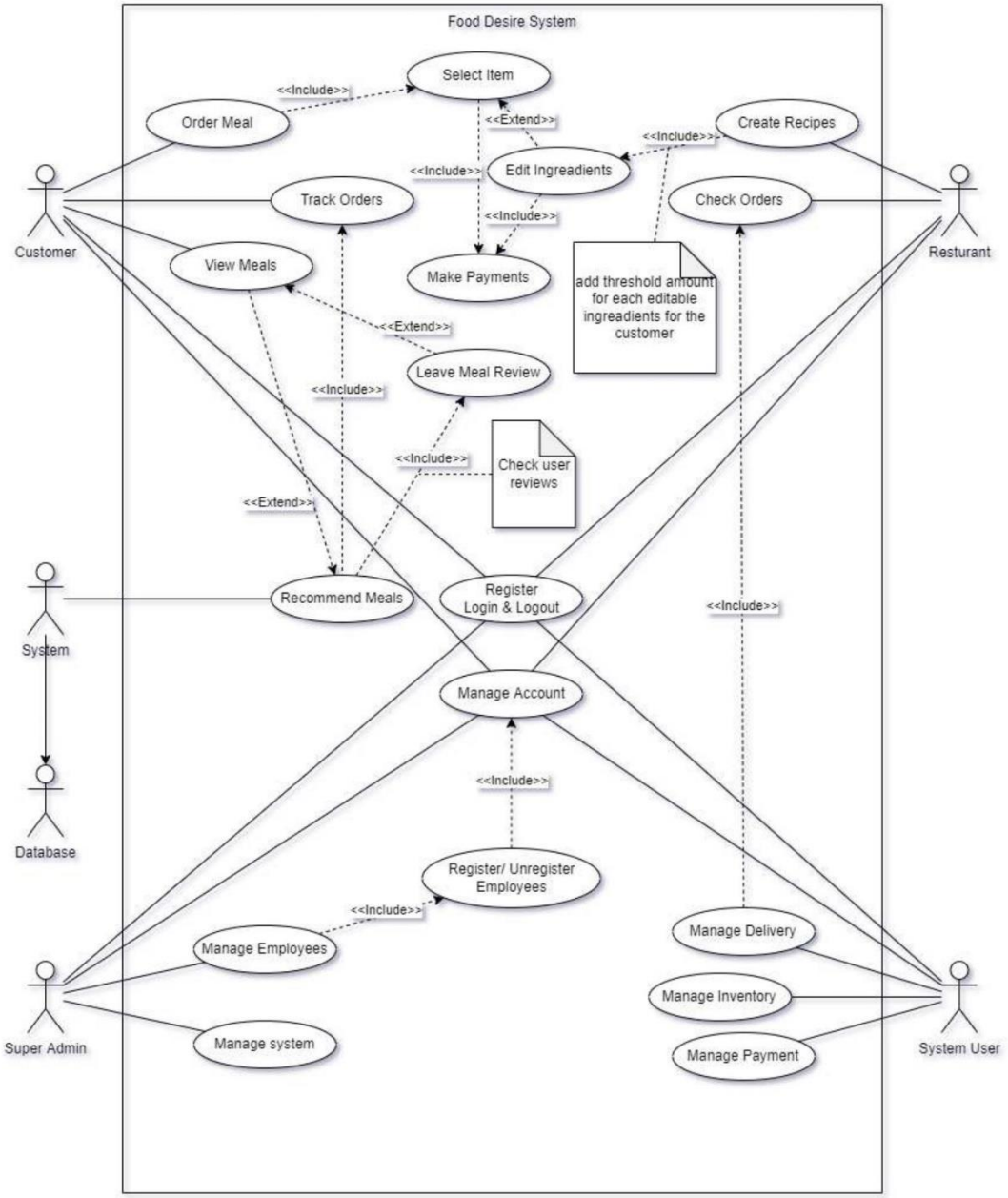


Figure 5. 1 Use case diagram

In the figure 5.1 the customer, and the system actors interact on the web application. All the other actors take apart in the inventory management system. So, the system will remain as 2 separated projects. The restaurant can produce

food items by creating the recipes as the figure 5.1 and those food items will be rendered in the web application for the user. The reason for separating restaurant from interaction on creating or editing the food item because the inventory system will help to manage the ingredients real time as the food items are created.
As in the figure 5.1, the system can manage employees too. So, a small potion of restaurant management interactions will also be included in the inventory system. It is not a complete restaurant management system because it will only manage the actors that directly interact with the inventory like Chefs who create recipe items, Delivery persons and supplier persons. Since the main goal of this system is managing the inventory, the project will remain as an inventory management system.

5.2.1 DATABASE

The database system will be used to implement the backend of the distributed system. A relational database Microsoft SQL server will be used to make the database system. To configure the database system a Framework called Entity Framework will be used. The Entity Framework is a Microsoft .NET Framework data-access platform that models a relational database as a set of conceptual objects(Garofalo, 2011). Entity Framework can help reduce the impedance mismatch for applications and data services.

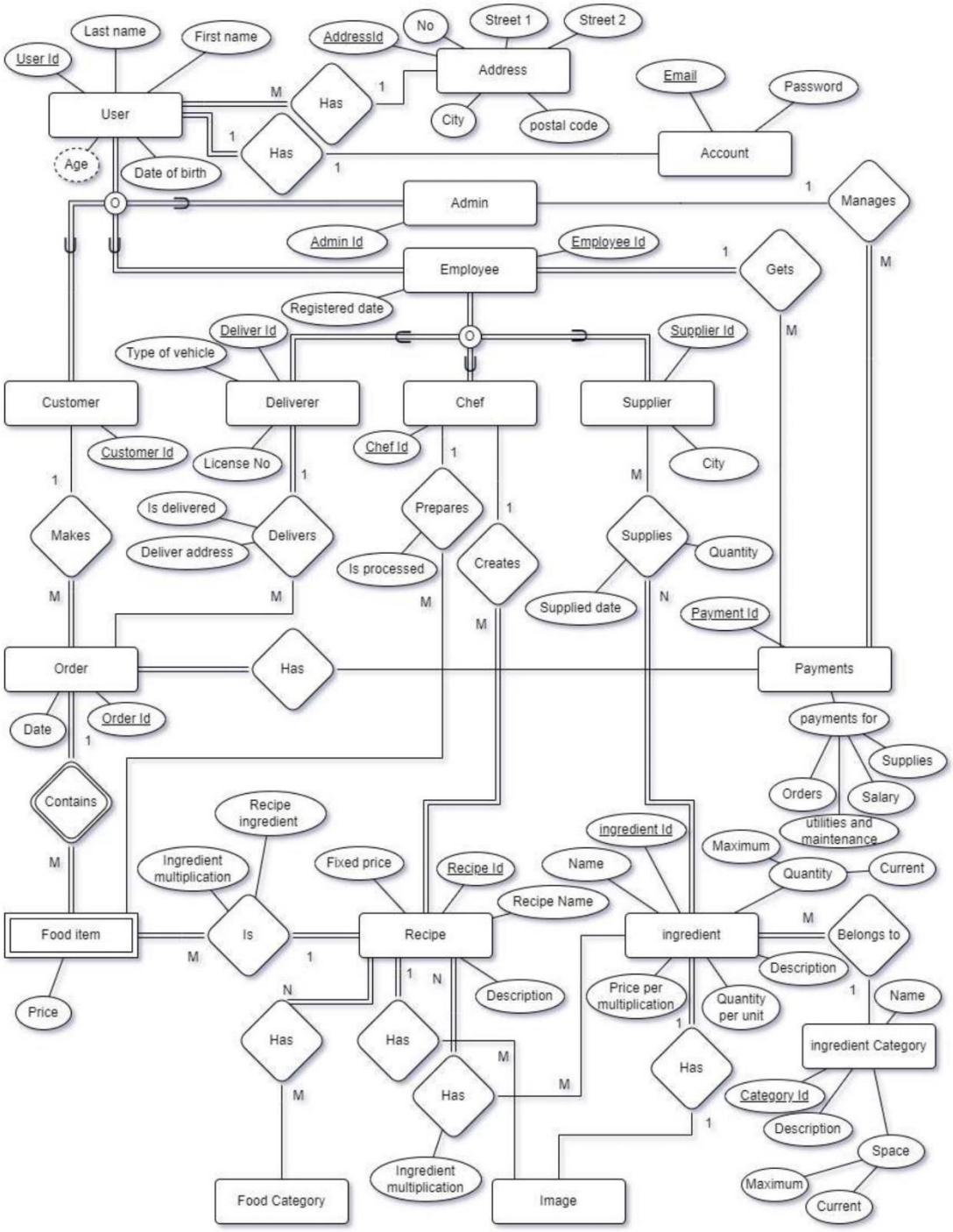


Figure 5. 2 ER Diagram

Figure 5.2 ER diagram assumptions.

1. Every customer, employee is a user.
2. A user can have one account.
3. Multiple users can have the same address.

4. A user can have one default address, another address can be used when ordering food items.
5. Every deliver, chef, supplier is an employee.
6. Every employee gets paid.
7. Payments are managed by the system admin who may be the owner of the restaurant.
8. Food item is dependent of both customer order and the recipe.
9. Food item has ingredient multiplication so the original recipe can be customized.

According to the ER diagram in figure 5.2, a shared class library will be deployed to be share on both Inventory system and the web application. The shared class library will contain all the entities in figure 5.2 as models. These models will be used to generate the database using Entity Framework migrations.

5.2.2 DATA SERVICES

After generating the database, a services layer will be created to interact with the database. Entity Framework allows to use C# OOP methods to handle database operations.

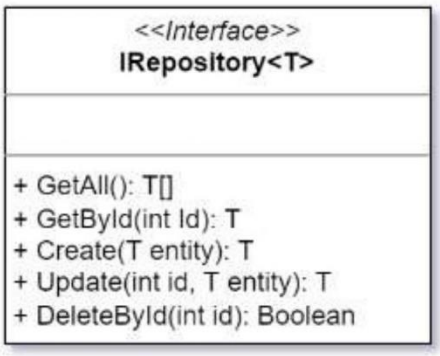


Figure 5. 4 Data Repository

Figure 5. 3 Repository Usage

Repository pattern will be used to handle database operation. The interface in the figure 5.4 will uses a generic type of object, so any entities in the figure 5.2 can implement this interface to do database crud operation. As an example, IRepository<Customer> can be used to do crud operations on Customer entity. For the complex crud operations that includes relation, a service will be deployed for each relation that includes the IRepository as an instance. As an example, figure 5.3.

5.2.3 INVENTORY MANAGEMENT SYSTEM

After designing the data access layer and the services layer, those layers can be used in other project, like in the inventory management system. The inventory system will be developed including all the use cases of actors Admin, system user, restaurant in the figure 5.1. The main purpose of the inventory system is to manage food items for the web application and allowing customers to customize the food items.

For the Ui development of the project WinUi3 will be used. WinUi3 is a native Ui component in the Windows App SDK. It targets Windows 10 or higher. The fluent design of it matches the modern Windows 11 platform.

These are the major interaction in inventory management system.

1. Employees (as the figure 5.2) are registered in this system.
2. Supply details management.
3. Payment management.
4. Food item recipe creation by chefs.

Mainly, there will be the following views in the inventory system.

1. Home view (Where the business summery shows).
2. Recipe view (Where chefs can create food items).
3. Supply view (Where system users manage supplies and order supplies)
4. Employee view (Where the system user manages employees).

The above views will be limited for each role in the system as some of views will be unnecessary show them, as Chefs do not want to see the employee views.

The Recipe view takes a major part in the system as that is the view where the restaurant enables web users to customize their food items. From that view, chefs can create a recipe by using ingredients that are contained in the inventory. Chef can also restrict the amount of adjustable ingredients to stop customers for overdosing the ingredients.

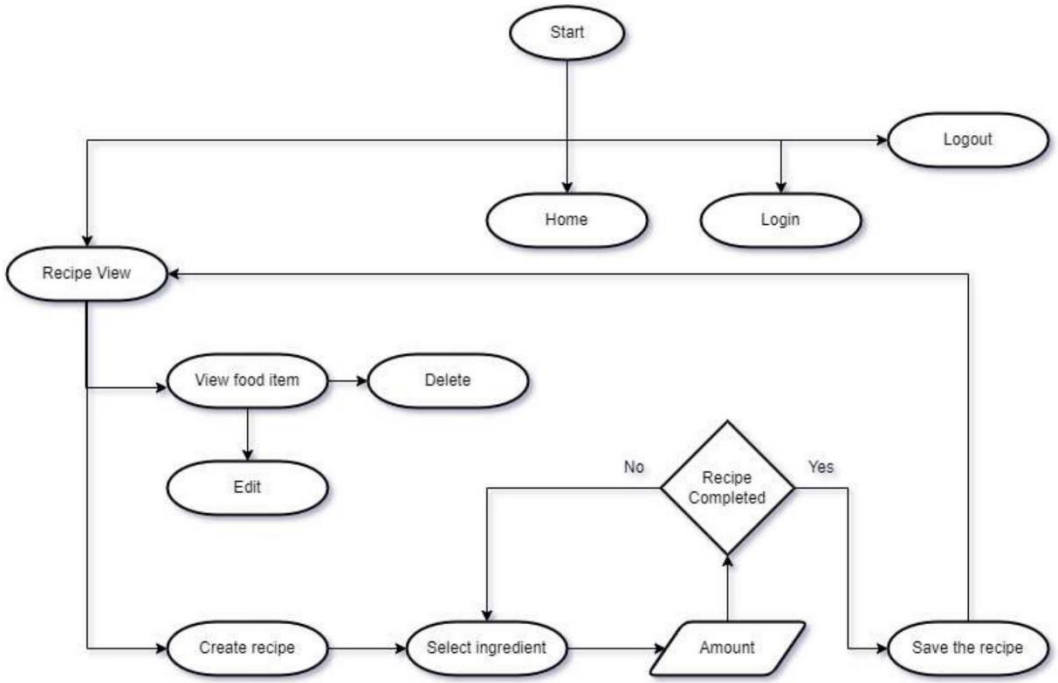


Figure 5. 5 Chef Activity Diagrams

5.2.4 WEB APPLICATION

The goal of developing the web application is to allow the customers of the restaurant to order food via internet. The typical usage of online food ordering system will be followed in this project. Addition to that, the customer can modify the existing recipe for their likings.

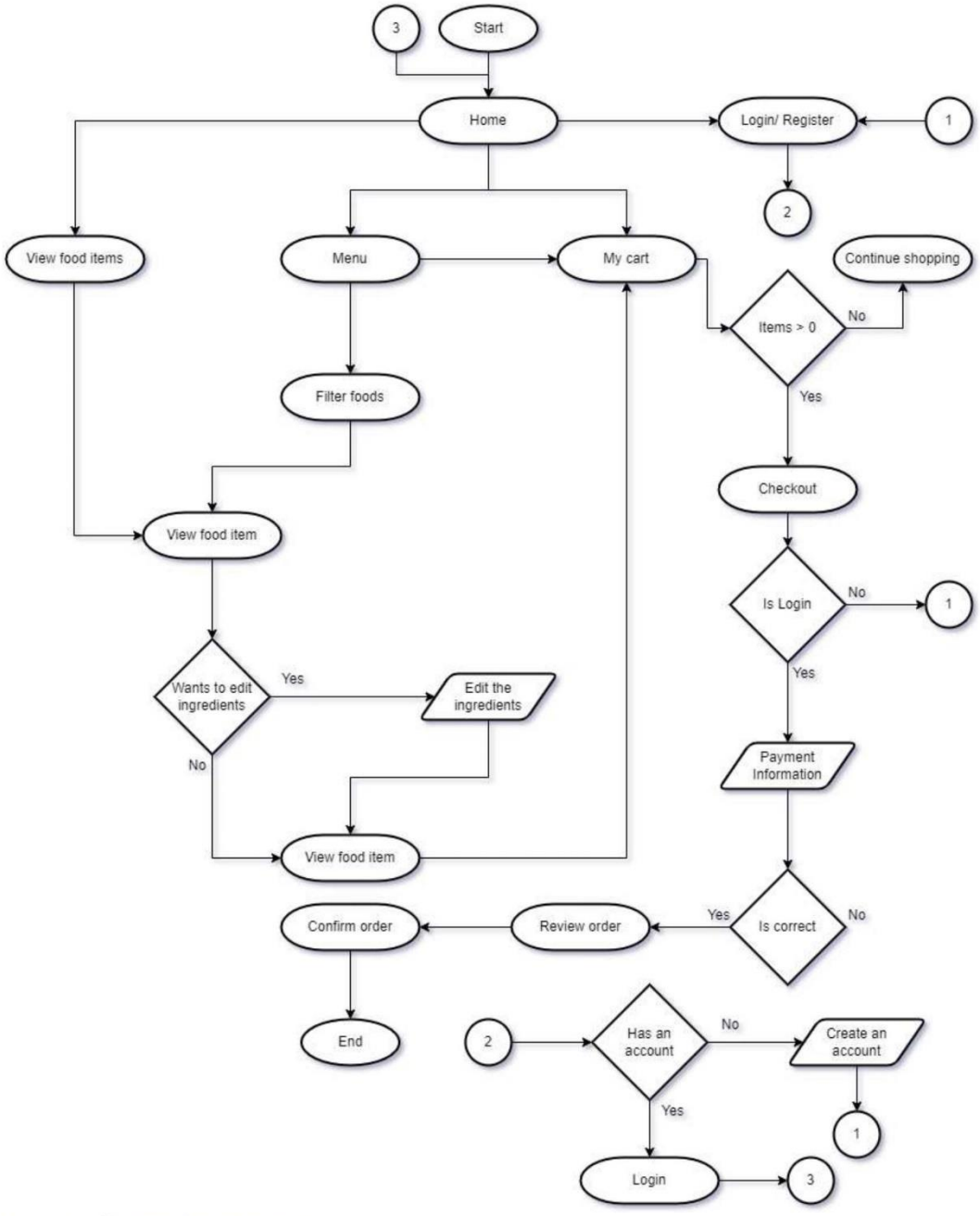


Figure 5. 6 Customer Activity Diagram

The web user will be granted the following views.

1. Home.
2. Sign in/ Sign up.
3. Menu.
4. New food item.
5. Cart.
6. Payment.
7. Food review
8. Account.

New food item view is the main component in this solution. This is the view where the customer customizes their food item by selecting an existing food item from the Food item menu view. After customer customizes their food item customer can order the food from the cart view.

To develop the web application ASP.NET will be used and MVC design pattern will be used to develop the application. The same data and services layers will be used in this project to.

The solution comes with a recommendation system as well. The system will be implemented for the Home view in the web application. To develop the recommendation system ML.NET will be used.

5.3 COMMON DESIGN PATTERNS

The proposed project is a distributed system. So, following good design patterns and practices will help to manages the development and the after services as well.

A common design pattern that is decided to use in the development is MVVM design pattern. Models, Views, View Models. This design pattern is used to loosely couple each component in the project.

As an example, the models that is discussed in chapter 5.2.1 Models is a component in MVVM design patter. That component is loosely coupled so it can be shared across multiple projects without concerning that, that model component will negatively affect the project. These design patterns promote separation of concern throughout the project, so multiple developers can work on the same project different component without cornering about the other component. This design pattern will also separate the view from view logics. Any views that are developed will not affect directly by its view logics.

5.4 TESTING

Testing is a mandatory stage of any project. At this stage a lot of faults can be identified. This helps to deliver a perfect product to the customer.

In the proposed system, the project follows separation of concern practices. In that case, developing testing for the project will be a lot easier be case an each of every component can be tested individually.

Unit testing approach will be the best testing methodology of this type of solution. Unit testing will be implemented as soon as the database is created.

5.5 APPLICATION ARCHITECTURE

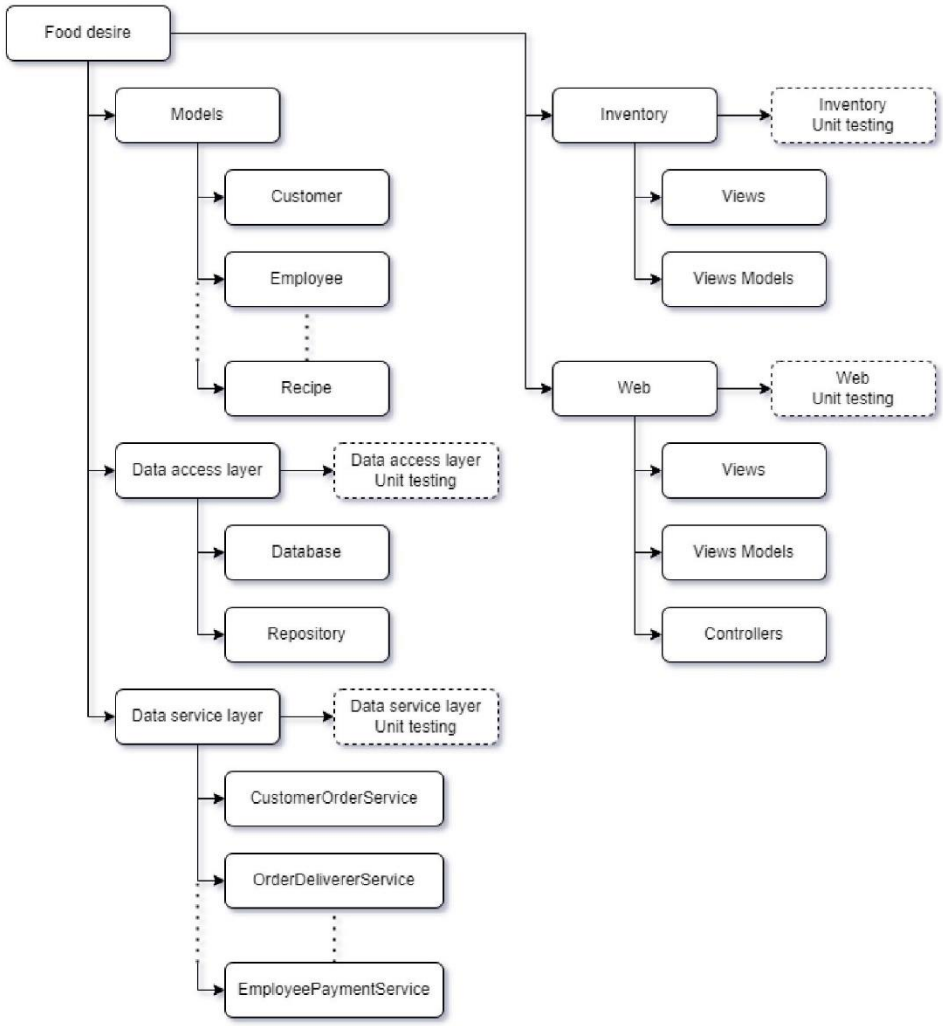


Figure 5. 7 Sketch of Application Architecture

Chapter Six
INITIAL PROJECT PLAN

In the development of there are 4 major phases in the proposed project.

1. Backend development.
2. Desktop application development.
3. Web application development.
4. Recommendation system development.
5. Unit testing.

Task	Group	Start Time	Due Time
Questioner design	Analysis	11/1/2022	11/2/2022
Publish the questionnaire	Analysis	11/2/2022	11/9/2022
Analysis	Analysis	11/10/2022	11/11/2022
User cases design	Planning	11/12/2022	11/12/2022
ERD design	Planning	11/13/2022	11/14/2022
Desktop app wireframe	Planning	11/15/2022	11/19/2022
Web app wireframe	Planning	11/15/2022	11/19/2022
PI Document	Planning	11/20/2022	11/24/2022
Models layer	Backend	11/25/2022	11/25/2022
Data access layer	Backend	11/26/2022	11/26/2022
Services layer	Backend	11/27/2022	11/27/2022
Entity framework	Backend	11/25/2022	11/28/2022
Unit testing Entity framework	Backend	11/28/2022	11/28/2022
Front end	Desktop	11/29/2022	12/13/2022
Add services	Desktop	12/5/2022	12/10/2022
Inventory Analysis view	Desktop	12/5/2022	12/9/2022
Basic views	Desktop	11/30/2022	12/4/2022
Front end	Web	12/14/2022	12/28/2022
Unit testing	Desktop	12/11/2022	12/13/2022
Basic views	Web	12/15/2022	12/19/2022
Add services	Web	12/20/2022	12/28/2022
Design	Recommendation syste	12/29/2022	1/27/2023
Unit test	Recommendation syste	1/23/2023	1/27/2023
Merge with web	Recommendation syste	1/28/2023	2/1/2023 1
Deploy	Launch	2/2/2023 1	2/3/2023 1
Server test runs	Launch	2/4/2023 1	2/7/2023 1

Figure 6. 1 Initial Time Frame

6.1 BACKEND DEVELOPMNET

In this development phase the core structure of application, the database and the data services will be developed for both desktop application and the web application. The backend for both desktop app and the web app use the same backend project where all the services are implemented.

6.2 DESKTOP APPLICATION

The desktop application is the Inventory management system in this project. It will be developed as discussed in the Chapter Five. The development of the desktop application can be started after the development of the Backend. Developing the backend first, allows the development of the desktop application development more efficient because there are only view logics to be implemented in the desktop application.

6.3 WEB APPLICATION

The development of the web application can be also started after developing the Backend in chapter 6.1. But in this scenario, it is best to wait for the development of the Inventory system, the desktop application is finished because the core component of the web application in the Chapter 2.2 is depending on the Inventory management system. The same benefits can be achieved as developing the backend first in the desktop application in Chapter 6.2.

Both development of desktop application in the Chapter 6.1 and the web application in Chapter 6.3 are focused on the development of the frontend. As for the backend of each development there are only view logics to be implemented because the main backend for both projects is developed as discussed in chapter 6.1.

6.4 RECOMMENDATION SYSTEM

The recommendation system is a key component in the web application. But it minorly effects the development of the web application. The recommendation system can be developed as a completely different project and implement that after the development of the web application.

6.5 UNIT TESTING

Since the projects of the system are completely separated from another, Unit testing for each project, the backend, desktop application, web application and recommendation system, can be implemented at anytime after the development of each project is completed. In this scenario the unit testing will be implemented after the development of each project is completed because some of the projects shares its functionalities across the multiple projects.

Chapter Seven
RISK ANALYSIS

7.1 CHOOSING PLATFORM

To develop the proposed system, the decided development platform is .NET. As for this project, it is proposed to use the up to dated SDK and frameworks. Since these SDKs and frameworks are latest version there may be some lack of community support. But the official documentation provides enough support to develop this application.

7.2 COMPATIBILITY

After launch of the system client may face incompatibility issue. Most of the restaurants are unlikely to use an up to dated system for their general-purpose activities. Since this project uses the latest developing tools, some of the SDKs require latest Windows platform to support. As an example, WinUI3 require Windows 10 and later.

This risk can be managed if all the hardware components are provided by the developers. Otherwise, the client must be aware of the system requirements.

7.3 RECOMMENDATION SYSTEM

One of major risk in the proposed project is the development of the recommendation system. It is an AI that must be trained before the implementation. To train the AI it is required a huge amount of historical data. The risk the system having is the lack of data to be trained.

There are some datasets related to the food industries. But most of the data might be useless as not all the food categories will be used in the project. Filtering out the data might significantly reduce the accuracy of the AI model. But it can be used for the initial launch of the project. After that the AI model can be trained again by using the data that gathered from this system.

Bibliography

Ahrens, S. (2012) 'Recommender Systems', in.

Daim, T.U. *et al.* (2013) 'Exploring technology acceptance for online food services', *Int. J. Bus. Inf. Syst.*, 12, pp. 383–403.

Dana, L.M. *et al.* (2021) 'Factors associated with ordering food via online meal ordering services', *Public Health Nutrition*, 24(17), pp. 5704–5709. Available at: https://doi.org/10.1017/S1368980021001294.

Fujita, T., Shimada, H. and Sato, K. (2014) 'Self-ordering system of restaurants for considering allergy information', *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pp. 179–184.

Gao, F. and Su, X. (2018) 'Omnichannel Service Operations with Online and Offline Self-Order Technologies', *Manag. Sci.*, 64, pp. 3595–3608.

Garofalo, R. (2011) 'The Entity Framework', in.

Goffe, L. *et al.* (2021) 'Appetite for Disruption: Designing Human-Centred Augmentations to an Online Food Ordering Platform', in *34th British Human Computer Interaction Conference Interaction Conference, BCS HCI 2021*. BCS Learning and Development Ltd., pp. 155–167. Available at: https://doi.org/10.14236/ewic/HCI2021.16.

Mishra, P. (2019) 'Supervised Learning Using PyTorch', *PyTorch Recipes* [Preprint].

R., A. *et al.* (2017) 'Online Food Ordering System', *International Journal of Computer Applications*, 180(6), pp. 22–24. Available at: https://doi.org/10.5120/ijca2017916046.

Ratto, F. *et al.* (2008) 'A numerical approach to quantify self-ordering among self-organized nanostructures', *Surface Science*, 602, pp. 249–258.

Sarwar, B.M. *et al.* (2001) 'Item-based collaborative filtering recommendation algorithms', in *The Web Conference*.

Singh, A., Thakur, N. and Sharma, A. (2016) 'A review of supervised machine learning algorithms', *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315.

Thomas, A. and Davis, S.E. (1978) 'Laboratory ordering: a system which eliminates paperwork.', *The American journal of medical technology*, 44 10, pp. 1026–7.

Zulkarnain, K. *et al.* (2015) 'Key success factors of online food ordering services:an empirical study', in.

Interim report

PUSL3119 Computing Individual Project

Project Interim Report

Food Desire

An Online Food Ordering System

Supervisor: Mr. Pramudya Thilakarathne

Name: Rathnayake M Rathnayake

Plymouth Index Number: 10747887

Degree Program: BSc (Hons) Computer Science

Table of Contents

TABLE OF FIGURE

Chapter One
INTRODUCTION

1.1 INTRODUCTION

Online shopping has become a convenient and popular way of purchasing goods in the modern world. It offers many benefits such as speed, security, variety and comfort. One of the main advantages of online shopping is that it saves time for busy consumers who do not want to waste time going to physical stores. Online shopping allows them to order anything they need from their doorsteps and receive it quickly and safely.

Online food ordering is the latest trend of service that can be offered by fast-food restaurants. It allows customers to order food online and have it delivered to their location. It also enables electronic payment systems that make transactions convenient and secure. Online food ordering can benefit both customers and restaurants by saving time, reducing costs and increasing customer satisfaction.

These systems allow fast-food restaurants to focus more on food preparation and faster delivery, while reaching a wider customer base through the internet. Online food ordering systems are a valuable business opportunity for the food industry, as they leverage the high demand and convenience of internet usage.

1.1 PROBLEM DEFINITION

The food industry is expanding rapidly and constantly innovating to meet the diverse needs and preferences of consumers. One of the emerging trends is the customization of food products, which allows customers to tailor their meals according to their tastes and dietary requirements. However, this option is still limited in scope and availability, especially in online food ordering systems. For example, most pizza restaurants only offer a fixed menu of toppings and crusts, without giving customers the freedom to adjust the quantity or quality of ingredients. This restricts customers from creating their ideal pizza and prevents them from knowing the nutritional value of their order. Therefore, online food ordering systems should enhance their features to enable more flexibility and transparency in food customization.

1.3 PROJECT OBJECTIVES

The online food ordering system is a convenient and efficient way for customers to order their meals from various restaurants. However, some customers may have specific preferences or dietary restrictions that limit their choices of meals. To address this issue, the proposed solution is to add a feature that allows customers to customize their meals by adjusting the amount of ingredients in each meal item. This feature will enhance the customer experience by giving them more control and flexibility over their orders. To implement this feature, we will also need to modify the inventory management system of the restaurant end, so that they can keep track of the available ingredients and update the list of ingredients for each meal item accordingly. This will also improve the restaurant end experience by reducing waste and optimizing inventory.

This solution will update both customer and restaurant end to enable this feature. Customers will be able to select their preferred meal item, before going to the checkout or cart, they will be presented with a page that includes the meal information. The restaurant can provide customers with which ingredients are used in each dish so that customers have full transparency when ordering. Additionally, customers can edit the ingredients as they prefer - for example if a customer would like more chicken pieces in their fried rice then they can increase it from the ingredient list accordingly. This allows for greater control over what goes into your meals while also providing restaurants with an easy way of informing consumers about what is included within each dish on offer.

Moreover, a recommendation system will be implemented to suggest meal items to the users based on their preferences and previous orders. This system will use a content-based filtering machine learning algorithm that will analyze the data collected from each user and match it with the attributes of the meal items. The recommendation system will improve the user experience by providing personalized and relevant suggestions instead of showing the default meals list on the home page. Moreover, the recommendation system will also benefit the restaurant by increasing customer satisfaction and loyalty, as well as boosting sales and revenue.

Chapter 02
SYSTEM ANALYSIS

2.1 FACTS GATHERING TECHNIQUES

2.1.1 RESEARCH

2.1.1.1 Online Food Ordering Systems

Self-ordering systems are systems that allow customers to place orders by themselves without the need for human intervention. These systems have been shown to have various benefits for both customers and businesses. According to (Ratto *et al.*, 2008), self-ordering systems can achieve a state of maximal overall 'success' by optimizing the trade-off between customer satisfaction and operational efficiency. (Gao and Su, 2018) also found that self-ordering techniques can significantly reduce waiting cost and increase demand, as customers prefer faster and more convenient ways of ordering their needs. Self-ordering technology can be implemented through various channels, such as text messages, emails, telephone calls and internet websites or applications.

This system allows customers to place their orders online or through a digital device without the need for human intervention. According to (Thomas and Davis, 1978), this system can replace the traditional ordering technique and save time and resources. Moreover, self-ordering systems can enhance customer satisfaction by offering convenience, speed and delivery options. From a business perspective, self-ordering systems can increase sales volume and revenue by reaching more customers and reducing operational costs. Therefore, self-ordering systems are a beneficial tool for both businesses and customers in today's competitive market.

One of the most modern methods to implement a self-ordering system for a restaurant is online food ordering. This method allows customers to order food from their devices without having to wait for a waiter or a menu. Online food ordering has many benefits for both customers and restaurants. According to (R. *et al.*, 2017), online food ordering systems are efficient and easy to use, as customers can browse through the food items that the restaurant offers and order

the food with few steps. (Daim *et al.*, 2013) also found that customers value the efficiency of the website, speed of response duration on online services, and the quality of the personnel when choosing an online food service. Therefore, online food ordering can enhance customer satisfaction and loyalty, as well as increase sales and revenue for restaurants.

Despite the popularity and convenience of online food ordering services, there have been few significant innovations in this domain. The current system can be improved in various aspects to enhance customer satisfaction and loyalty. According to (Zulkarnain *et al.*, 2015), website quality and service quality are crucial factors that influence the success of online food ordering services. They suggest that online platforms should provide clear and accurate information, easy navigation, fast delivery, and responsive feedback. Moreover, (Goffe *et al.*, 2021) propose that online food ordering services should adopt a more human-centered design approach. They argue that customers should have more autonomy and flexibility in choosing their food options, such as customizing their ingredients, portions, or preferences.

2.1.1.2 Recommendation System

Artificial intelligence (AI) and machine learning (ML) are powerful technologies that can enhance various aspects of e-commerce. E-commerce is a business model that involves buying and selling goods and services online or through other computerized systems. AI and ML can help e-commerce businesses optimize their operations, improve customer experience, increase sales and revenue, and gain competitive advantage. Some examples of AI and ML applications in e-commerce are product recommendation systems, chatbots, fraud detection, inventory management, pricing optimization, and sentiment analysis.

Recommendation systems are a type of artificial intelligence model that are widely used in e-commerce domains. They aim to provide personalized suggestions to users based on their preferences and past interactions with products or services. According to (Ahrens, 2012), recommendation systems can enhance customer satisfaction and loyalty by increasing the time spent on the site, the number of products viewed, and the likelihood of purchase or engagement.

2.2 EXISTING SYSTEM

The online food ordering system is a convenient and efficient way for customers to order food from various restaurants. The system allows customers to browse through a food menu online and select the items they want to order. The system also provides information about the availability, price, and delivery time of the food items. Customers can track their orders online and see when they will receive their food(R. *et al.*, 2017). The system also has a feedback mechanism that enables customers to rate and review the food items and the restaurants. This helps other customers to make informed choices and helps the restaurants to improve their service and quality. The system also has a recommendation feature that suggests food items based on the customer's preferences and previous orders. The system aims to increase the transparency and trustworthiness of the restaurants by providing accurate and reliable information. Customers can pay for their orders online using various payment methods or opt for cash on delivery if they prefer. The system also ensures the security and privacy of each customer by creating separate accounts with unique ID and password(R. *et al.*, 2017).

Online food ordering systems are platforms that allow customers to order food from various restaurants and pay securely online. These systems can include both mobile and web applications that improve the user experience by offering features such as menus, ratings, reviews, delivery tracking and loyalty programs. Online food ordering systems benefit both customers and restaurants by providing convenience, efficiency and transparency.

2.3 DRAWBACKS OF THE EXISTING SYSTEM

One of the drawbacks of current online food ordering systems is that they do not allow customers to customize their food ingredients according to their preferences, dietary needs, or allergies. This can limit customer satisfaction and loyalty, as well as reduce sales opportunities for restaurants that offer such options.

Another drawback is that most online food ordering systems do not have a separate inventory management system for restaurants, which means that restaurants must manually track their stock levels, order supplies, and avoid wastage. This can increase operational costs, errors, and inefficiencies for

restaurants that rely on online orders. A food ordering system that can customize food ingredients and have a separate inventory management system for restaurants would solve these problems by providing more flexibility, convenience, and efficiency for both customers and restaurants.

Chapter 03
REQUIREMENTS SPECIFICATION

3.1 FUNCTIONAL REQUIREMENTS

- Online Food Ordering System
  - Customers can browse through different food categories and items.
  - Customers can customize their orders by changing the amount of ingredients.
  - Customers can see the price of their orders based on their customization.
  - Customers can place their orders online and pay with various methods.
  - Customers can track their orders status and delivery time.
  - Customers can rate and review their orders after delivery.
  - Customers can view their order history and recommendations.

- Inventory Management System
  - Admins can register new employees, suppliers, chefs, and deliverers.
  - Admins can assign roles and permissions to different employees.
  - Suppliers can supply ingredients to the restaurant based on demand.
  - Chefs can create recipes for different food items using ingredients.
  - Chefs can update recipes based on customer feedback or availability of ingredients.
  - Deliverers can take out orders for delivery based on location and priority.
  - Admins can pay for maintenance, salaries, taxes, etc.

- Recommendation System
  - The system collects data from customer orders, ratings, reviews, preferences, etc.
  - The system analyzes data using machine learning algorithms to find patterns and trends.
  - The system suggests food items for customers based on their most consumed food and individual ingredients.

- The system also considers factors such as seasonality, availability, popularity, nutrition, etc.
- The system displays suggestions on the website's home page.

3.2 NON-FUNCTIONAL REQUIREMENTS

- Online Food Ordering System
  - The system has a user-friendly interface with clear buttons, colors, fonts, etc.
  - The system supports English.
  - The system has a high performance.
  - It loads fast.
  - It responds quickly.
  - It handles concurrent requests.
  - The system has a high security.
  - It encrypts sensitive data such as payment information.

- Inventory Management System
  - The system has a reliable database that stores all information related to inventory, employees, suppliers, etc.
  - The system has a backup mechanism that saves data regularly in case of failure or data loss.

- Recommendation System
  - The system has a scalable architecture that allows it to handle increasing numbers of customers, orders, data, etc.
  - The system has an adaptive algorithm that updates itself based on new data or feedback.

3.3 USE CASE DIAGRAM

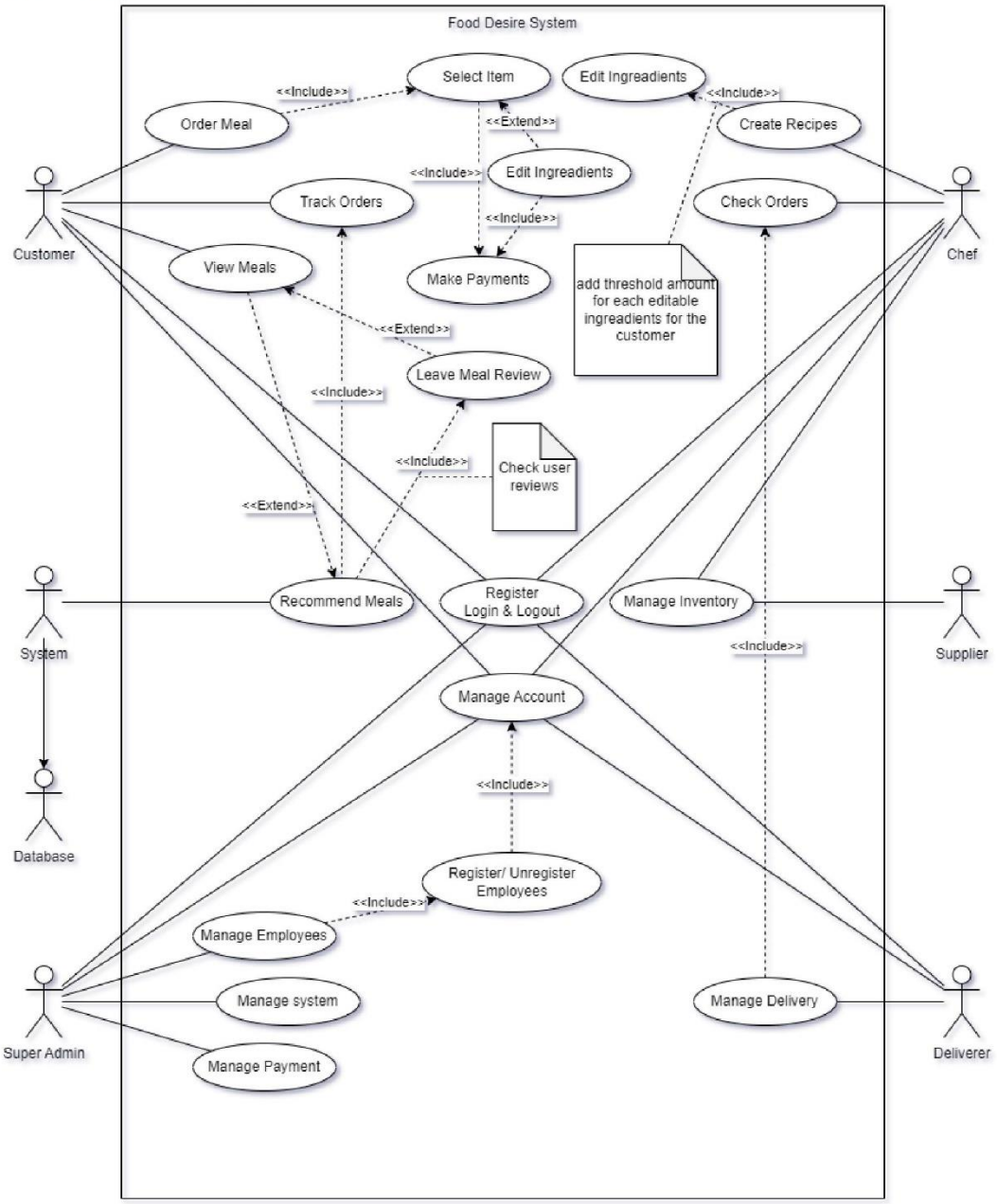


Figure 3. 1 Use case diagram.

3.4 HARDWARE / SOFTWARE REQUIREMENTS

3.4.1 HARDWARE REQUIREMENTS

The online food ordering system is structured like a standard web application, and as such requires no special hardware and only basic software, namely web and database servers, to function properly. The system can run on any device that supports a web browser, such as desktops, laptops, tablets, or smartphones.

The inventory management system is a software application that helps businesses track and manage their inventory levels, orders, sales, and deliveries. The system can also generate reports and insights to optimize inventory performance and reduce costs.

One of the requirements for the inventory management system is the hardware configuration. Depending on the size and complexity of the business, it may need few PCs or a single PC that can be shared with multiple users at the same time with separated display. This option can save space and money by using one PC as a server and connecting multiple monitors, keyboards, and mice to it. Each user can have their own session and access to the inventory management system.

However, this option also has some drawbacks. It may affect the performance and reliability of the system if there are too many users or tasks running on one PC. It may also pose security risks if the PC is not properly protected from unauthorized access or malware. Moreover, it may limit the scalability and flexibility of the system if there is a need to add more users or features in the future.

Therefore, another option is to use a single PC for each user. This option can ensure better performance and security of the system as each user has their own dedicated hardware and software resources. It can also allow more customization and integration of the system with other applications or devices. However, this option also has some challenges. It may require more space and money to purchase and maintain multiple PCs. It may also require more coordination and synchronization among different users to avoid data inconsistency or duplication.

3.4.2 SOFTWARE REQUIREMENTS

To use either option in Chapter 3.4.1, the PC must meet some minimum specifications to run .NET 7.

- Operating system: Windows 10 version 1809 or later.
- Processor: 1.8 GHz or faster processor.
- Memory: 4 GB of RAM; 8 GB of RAM recommended.
- Disk space: Minimum of 800 MB up to 5 GB of available space, depending on features installed; typical installations require 1 GB of free space.
- Network: Internet access (fees may apply).

3.5 NETWORK REQUIREMENTS

The system should have a reliable and secure network connection to communicate with the WinUI 3 desktop application, the Web API, and the Blazor web UI. The network bandwidth should be sufficient to handle concurrent requests and data transfers without delays or errors.

The system should use HTTP protocol for encryption and authentication of data between the client and server. The system should also implement SSL certificates for verifying the identity of the server and preventing man-in-the-middle attacks.

The system should use a cloud service provider such as Azure Cloud to host the Web API and the Blazor web UI. The cloud service provider should offer scalability, availability, backup, and disaster recovery features for the system. The cloud service provider should also comply with relevant data protection laws and regulations.

Chapter 04
FEASIBILITY STUDY

4.1 OPERATIONAL FEASIBILITY

To ensure that the system meets the needs of stakeholders, surveys and interviews can be conducted to gather information on their requirements, preferences, and expectations. These can be conducted with potential customers, restaurant owners, chefs, deliverers, and suppliers. This information can be used to design a user-friendly and efficient system that meets their needs.

Additionally, the benefits and costs of the system for each stakeholder group can be evaluated. This can include estimating the time and money the system can save customers by providing them with online ordering and fast delivery, as well as estimating the revenue and profit the system can generate for restaurant owners by reducing food waste and optimizing inventory management.

The risks and challenges of implementing the system in the real world should also be assessed. This can include identifying legal, ethical, and social issues that may arise from collecting and processing personal data, as well as potential threats to the system's security and reliability.

4.2 TECHNICAL FEASIBILITY

Reviewing the technical resources available for developing the system using .NET is necessary. This includes the necessary tools, technology, materials, labor, and logistics to create a WinUI 3 desktop application for inventory management system (IMS), a Web API to consume in a blazorUi (web app), and an Azure cloud deployment.

Additionally, the technical capabilities of the system should be evaluated in terms of functionality, performance, scalability, and compatibility. This can include testing how well the system can handle different types of inputs, outputs, processes, and data formats, as well as testing how quickly it can respond to user requests and how many concurrent users it can support.

Exploring technical innovations that can be incorporated into the system to enhance its value proposition is also necessary. This can include researching content filtering machine learning algorithms to provide personalized recommendations based on customer feedback, as well as artificial intelligence techniques such as natural language processing or computer vision to improve user experience by enabling voice or image recognition features.

Chapter 05
SYSTEM ARCHITECTURE

5.1 ER DIAGRAM

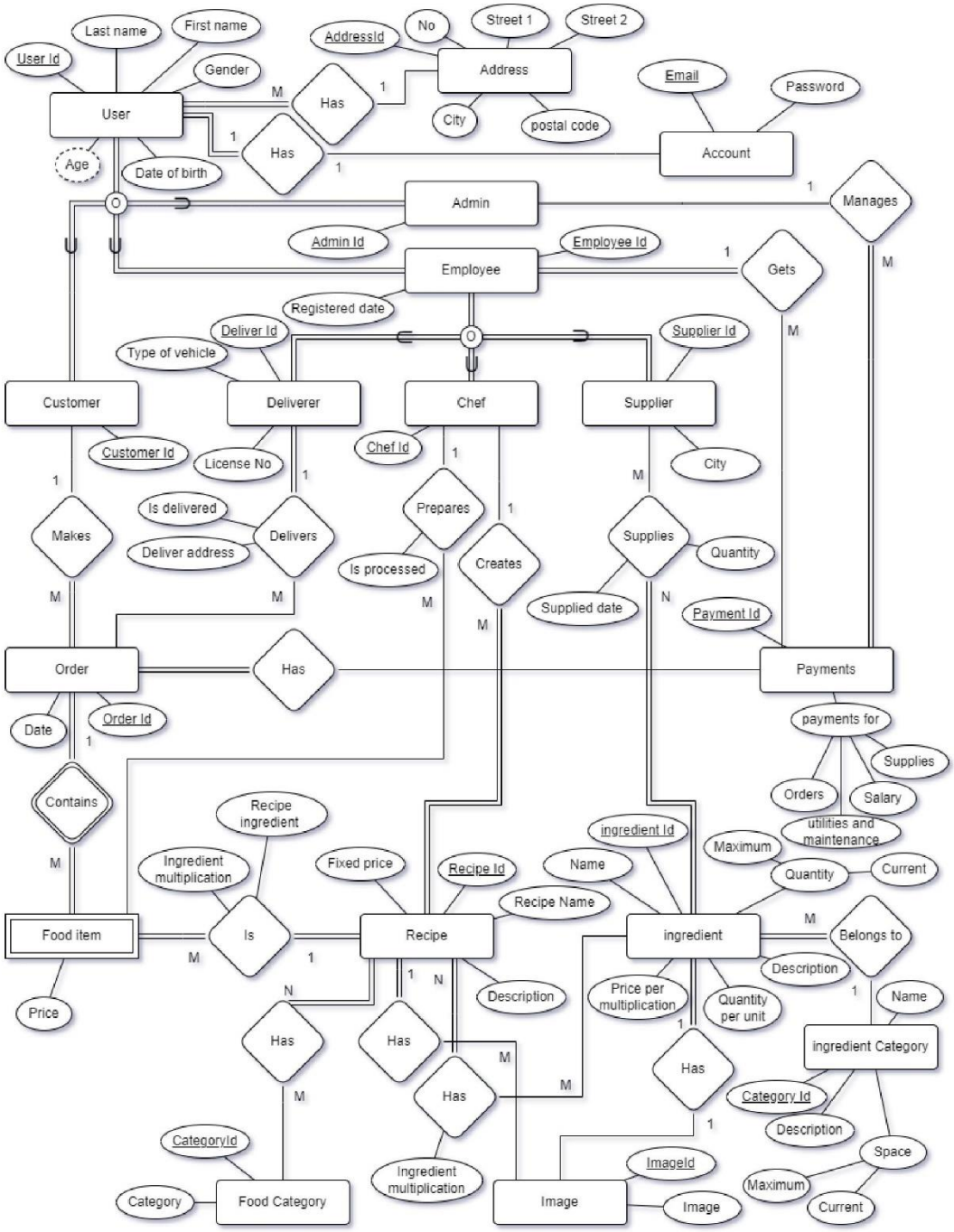


Figure 5. 1 ER Diagram.

5.2 CLASS DIAGRAMS

5.2.1 MODELS

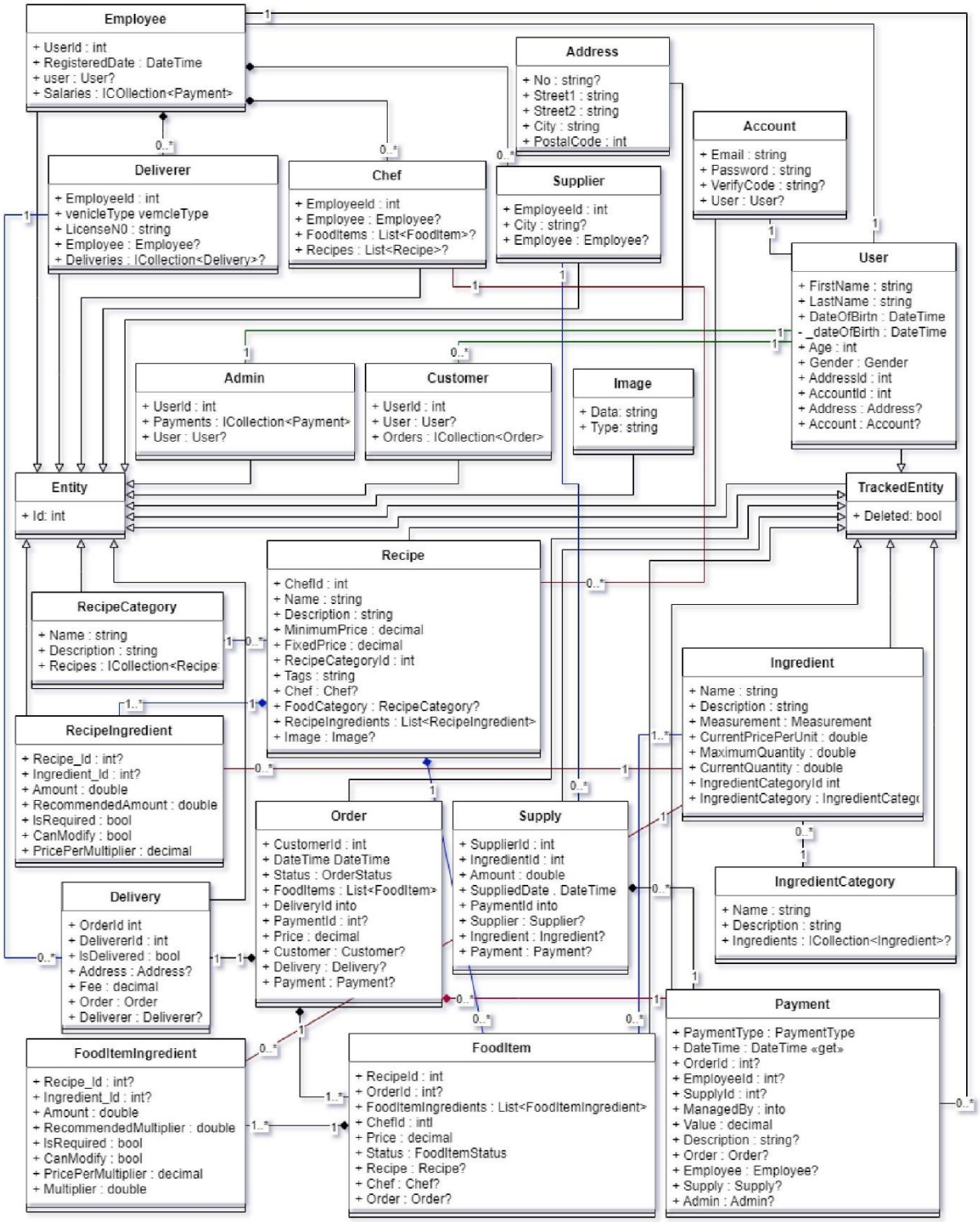


Figure 5. 2 Models Class Diagram.

The class diagram represents a model for managing each entity in figure 5.1 in the system, which is built using Entity Framework Core. This is a class library that can be used across the system.

The Entity class serves as a base class for all entities in the project, and provides a single property, Id, which represents the primary key for each entity. By inheriting from this base class, all entities in the project can access this property, simplifying the implementation of the generic repository pattern which will be further discuss in the document. In the context of the generic repository pattern, exposing the primary key property allows for easy execution of queries that are specific to a given entity. For example, a query to retrieve a single entity by its primary key can be executed using a simple LINQ expression Where “T” is the entity type being queried. T can be any entity in figure 5.1/5.2

```
public T GetById(int id) {
    return _dbContext.Set<T>().Find(id);
}
```

Overall, the Entity class serves as a useful tool for implementing the generic repository pattern and simplifying database operations across all entities in the project.

TrackedEntity is a class that can be used to implement soft delete functionality for any entity class that inherits from it. Soft delete is a pattern used to mark a record as deleted instead of deleting it from the database. This allows the record to be easily restored if needed and makes it possible to track when a record was deleted. TrackedEntity contains a Boolean property named IsDeleted, which is set to true when the entity is deleted. This property can be used to filter out deleted records when querying the database.

```
public T DeleteById(int id) {
    T entiy = _dbContext.Set<T>().Find(id);
    entiy.Deleted = true;
    _dbContext.SaveChanges();
}
```

5.2.1 DATA ACCESS LAYER (DAL)

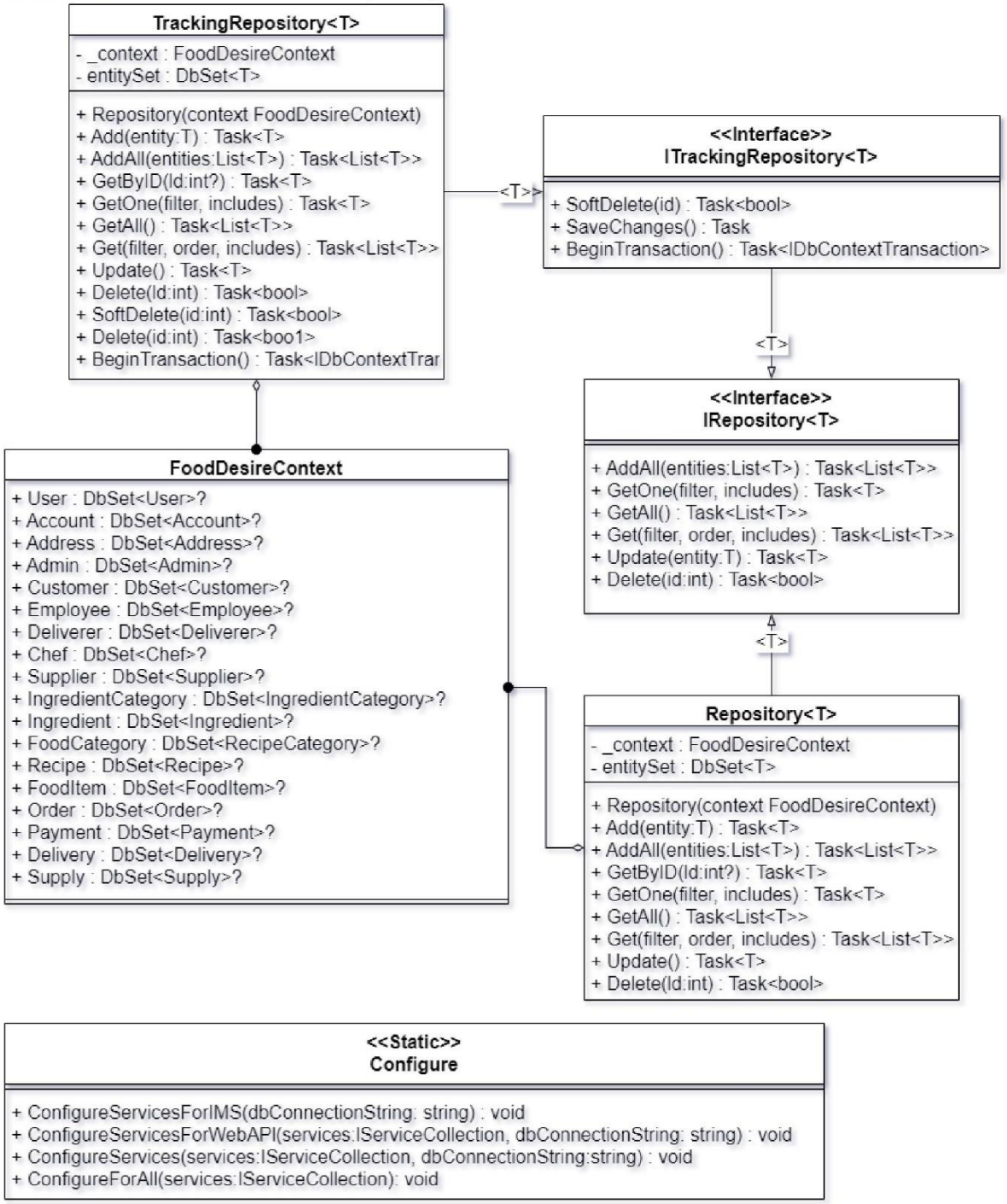


Figure 5. 3 DAL Class Diagram.

The class diagram (Figure 5.3) represents the data access layer (DAL) for a food ordering system. The FoodDesireContext class is for Entity Framework and contains properties for each entity (e.g., User, Account, Address, Admin, etc.) that can be persisted in the database.

Repositories are used as an abstraction layer between the data access logic and the business logic of an application, which is shown by the IRepository<T> interface and it is implemented in the Repository<T> class. The Repository<T> class provides methods for CRUD (Create, Read, Update, Delete) operations on the entities, which are executed on the DbContext instance.

There is also a ITrackingRepository<T> interface and it is implemented in the TrackingRepository<T> class that extends the IRepository<T> interface and provides additional methods for soft delete and transaction management.

Finally, the Configure class is used for dependency injection to configure the services that the application needs. This allows for loose coupling and better testability of the code. It defines the mappings between interfaces and implementations, which allows the application to configure dependencies at runtime.

5.2.2 CORE LAYER

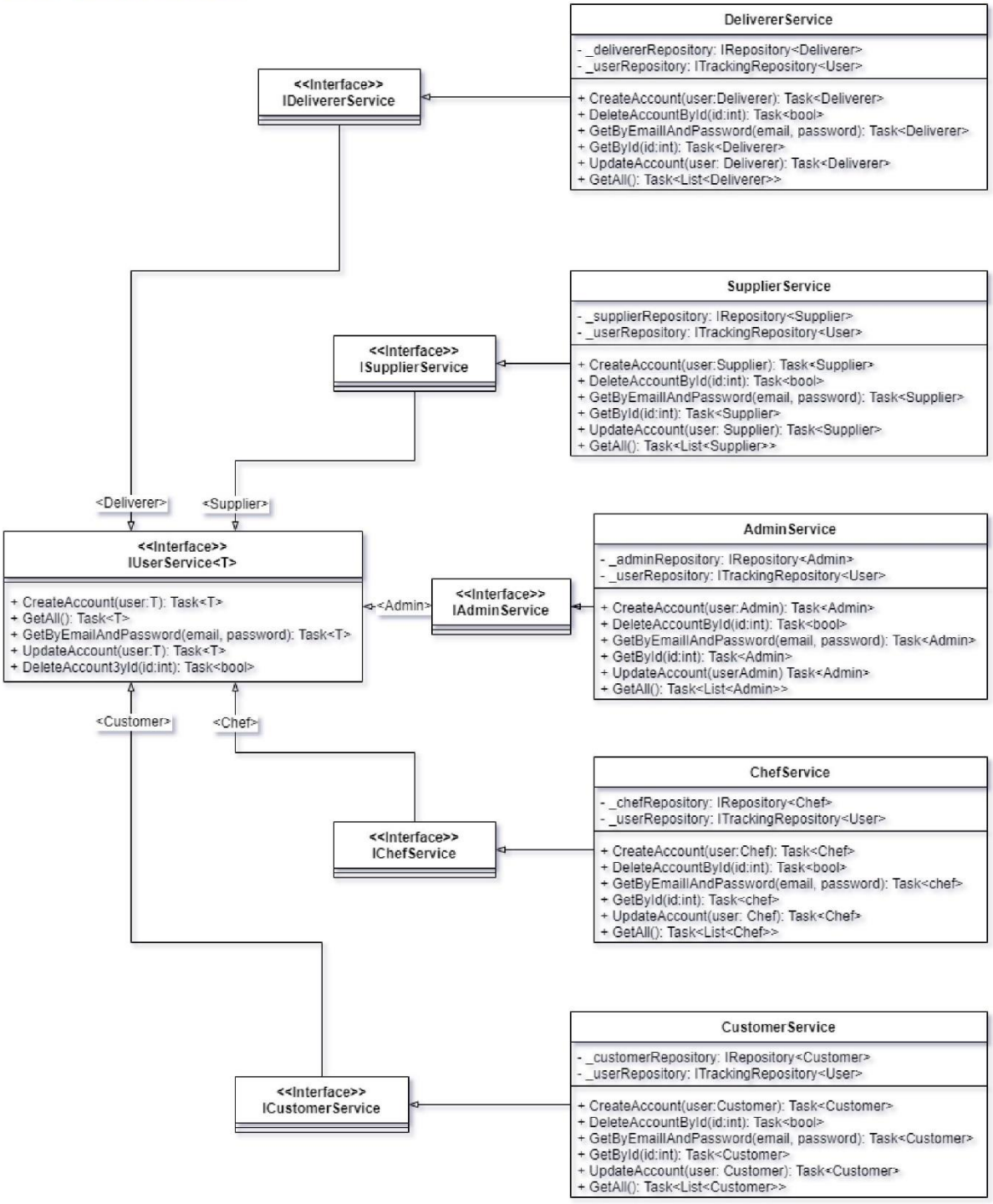


Figure 5. 4 User Services Diagram. (Inheritance only).

20

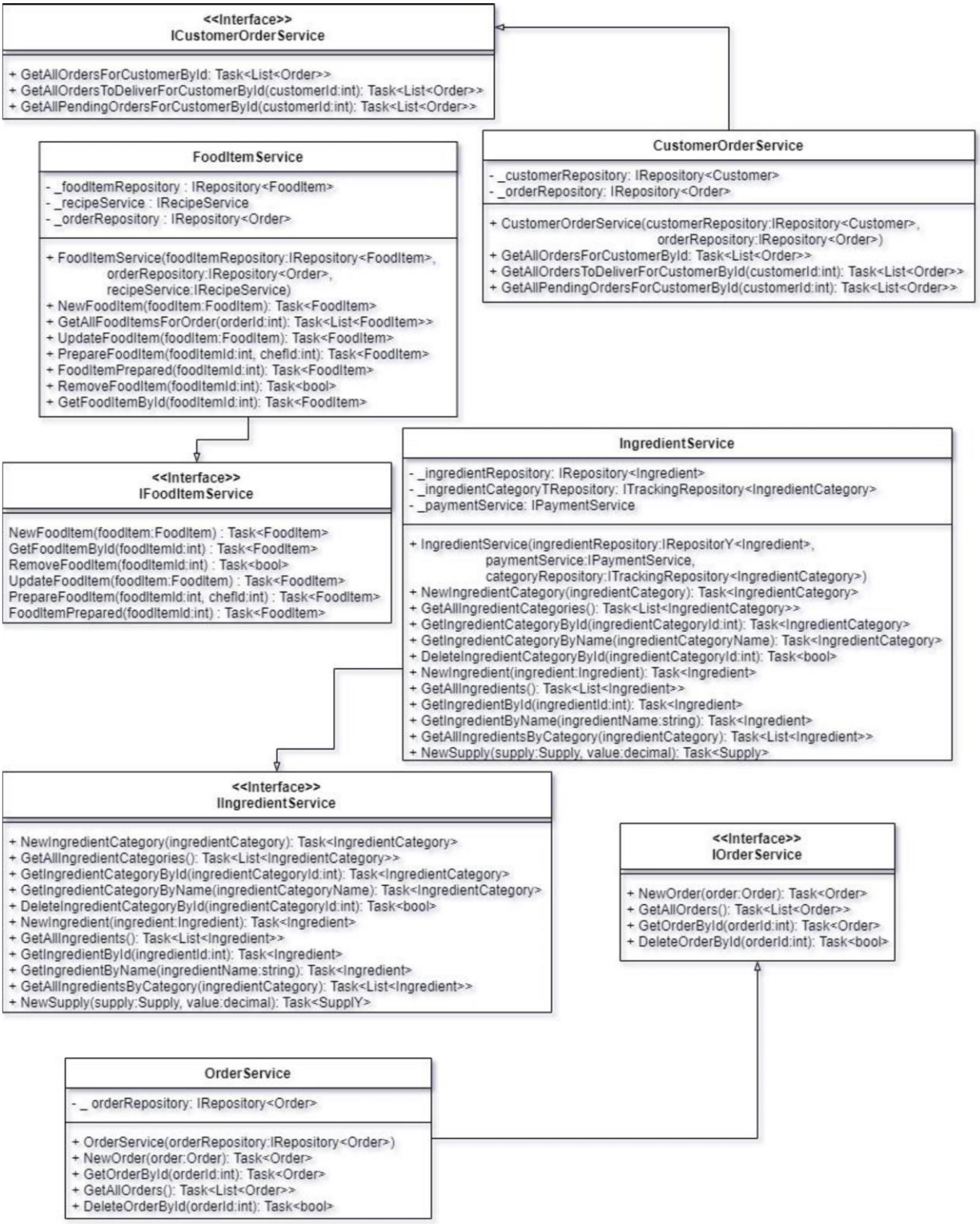


Figure 5. 5 Core Services 1. (Inheritance only).

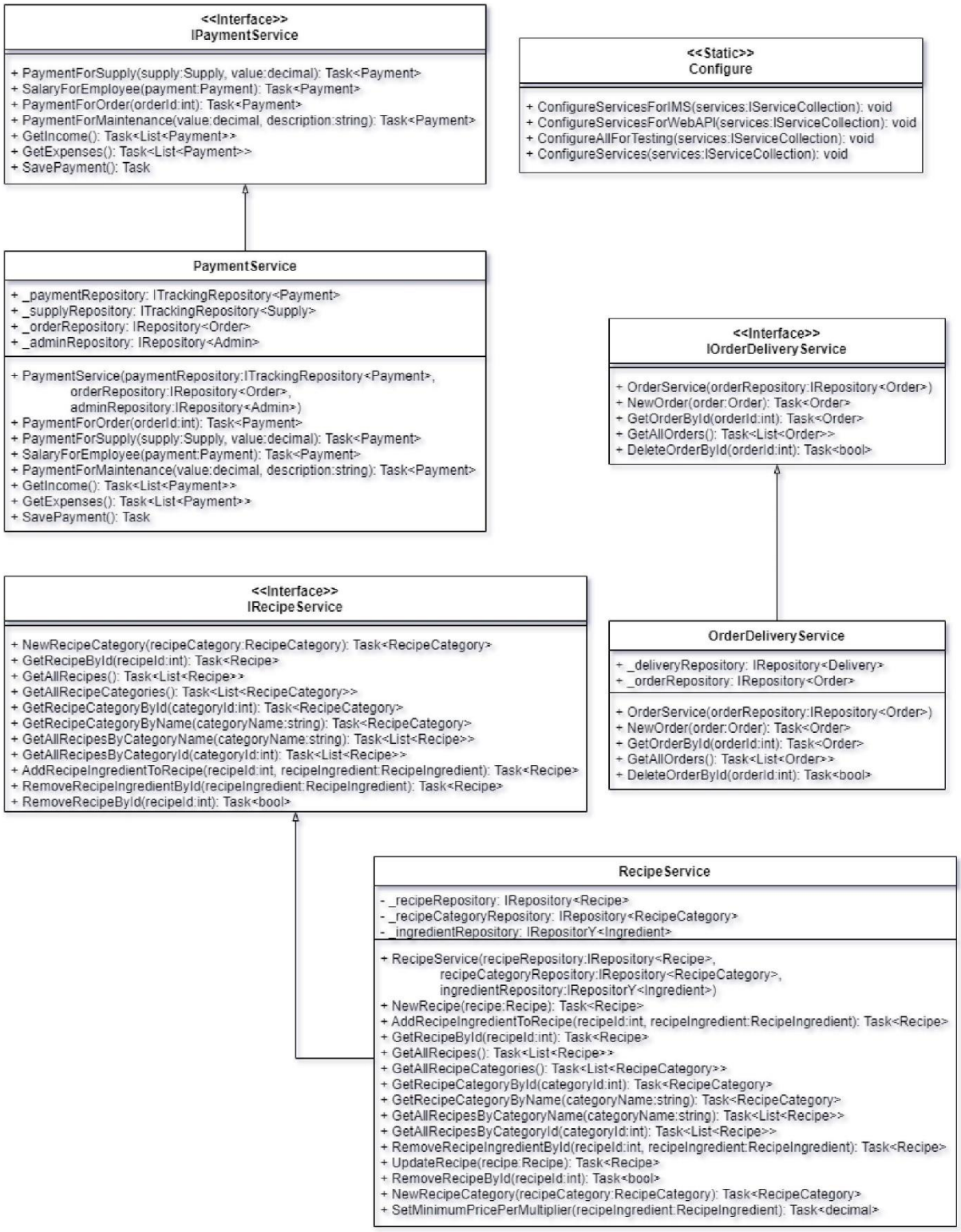


Figure 5. 6 Core Services 2. (Inheritance only).

Associations and dependencies for figure 5.4, 5,5, 5,6 as follows.

- AdminService depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Admin entity.
- ChefService depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Chef entity.
- CustomerService depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Customer entity.
- DelivererService depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Deliverer entity.
- SupplierService depends on IRepository and ITrackingRepository interfaces and has a composition relationship with the Supplier entity.
- FoodItemService depends on IRepository and IRecipeService interfaces and has a composition relationship with the FoodItem entity.
- CustomerOrderService depends on IRepository interface and has a composition relationship with the Customer and Order entities.
- IngredientService depends on IRepository, ITrackingRepository, and IPaymentService interfaces, and has a composition relationship with the ingredient entity.
- OrderDeliveryService depends on IRepository interface and has a composition relationship with the Delivery entity.
- OrderService depends on IRepository interface and has a composition relationship with the Order entity.
- PaymentService depends on IRepository, ITrackingRepository, and IPaymentService interfaces, and has a composition relationship with the Payment entity.
- RecipeService depends on IRepository interface, and has a composition relationship with the Recipe, RecipeCategory, and Ingredient entities.

The class diagram (figures 5.4, 5,5, 5,6) depicts several services and their respective interfaces that are used in the system. Each of these services implements a corresponding service interface, which defines the contract for the service and allows for loosely coupled code.

There are several associations and dependencies between the classes in the diagram. For example, the AdminService and ChefService both depend on some Repositories, which is used to access data storage. Additionally, the IngredientService depends on the PaymentService, which provides payment

functionality. The OrderDeliveryService and FoodItemService both depend on the Order Repository, indicating that they need to access and manipulate order data of the order entity in the data store.

The Configure class is a key component in setting up the dependency injection framework for the application, allowing for loose coupling between components and promoting code reuse and maintainability. (As discussed in Chapter 5.2.1)

5.3 HIGH-LEVEL ARCHITECTURAL DIAGRAM

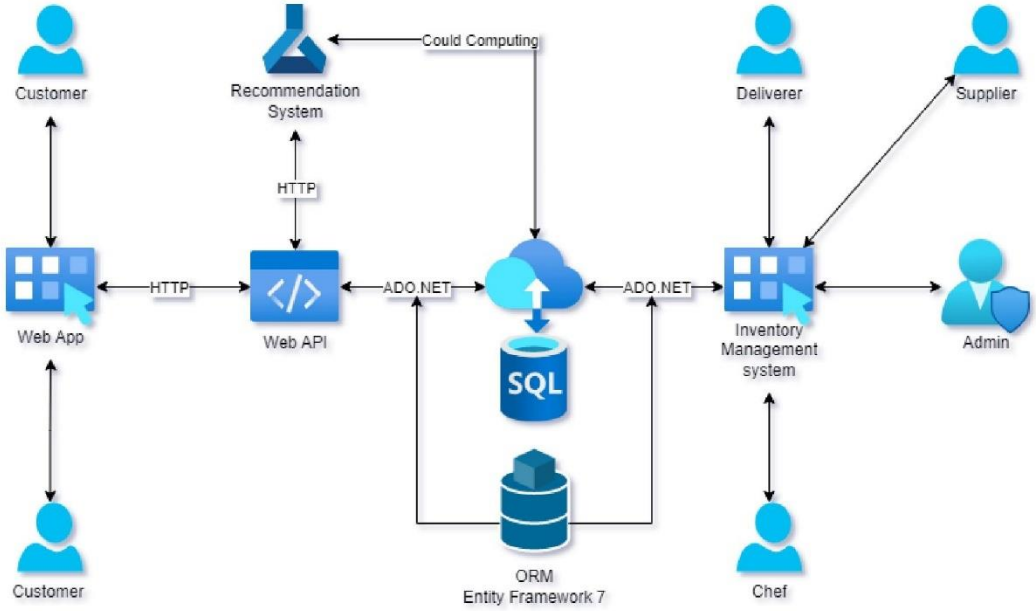


Figure 5. 7 High-level Architecture Diagram.

The architecture diagram (Figure5.7) consists of a distributed system of several components. A Web Application is hosted on Azure, which communicates with an API Layer to access the Cloud Storage. The Web Application serves customers who place orders, edit recipes, and leave reviews. The IMS is managed by Admin, Supplier, Chef, and deliverer. Additionally, there is a Recommendation System that communicates with the Web API over HTTP. Azure Cloud SQL is used as a database to store data related to employees, inventory, orders and recipes. This architecture allows for efficient management of inventory, orders, and employees in a restaurant setting.

5.4 PROJECT STRUCTURE

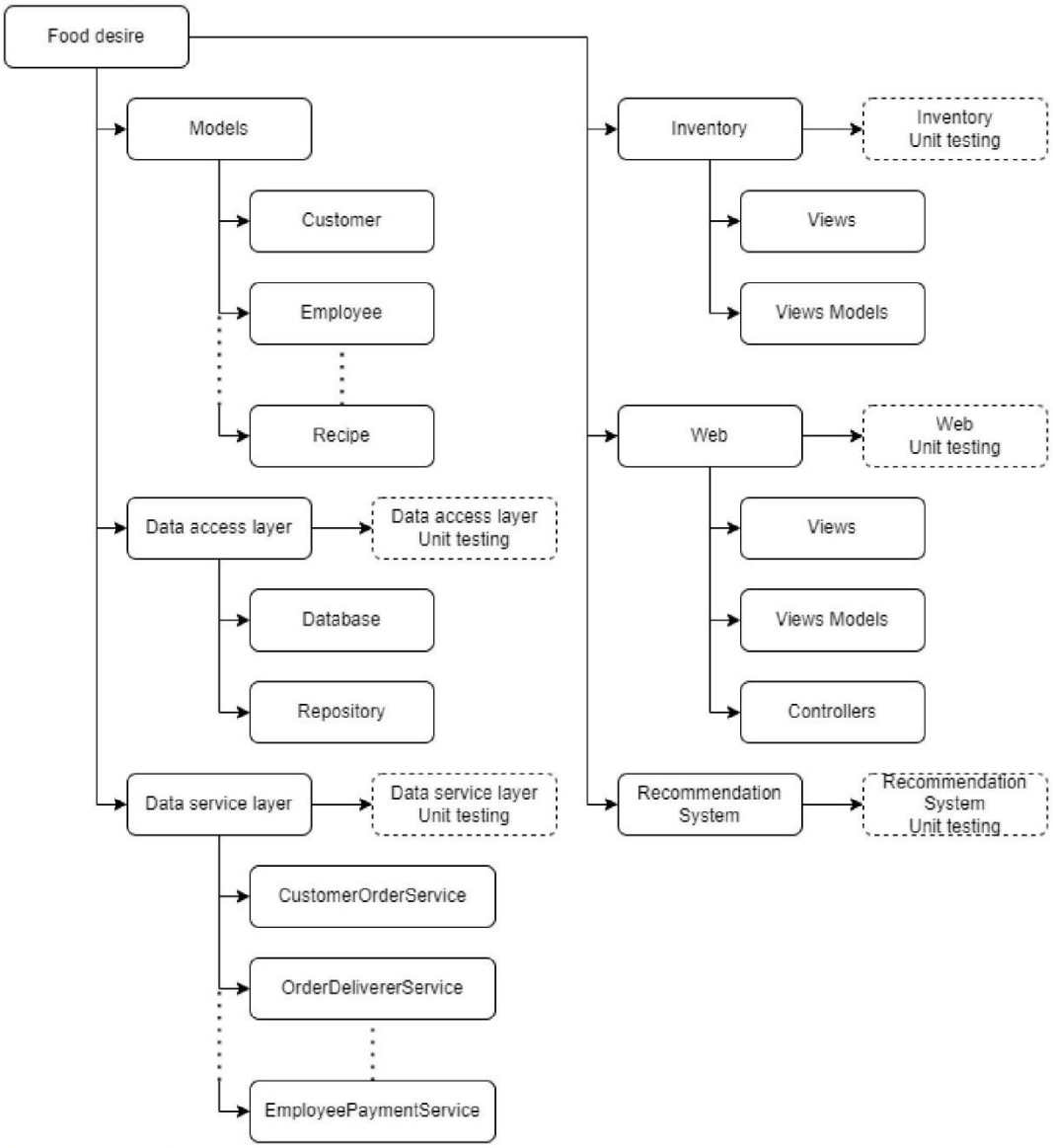


Figure 5. 8 Project Structure.

Chapter 06
DEVELOPMENT TOOLS AND TECHNOLOGIES

6.1 DEVELOPMENT METHODOLOGY

Requirements gathering is a crucial step in any software development project. It involves communicating with the stakeholders who have an interest in the system, such as customers, users, managers, and developers. The goal is to be ensure of their needs, expectations, and constraints for the system and document them clearly. A requirements specification document should be created that defines the scope, objectives, functionality, quality attributes, and acceptance criteria of the system. (Ref: Project initialization document, Chapter 03.)

Design is the next step after requirements gathering. It involves creating a high-level plan for how the system will be structured and implemented. The design should specify the system architecture and components, such as classes, interfaces, modules, and patterns. A high-level design document should be created that describes the rationale behind the design decisions and how they meet the requirements. (Chapter 05)

Implementation is the step where the actual coding of the system takes place. The development developer should follow the design document and use best practices for writing clean and maintainable code. The code should be well-commented and documented to facilitate understanding and reuse. The code should also adhere to coding standards and conventions to ensure consistency and quality. Correct use of good design pattern will help for the further development of the system.

Testing is the step where the system is verified and validated against the requirements. Testing should be done throughout the development process to ensure that each feature works as expected and meets its acceptance criteria. Testing can involve both manual and automated methods, such as unit testing, integration testing, system testing, performance testing, usability testing etc.

Integration and deployment are the step where all the features are combined into a complete system and delivered to a production environment. Integration involves ensuring that all components work together seamlessly without any

errors or conflicts. Deployment involves installing or uploading the system to a server or platform where it can be accessed by end-users.

Maintenance is the final step in software development lifecycle. It involves providing ongoing support for the system after it has been deployed. Continues integration and continues development techniques can be implemented for the system to improve the maintainability. Maintenance can include,

- Fixing bugs.
- Applying security patches.
- Adding new features.
- Enhancing existing features or improving performance.

This development methodology is based on the classic waterfall model, but it can be adapted to incorporate agile principles if desired. The key is to have a clear process in place for each stage of development, from requirements gathering to maintenance.

6.2 PROGRAMMING LANGUAGES AND TOOLS

The FoodDesire application is developed using the C# programming language, and its backend was created using the .NET 7 and Entity Framework 7 frameworks. For coding and debugging purposes, Visual Studio and Visual Studio Code being used for as development environments (IDEs).

Database management and querying are carried out using either SQL Server Management Studio or Azure Data Studio.

To ensure efficient version control, Git is used as the version control system.

To facilitate API documentation, Swagger is implemented, which provided a user-friendly interface for developers to explore and understand the different endpoints available in the application.

For project management and CI/CD pipelines, Azure DevOps being used to streamline the software development process and automate the deployment of the application. GitHub Action will also be used in the CI/CD pipelines.

In summary, the FoodDesire application was built using C# programming language and the .NET Core and Entity Framework Core frameworks, along with Visual Studio or Visual Studio Code for IDEs, SQL Server Management Studio or Azure Data Studio for database management, Git for version control, Azure DevOps for project management and CI/CD pipelines, and Swagger for API documentation.

6.3 THIRD PARTY COMPONENTS AND LIBRARIES

The system leverages several third-party components and libraries to deliver a robust and user-friendly solution. Entity Framework 7 (EF7) is the core component of the system as it provides a lightweight and extensible way to access data from various sources using C#. EF7 supports different types of databases, such as relational, non-relational, and in-memory. EF7 also enables code-first development, migrations, change tracking, and query optimization.

Microsoft.Extensions.Hosting, which is a library that simplifies the initialization and configuration of applications. Microsoft.Extensions.Hosting allows dependency injection to register services and components that can be resolved by the application at runtime.

The system also uses NUnit testing to ensure the quality and reliability of the code. NUnit is a unit-testing framework for .NET that supports parameterized tests, assertions, attributes, and test runners.

The IMS uses WinUI 3 library, which is a modern and fluent user interface framework for Windows desktop applications. WinUI 3 provides a consistent and adaptive design across various devices and platforms. The IMS also follows the Model-View-ViewModel (MVVM) design pattern to ensure a clear separation of concerns between the data model, the user interface, and the business logic. To facilitate the implementation of MVVM, the system uses MVVM Community Toolkit, which is a collection of helpers, extensions, converters, behaviors, and controls for MVVM development.

Another key component of the system is the Web UI, which allows customers to browse and order products online. The Web UI is built using Blazor framework, which enables web development using C#. Blazor offers several benefits such as

full-stack development with a single language, fast performance with Web Assembly runtime, rich interactivity with JavaScript interoperability, and code reuse with .NET libraries.

Swagger is a library that helps developers design, build, document, and test RESTful APIs. Swagger provides tools for generating interactive documentation from API specifications, validating API requests and responses, and testing API endpoints.

ML.NET is a machine learning framework for .NET developers that can be used to build recommendation systems. This will be used to develop the recommendation system of the Food Desire application.

By using these third-party components and libraries, the system achieves a high level of functionality and usability for both internal and external users.

6.4 ALGORITHMS

Content-based filtering: This algorithm can be used to recommend food items to users based on their previous orders or other preferences. The algorithm analyzes the content of food items and recommends similar items to users. For instance, if a user orders a lot of spicy food, the system may recommend other spicy food items to that user.

Collaborative filtering: This algorithm analyzes user behavior and interactions to recommend food items. The system can use this algorithm to identify patterns in user behavior and suggest food items that are popular among similar users.

Hybrid algorithm: A combination of content-based filtering and collaborative filtering can be used to provide more accurate recommendations to users. This algorithm analyzes both user preferences and similar user behavior to recommend food items.

Search algorithm: The system can use a search algorithm to allow users to search for specific food items or ingredients.

Overall, the algorithms used in the food desire application should be selected based on the specific requirements of the system and the data available. Entity Framework 7 can be used to retrieve data from the Azure Cloud SQL database and perform the necessary calculations to generate recommendations or perform searches.

Chapter 07
DISCUSSION

7.1 OVERVIEW OF THE INTERIM REPORT

The project aims to develop a web-based application using .NET framework that allows customers to order food online from the restaurants and customize their meals. The project also intends to implement a recommendation system that suggests meal items to the users based on their preferences and previous orders.

7.2 SUMMARY OF THE REPORT

Using a waterfall model methodology, the report outlines six stages of development that are crucial for the successful implementation of an online food ordering system. The first stage is requirements gathering, where the developer identifies and documents the functional and non-functional requirements for both customer and restaurant ends of the system. In the design stage, the developer uses the requirements to create detailed design specifications for the system, including the database schema, user interface design, and system architecture.

During the implementation stage, the developer begins coding the system using .NET framework, following best practices, and coding standards. The system is then tested in the next stage, where the developer ensures that all requirements have been met and that the system is functioning as expected. In the integration and deployment stage, the developer deploys the system to a web server using tools such as Apache software.

Finally, the developer must ensure that the system is maintained, by fixing bugs, applying security patches, adding new features, and ensuring the system is running smoothly. The report also provides some details on how each stage can be performed using various tools and techniques. For instance, in the testing stage, the developer can use tools such as NUnit and GitHub action to perform unit and integration testing, respectively.

7.3 CHALLENGES FACED

- Finding reliable sources of information on existing online food ordering systems

- Defining clear and consistent requirements for both customer end and restaurant end
- Testing the currently developed features thoroughly to ensure quality and functionality.
- Integrating currently developed components smoothly into one coherent system

7.4 FUTURE PLANS / UPCOMING WORK

Great progress has been made on the project with the completion of the DAL and CORE layers, as well as the unit testing for both layers. The next phase of the project will involve implementing the IMS UI, which will provide a user-friendly interface for managing inventory. Additionally, the developer will work on creating the web API and web UI, which will enable users to interact with the system and place orders online. Moreover, more research should be conducted to implement the recommendation system.

Overall, these upcoming works represent an exciting opportunity to bring the online food ordering system project to finished. The goal is to deliver a high-quality system that meets the needs of customers and restaurant.

References

Ahrens, S. (2012) 'Recommender Systems', in.

Daim, T.U. *et al.* (2013) 'Exploring technology acceptance for online food services', *Int. J. Bus. Inf. Syst.*, 12, pp. 383–403.

Gao, F. and Su, X. (2018) 'Omnichannel Service Operations with Online and Offline Self-Order Technologies', *Manag. Sci.*, 64, pp. 3595–3608.

Goffe, L. *et al.* (2021) 'Appetite for Disruption: Designing Human-Centred Augmentations to an Online Food Ordering Platform', in *34th British Human Computer Interaction Conference Interaction Conference, BCS HCI 2021*. BCS Learning and Development Ltd., pp. 155–167. Available at: https://doi.org/10.14236/ewic/HCI2021.16.

R., A. *et al.* (2017) 'Online Food Ordering System', *International Journal of Computer Applications*, 180(6), pp. 22–24. Available at: https://doi.org/10.5120/ijca2017916046.

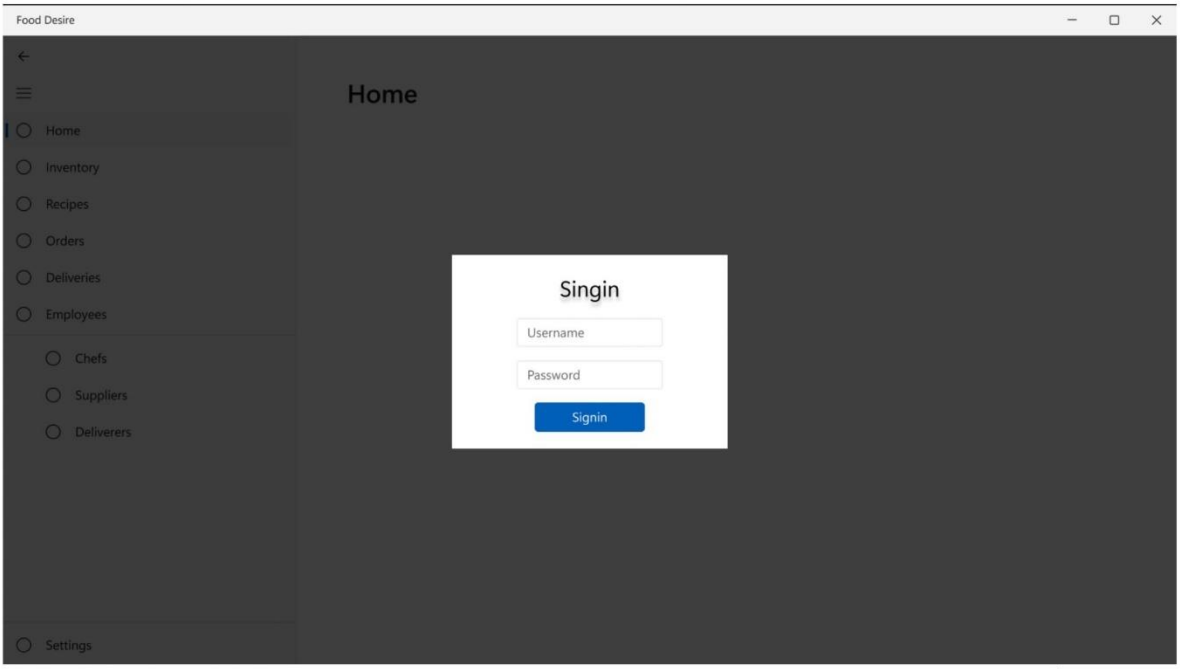
Ratto, F. *et al.* (2008) 'A numerical approach to quantify self-ordering among self-organized nanostructures', *Surface Science*, 602, pp. 249–258.

Thomas, A. and Davis, S.E. (1978) 'Laboratory ordering: a system which eliminates paperwork.', *The American journal of medical technology*, 44 10, pp. 1026–7.

Zulkarnain, K. *et al.* (2015) 'Key success factors of online food ordering services:an empirical study', in.

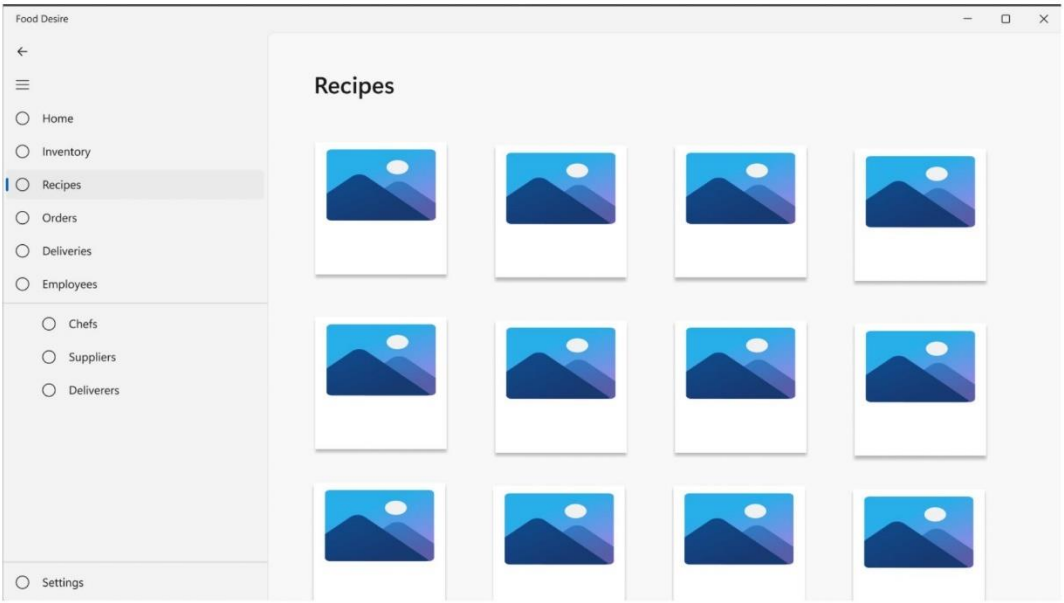
Appendixes

INVENTORY MANAGEMENT SYSTEM WIREFRAME.

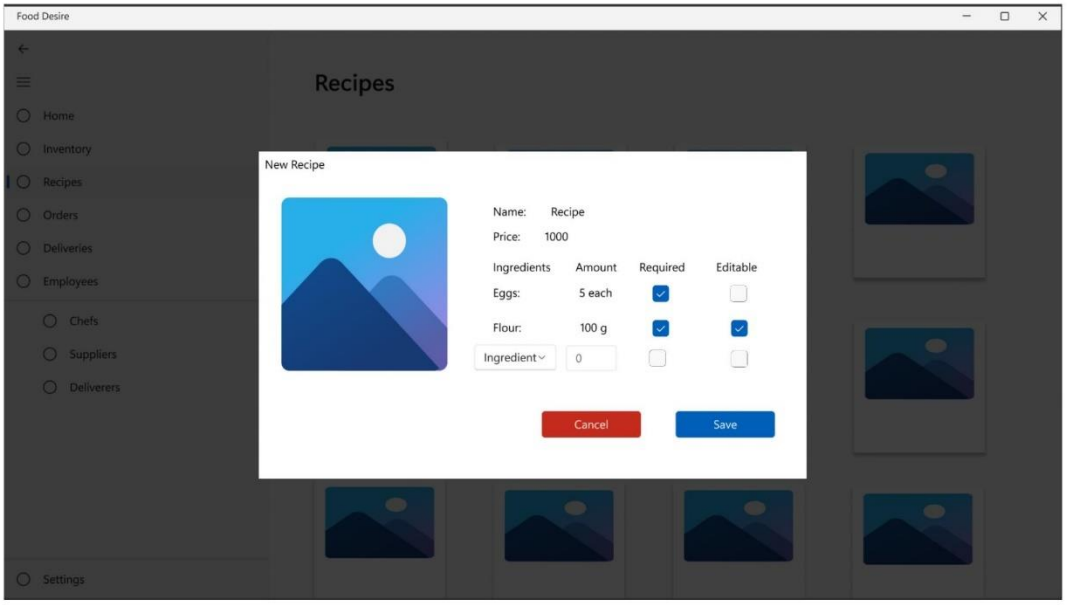


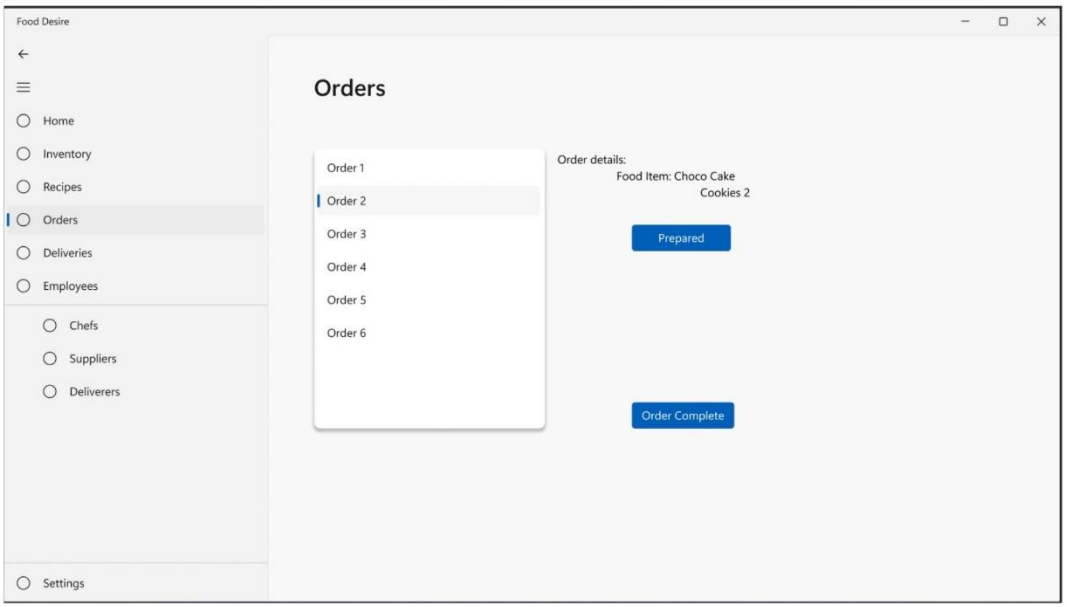
34

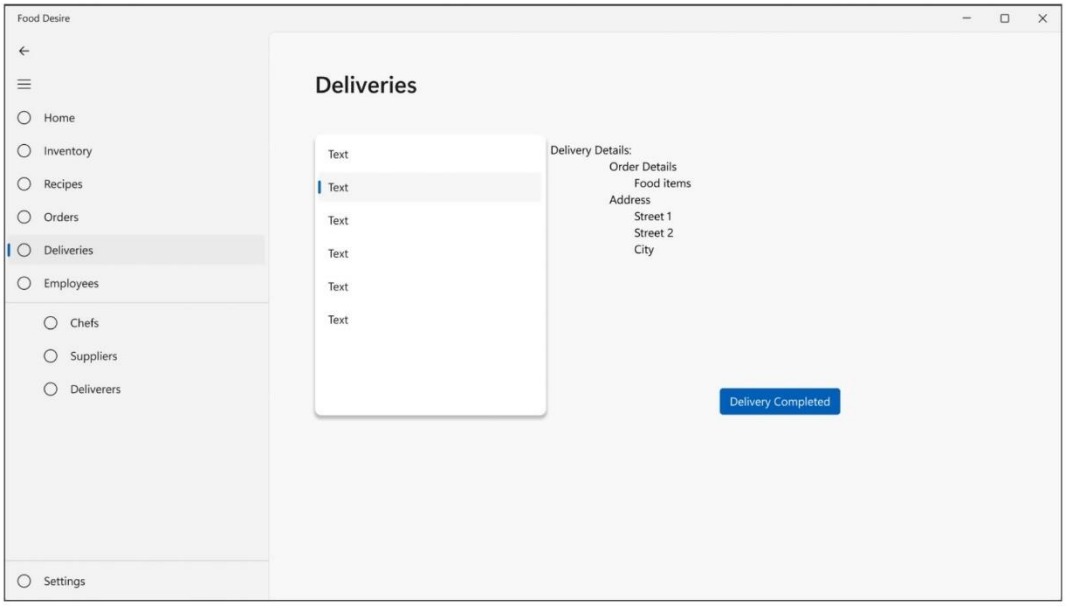

Food Desire
Inventory
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings


Food Desire
Recipes
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings

35


Food Desire
Recipes
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings
New Recipe
Name: Recipe
Price: 1000
Ingredients
Amount
Required
Editable
Eggs:
5 each
Flour:
100 g
Ingredient
0
Cancel
Save

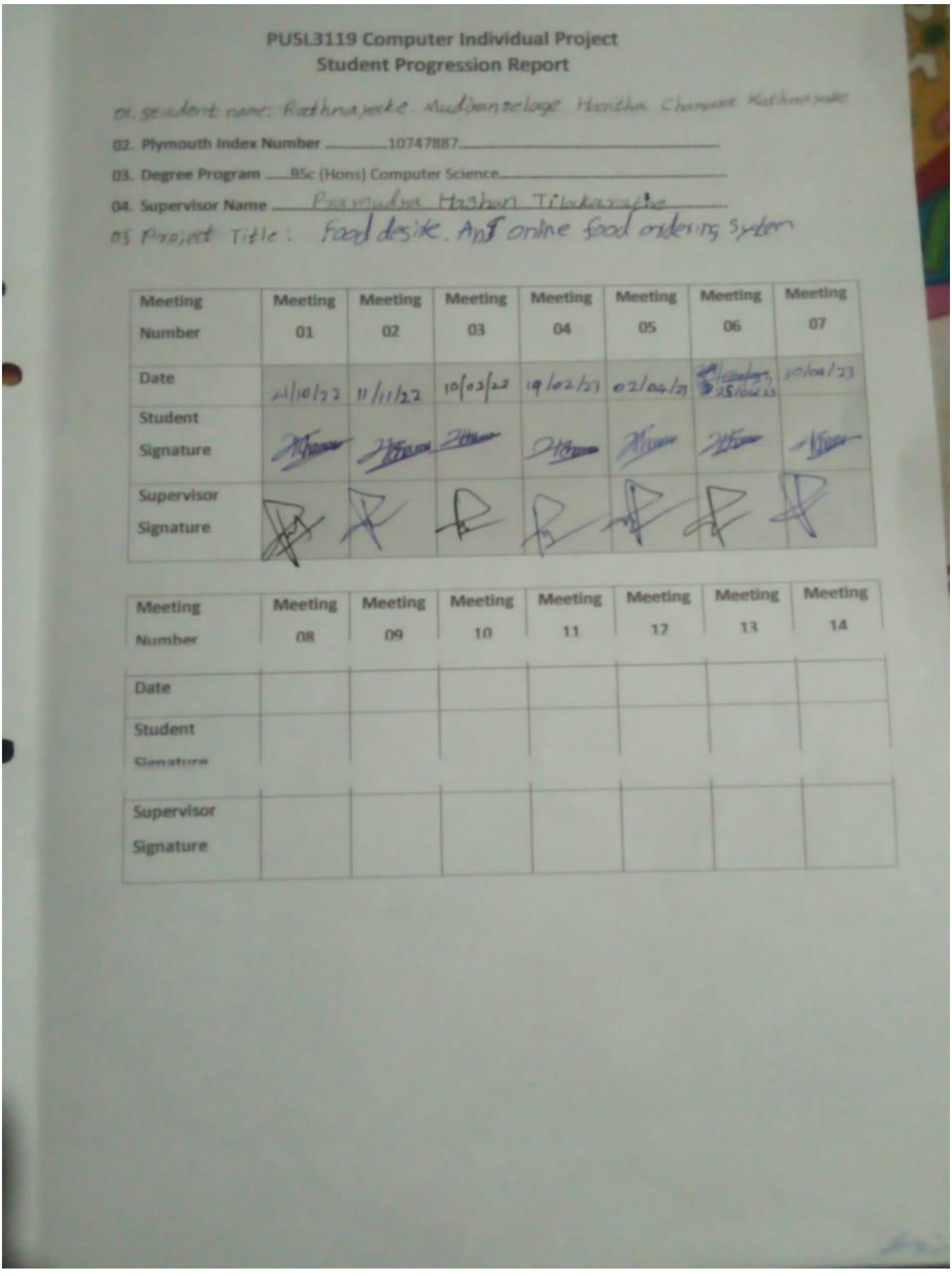

Food Desire
Orders
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings
Order 1
Order 2
Order 3
Order 4
Order 5
Order 6
Order details:
Food Item: Choco Cake
Cookies 2
Prepared
Order Complete


Food Desire
Deliveries
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings
Text
Text
Text
Text
Text
Text
Delivery Details:
Order Details
Food items
Address
Street 1
Street 2
City
Delivery Completed


Food Desire
Employees: Chefs
Home
Inventory
Recipes
Orders
Deliveries
Employees
Chefs
Suppliers
Deliverers
Settings
Text
Text
Text
Text
Text
Text
Chef details:

END

Student Progression Reports



PUSL3119 Computer Individual Project
Student Progression Report

01. Student name: Rathnayake Mudiyanselage Harsha Chamara Rathnayake
02. Plymouth Index Number ..... 10747887 .....
03. Degree Program ..... BSc (Hons) Computer Science .....
04. Supervisor Name ..... Prasadini Harshani Tilakaratne .....
05. Project Title: Food desire. App online food ordering System

Meeting Number	Meeting 01	Meeting 02	Meeting 03	Meeting 04	Meeting 05	Meeting 06	Meeting 07
Date	21/10/22	11/11/22	10/03/22	19/02/23	02/04/23	25/04/23	20/04/23
Student Signature							
Supervisor Signature							

Meeting Number	Meeting 08	Meeting 09	Meeting 10	Meeting 11	Meeting 12	Meeting 13	Meeting 14
Date							
Student Signature							
Supervisor Signature							

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 1

Date : October 21st, 2022

Project Title :

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Pramudya Hashan Tilakaratne

Items discussed:

Project idea approval

Items to be completed before the next supervisory meeting:

Project proposal Submission

21/10/2022

Supervisor (Signature & Date)

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 2

Date : November 11, 2022

Project Title : Food Desire, An online food ordering System

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr. Pramudya thilakarathne

Items discussed:

Scope of the project, ML/Fullstack

Items to be completed before the next supervisory meeting:

PID ~~form~~ Submission

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 03

Date : Feb. 10, 2023

Project Title : Food Desire An online food ordering system

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr. Ravindra Thilakarathne

Items discussed:

Implementation of Testing

Items to be completed before the next supervisory meeting:

Domain layer unit testing

10/02/23

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 03

Date : 19/02/2022

Project Title : Food Desire

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr Pramudya Thilakarathne

Items discussed:

Wireframe dagrams for IMS

Items to be completed before the next supervisory meeting:

IMS : Navigation services

Interim report : ~~At least 75%~~.
At least 75%.

19/2/22

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 05

Date : 2nd April 2023

Project Title : Food Desire

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr. Pramudya H. Thilakarathne

Items discussed:

Hosting the AI model for recommendation system

Items to be completed before the next supervisory meeting:

Complete the Web API

3/4/23

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 05

Date : 25 April 2023

Project Title : Food DeSire

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr. Pramudya H. Thilakarathne

Items discussed:

Current state of the recommendation system

Items to be completed before the next supervisory meeting:

Complete the web UI covering the functional requirments ~~of for Cost~~ for "customer role"

25/4/23

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.

UNIVERSITY OF PLYMOUTH

Final Year Project – Supervisory meeting minutes

Meeting No: 07

Date : 30 April 2023

Project Title : Food Desire

Name of the Student : Rathnayake Mudiyanselage Haritha Chamara Rathnayake

Students ID : 10747887

Name of the Supervisor : Mr. Pramudya H. Thilakarathne

Items discussed:

Current state of the Project.

Items to be completed before the next supervisory meeting:

Complete the Project / bring the Project to Production

Supervisor (Signature & Date)

Instructions to the supervisor: **Do not sign** if the above boxes are blank.