**CS 3343**

Software Engineering Practice

**Group Project – “Hot Meals”**

Group-1

**Design & Analysis Report**

**Group Members**

* Banbah, Kush: 55786740
* Gupta, Aarnav: 55990960
* Jain, Utkarsh: 55992915
* Kasliwal, Aryan Girish: 55972222
* Malhotra, Avi: 55773896
* Rajagopalan, Pratul: 55858290

**Table of Contents**

[Requirements Specifications 3](#_Toc89621154)

[Design Constraints 4](#_Toc89621155)

[Use Case Diagram 5](#_Toc89621156)

[Story Line 6](#_Toc89621157)

[Use Case Specification 7](#_Toc89621158)

[Class Diagram 8](#_Toc89621159)

[Sequence Diagram 9](#_Toc89621160)

[Design Principles 10](#_Toc89621161)

[Design Patterns 11](#_Toc89621162)

# Requirements Specifications

|  |  |
| --- | --- |
| Objectives | Requirements |
| Efficiency | Minimize waiting time for customers |
| Efficient seat management |
| Kitchen time management |
| Scalability | Multiple restaurant locations |
| Multiple dishes and dish types per location |
| Variable restaurant capacity |
| Profit maximization | Organize table reservation to always ensure maximal seat usage |
| First-to market, innovative solution |

After devising the above client objectives and the system requirements needed to achieve said objectives, the following solution goals were agreed upon:

1. Customers should be able to make a booking with their order and get a time to arrive from the restaurant.
2. They should be able to make a booking for several people at any location.
3. They can delete or view booking details later.
4. Once a booking is made, the time selected must be free enough for the restaurant to be able to prepare the meal and have seats so the customer doesn't have to wait.

# Design Constraints

The software itself was technically feasible, given that our market research yielded no such existing products. Java was used as the main programming knowledge to build the software, nonetheless, there were certain design constraints that needed careful consideration as they impacted the scope of the project consequently. While these constraints did to some extent impact the user features that were ultimately released, we ensured that all requirements and objectives were met without any qualitative and quantitative hindrance.

The primary design constraint was perhaps the console itself, which limited the manner in which the system entertained user input and responses with user queries as output. It is worth mentioning that the idea of generating a Graphical User Interface (GUI) or a front-end interaction system was strongly considered during the initial few meetings. However, this proposal was then unanimously rejected since it would have resulted in limited testing. Since application of the testing strategies and methodologies learnt in class was one of my main objectives of the group project, the design constraint of the console thus became an opportunity cost.

The above design constraint resulted in a secondary constraint: all reservations needed to be generated sequentially. In other words, the software could not host concurrent users. Due to a limitation in console I/O, the “Zaizaeriya” system could not allow multiple users to make reservations simultaneously. However, the caveat here is that our design software needs no longer check for overlapping and conflicting reservation queries within the system. In other words, as two users cannot book a reservation at the same time, the backend system becomes a bit more manageable. Nonetheless, for future scope, if the need to allow concurrent users arises, the current design constraint can be overcome by using asynchronous programming and multithreading (via mutual exclusion and semaphores).

Finally, input checking needs to be enforced in the application by hard coding user paths to options, as opposed to simply displaying options in a GUI system.

# Use Case Diagram

Diagram

Description automatically generated

The above use case diagram is tailor made according to what the client’s experience with our application system will be. As you can see, the users have two primary use cases: they can choose to either make a reservation or view that reservation. Making a reservation is the first step the user takes but before sending a request to the system to reserve a table, the user has to enter their reservation details - namely the restaurant’s location, the number of accompanying guests, and decide on a menu from a given list.

The backend system, which does the computing to calculate the available reservation times, then sends a list to the user, who confirms their reservation time. Consequently, the user is directed to make their payment to reserve the table. It is worth mentioning that we do not have an external API as our third-party payment system, as using external APIs was discouraged by the professor since it makes testing difficult.

Lastly, A reservation ticket is then generated, and the user can view their reservation or alternatively, cancel it, in which case they are refunded their amount. Again, such functionality is not entirely implemented in our software and is hence handled by an external actor.

The entire user experience is surmised in the next section of this report.

# Story Line

Diagram, schematic

Description automatically generated

# Use Case Specification

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Making a new reservation | |
| **Actor(s):** | User and Backend System | |
| **Description:** | This use case specification documents the user’s journey while interacting with the app, particularly while making a new reservation to reserve a table and place an order. | |
| **Typical Course of Events:** | **User** | **System** |
| 1.  The user selects a restaurant location, enters the number of diners, and decides the dishes for dining.  3.  The user confirms their reservation time.  5a.  The user successfully makes payment. | 2a.  The system computes the list of available reservation times according to user input and returns the list.  4.  The system redirects the user to make payment. |
| **Alternate Courses:** | 2b.  The system could not find an available reservation time. Subsequently, the user is redirected to the home page.  5b.  The payment is unsuccessful, in which case the user can retry making a payment or go back to the menu. | |
| **Pre-condition(s):** | None | |
| **Post-condition(s):** | The user is given their booking id for the reservation ticket. | |
| **Assumption(s):** | None | |

# 

# Diagram Description automatically generatedClass Diagram

# Sequence Diagram

Chart

Description automatically generated with low confidence

# Design Principles

Much of the knowledge gained in the previous course – CS3342 Software Design – was incorporated into designing this project. Almost all the SOLID principles taught in the course were utilized.

The Open-Close Principle, Law of Demeter, and the Liskov Substitution Principle were implemented thoroughly. As the class diagram depicts, all classes are open for extension and closed for modification. In other words, while the functional behavior of every class can be changed readily, the important attributes inside classes are declared private and thus they are not directly accessible or modifiable. Furthermore, the degree of separation between classes is 1, thereby validating the principle of least knowledge. The subclasses designed for this project are also fully substitutable with their parent classes, which reinforces the compliance needed between parent-child classes. For example, this functionality is evident in the ‘Appetizer’, ‘MainCourse’, and ‘Dessert’ classes, which are subclasses of the Abstract ‘Food’ class.

Graphical user interface, text

Description automatically generated with medium confidence

*Single Responsibility Principle*

Finally, we also utilized the Single Responsibility Principle specifically for the following classes. The classes ‘DeleteBooking’, ‘TickClock’, and ‘CheckBooking’ exist for one and only one purpose, offering a more flexible design and categorizing class responsibilities accordingly.

# Design Patterns

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated

*Design Patterns*

The Zaizaeriya application design uses the Singleton and Facade Design Patterns. The ‘FlowController’, ‘Clock’, and ‘Restaurant’ classes are all implemented as Singletons. There can only be one instance of the flow controller, which stores our driver code. In addition, it makes sense to have only one instance of restaurant class, “Hot Meals” has designed the “Zaizaeriya” application for a single restaurant with multiple locations. To make sure that system time is consistent, the clock class was implemented as a singleton as well. It contains an inner class called iterator, which is where we define the intricate details of the clock. The other classes do not need to know the inner details of the clock, which is why we decided to use a facade pattern here.