OTLOB DELIVERY SYSTEM

CSE232: ADVANCED SOFTWARE ENGINEERING MAJOR TASK

Student ID	Student Name
19p6517	Abdelrahman Aboel Nasr
19p7817	Ibrahim Ashraf Ibrahim
20p5552	Muhammad Amr Fathy Muhammad Nasef
1900804	Omar Ayman Ayoub Abdelshafi
21p0188	Omar Mohamed Shams Eldeen
19P7343	Sohayla Ihab Abdelmawgoud Ahmed Hamed

Table of Contents

1. Introduction

1.1. Purpose

This document describes the full procedure of software engineering and lifecycle; detailed analysis and design, a summary of development, implementation, testing and maintenance. This document caters to all potential stakeholders who take interest in understanding the fundamentals and flesh of Otlob Delivery System, whether they be administrators, software developers, career-holders or even customers.

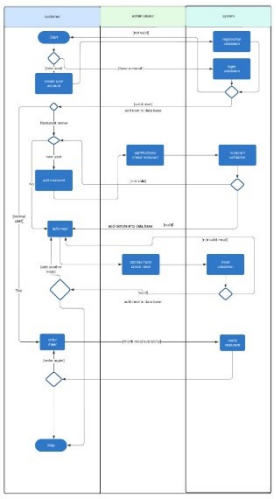


Figure 1: Sample of UML diagram

This project aims to show the steps of developing a software system based on object-oriented principles. The steps and methodology explained in this project can be used for any software system such that system goals and use cases are taken into consideration. The different paths taken in terms such as software lifecycle and architecture in place of others will be discussed in details to fulfill requirements of Otlob Delivery System.

1.2. List of Definitions

1. UML: Unified modeling language: developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems
2. OOP: Object Oriented Programming: developing software on the basis of objects
3. Functional Requirements: Capability requirements
4. Nonfunctional Requirements: Constraint requirements
5. Domain Requirements: depend on domain of work
6. Software Constraints: what the user expects for a trusted software
7. Adaptability: application must continually change to meet the demands of the customer and the market.
8. Reliability: expectations of the users as to the continual availability of the Internet application (including performance and fault tolerance requirements).
9. Scalability: adapt to the growth of the desktop application without the continual need to throw everything away and start over
10. OOAD: Object Oriented Analysis and Design: a software engineering approach that models a system as a group of interacting objects
11. OMT(Object Modelling Technique): an object modeling approach for software modeling and designing
12. Narrative Description: how a component or use case behaves correctly in the system with others
13. CRC(Class Responsibility Collaborator): brainstorming tool used in the design of object-oriented software
14. Architectural Model: illustrates software tradeoffs that are specific to the system
15. MVC (Model-View-Controller): pattern in software design commonly used to implement user interfaces, data, and controlling logic
16. Time Plan: a system that allows software development collaboration

1.3. Scope

The description of the written software system involves an outline of the products or services the software aims to provide. Otlob Delivery System allows different levels of users a reliable interface to order and deliver meals at the convenience of their smartphones/laptops.

1.4. List of References

[1] Patterson's Software Engineering for the Practitioner

[2] https://comfyliving.net/food-delivery-statistics/

[3] https://statista.com/outlook/dmo/eservices/online-food-delivery/worldwide

[4] https://dailynewsegypt.com/2018/08/30/21-of-egyptian-consumers-use-restaurants-meal-delivery-services-nielsens-report/

1.5. Overview

This document is divided into page border colour coded sections based on both the stage of software engineering and interested stakeholders. The software engineering stages consist of:

1. Identifying the software capabilities, constraints and functional environment
2. Specifying the identified capabilities and their corresponding stakeholders and perspective
3. The software analysis stage proceeds with different UML diagrams
4. Software architecture and Object Methodologies
5. Summary of software development stage and user guide
6. Functioning and implementation of internal components of the software
7. Testing and time plan

The stakeholders discussed in this document are:

1. Database administrators
2. Software administrators and developers: focused on in light blue
3. Restaurant managers
4. Customers

2. General Description

2.1. Product Perspective

The client discussed here is the customer who orders and pays for meals. Customers should be logged in the software to be able to functionally use it. This can be enabled by login through Google or through mobile phone number. In addition, the customer registration and login both require two-factor authentication, i.e. a validation key is sent to the phone number entered.

Payment coordination software may also be required for customers to be able to confirm cash on delivery payments and perform bank related payments.

Delivery/order tracking software may be necessary for both the customer and the restaurant manager.

2.2. General Capabilities

The main capabilities of Otlob Delivery System are shown below in terms of their appropriate stakeholders.[1]

Database Administrators:

1) Save customer and order information

2)Manage software database

Software Administrators and Developers:

1)Use their accounts to verify information such as login and registration

2)Accept restaurant additions to system

Restaurant Managers:

1)Use their accounts to add or delete restaurants

2)Add or delete meals

3)Approve meals and check their availability for customers

Customers:

1)Use their accounts to order meals

2)Complete Purchases

Figure 2: Types of stakeholders

[1] See Use Case, Swim lane and Class diagrams

2.3. General Constraints, Background Information, and Justification

2.3.1. Software background

Otlob Delivery System is a relatively small scale software that is mostly human-run (i.e. not fully automated). There are different types of users, with each having their own privileges. The user registers his username, password and phone number. A validation message is sent to users' phone numbers.

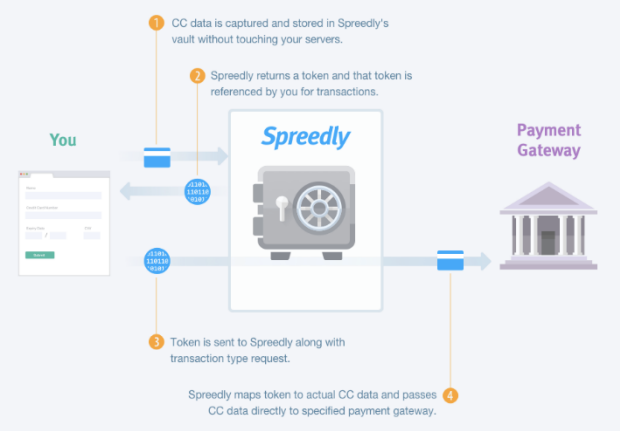


Figure 3: Sample of payment software

The customer is allowed to select and order meals. The restaurant owner is only supposed to edit restaurant menu. Only the database manager is allowed to verify an added restaurant. When a meal is ordered, the restaurant is notified to check for its availability. An invoice with the final order details is displayed after order is confirmed.

Otlob Delivery System assumes that users are capable of using other necessary software such as the aforementioned payment software, otherwise called payment orchestration. The software also assumes customers, restaurant managers and administrators will remember their login details. The meal availability and delivery coordination are the responsibility of the restaurant itself. The payment software is in cognition of both the customer and the restaurant. After restaurant approval, user information is sent to be recorded in database.

2.3.2. Software constraints

Software constraints are nonfunctional and domain requirements that may interfere with the functioning of a software.

2.3.2.1. Adaptability Requirements

During the software design and development stage, a software goes through many iterations and versions to meet the user requirements. Otlob Delivery System is developed with clean and understandable code. The architecture involved is an MVC[2] model. This means that our software is easily modifiable and has a faster development process.

2.3.2.2. Reliability Requirements

Otlob is available on a 24-hour-per-day basis. However, in case a user orders a meal during closing hours of a restaurant, the meal is placed on queue until the restaurant first opens, then the user gets a notification to confirm their order.

Due to its human-based functioning and scale, the mean time between software failures is minimal to none. There is also a back-up database for the user data. In the domain of food and hospitality, Otlob ensures the quality of its restaurants through verification and approval by the database manager.

[2] See architecture section

2.3.2.3. Scalability Requirements

The compliance with scalability requirements ensures the depletion of software development and maintenance costs. There are two different types of growth a software should be scalable to, vertical growth and horizontal growth.

Otlob can be scaled to support advertisements in order to increase sales of restaurants (vertical growth). It can also support multiple meals that can be added to all restaurants (horizontal growth).

2.3.2.4. Security Requirements

Otlob ensures to each user a username and a password. Users are also required to enter their phone numbers in order to verify their identities. Every user is sent a unique validation key. Restaurant authenticity is ensured as only the database manager can verify restaurants. Only restaurant-appointed managers are allowed to add/delete meals from restaurants. This decreases the possibility of fraud.

2.3.3. Software Justification

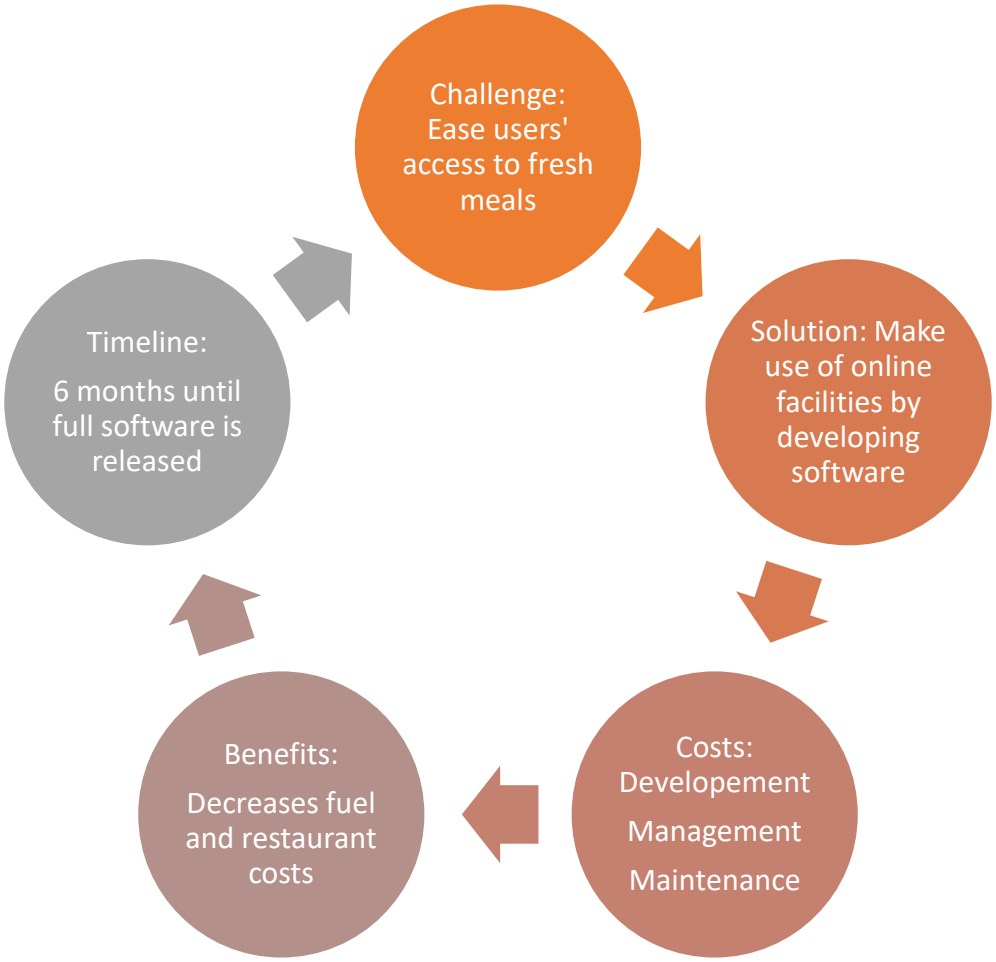


Figure 4: Summary of software justification

2.4. User Characteristics

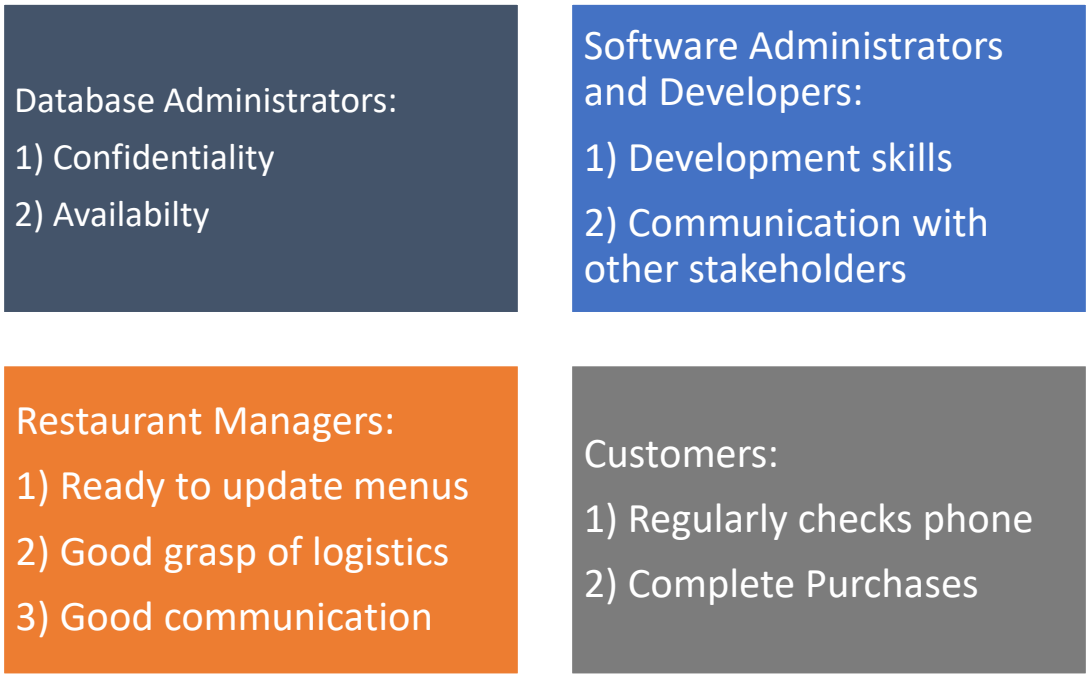


Figure 5: User characteristics

2.5. Environment Description

Operational environment are the hardware and software products installed in the users' or clients' facilities in which the component or system being tested will be used. The software can include operating systems, database management systems, and other applications. Otlob is a desktop application developed in Java on the Netbeans IDE. It can function on any operating system and doesn't require extensive network engineering.

2.6. Assumptions and Dependencies

Assumptions in this software are a good interface and responsive payment applications, restaurant users and delivery tracking.

Dependencies include background checks on restaurants and external deliveries and/or bank payments.

3. Specific Requirements

3.1. Capability Requirements

Most requirements can already be inferred from the above text, but since Otlob is a small-scale software, even its capability requirements aren't numerous. The capability requirements of Otlob software include restaurant related functions, verification of restaurant meals through meal ids, photos, or videos and issuance of customer invoice.

3.2. Constraint Requirements

In addition to software constraints[3], requirements of Otlob include an OOP language, Java in this case, and a copy o f customer invoices in the database to ensure quality control of the services provided.

Figure 6: Otlob offers all types of food and drink

[3] See section 2.3

4. Use-Case Diagram and the Narrative Description

4.1. Use Case Diagram

The use case diagram of the entire software system is shown below. The details of the diagram and iterations of proposed software shall be included in the Jacobson Methodology[4].

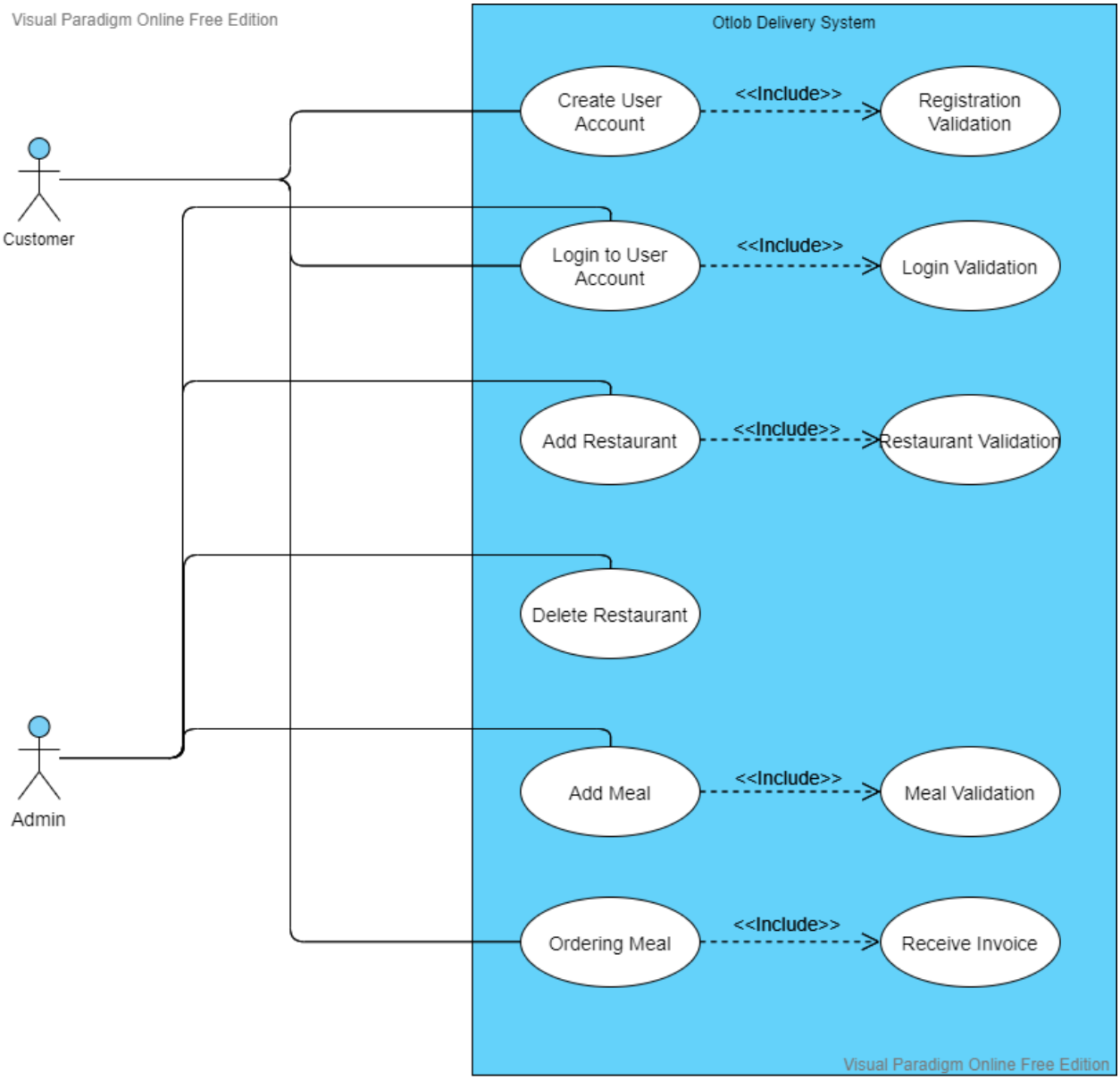


Figure 7

[4] See section 11 and 12

4.2. Narrative Description

4.2.1. Create User Account

Use case name	Create User Account
Goal	Creation of new customer account
Precondition	User browses main page
Primary actor	Customer
Successful condition	New account is created
Failed condition	User failed to register his account
Main flow	User start the main page User clicks on registration button User fills data User clicks submit Account created successfully Include:: Send registration validation key
Extensions	Account creation failed due to duplicated data

Table 1

4.2.2. Login to User Account

Use case name	Login to User Account
Goal	User log to his account to order his meal
Precondition	User browses main page
Primary actor	Customer
Successful condition	The user is logged in
Failed condition	User fails to login
Main flow	User start the main page User clicks on login as customer User fills data User clicks submit Include:: Send login validation key User logs in to his account
Extensions	login failed due to mismatched data

Table 2

4.2.3. Add Restaurant

Use case name	Add Restaurant
Goal	Admin adds new restaurant
Precondition	admin is in admin panel
Primary actor	admin
Successful condition	New restaurant is added
Failed condition	No restaurant is added
Main flow	Admin clicks on add restaurant Admin types restaurant name Admin clicks submit New restaurant is added
Extensions	Adding restaurants failed as there is an existing restaurant with same name
Use case name	Delete restaurant
Goal	Admin deletes existing restaurant
Precondition	admin is in admin panel
Primary actor	admin
Successful condition	existing restaurant is deleted
Failed condition	No restaurant is deleted
Main flow	Admin clicks on delete restaurant Admin chooses restaurant name Admin clicks submit existing restaurant is deleted
Extensions	deleting restaurants failed as there is no existing restaurant with same name

Table 3

4.2.4. Add Meal

Use case name	add meal
Goal	Admin add new meal to existing restaurant
Precondition	admin is in admin panel
Primary actor	admin
Successful condition	New meal is added to existing restaurant
Failed condition	No meal is added
Main flow	Admin clicks on add meal Admin chooses the restaurant Admin fills the meal information Admin clicks submit New meal is added
Extensions	Adding new meal is failed as there is existing meal with same name

Table 4

4.2.5. Ordering Meal

Use case name	Ordering Meal
Goal	User place an order
Precondition	User is in user panel
Primary actor	user
Successful condition	New order is placed
Failed condition	No order is placed
Main flow	User chooses restaurant name User choses meal name User choses quantity User choses payment options and fills its date User clicks submit Order is placed
Extensions	No order is placed due to wrong credit card data

Table 5

5. Swim-lane Diagrams

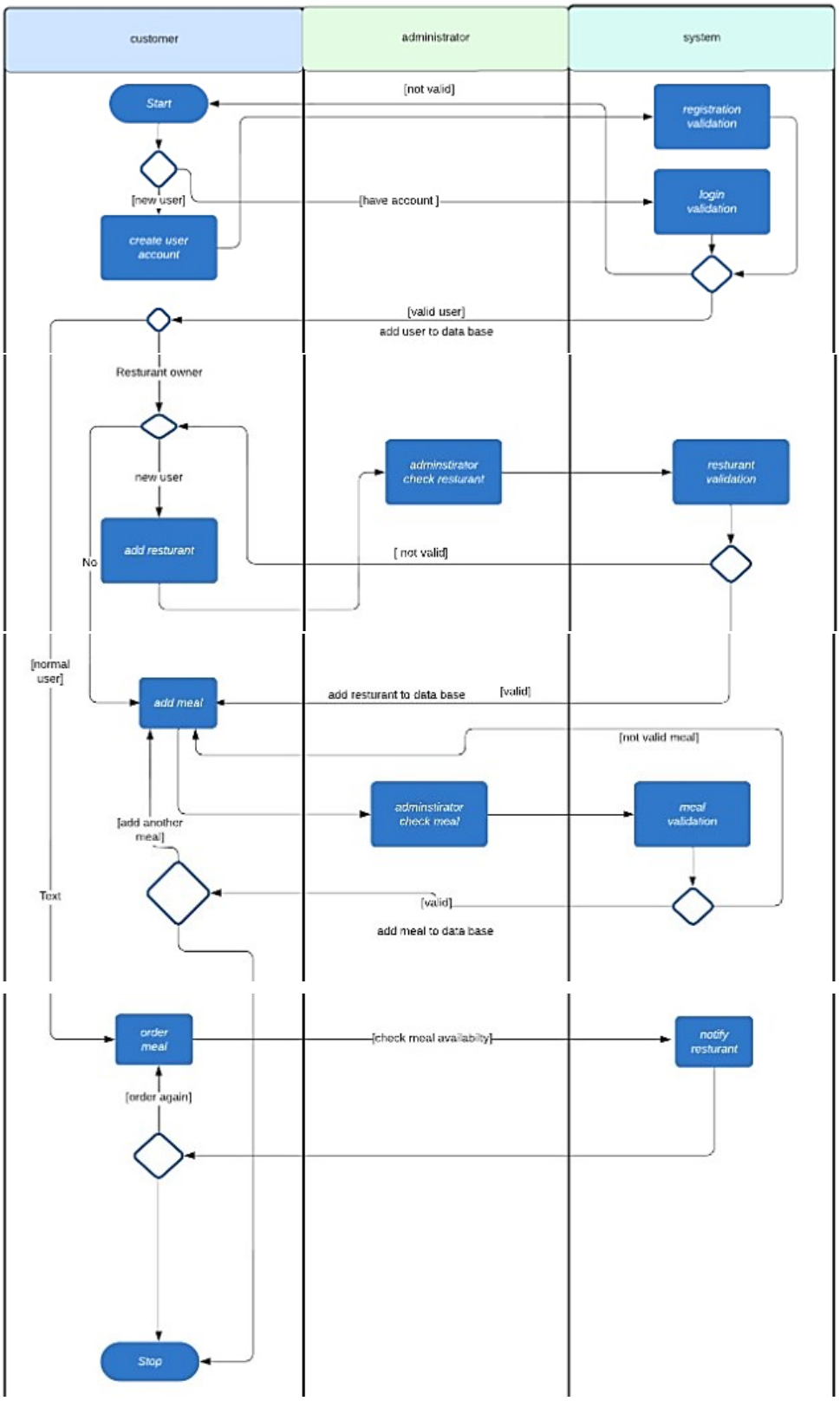


Figure 8

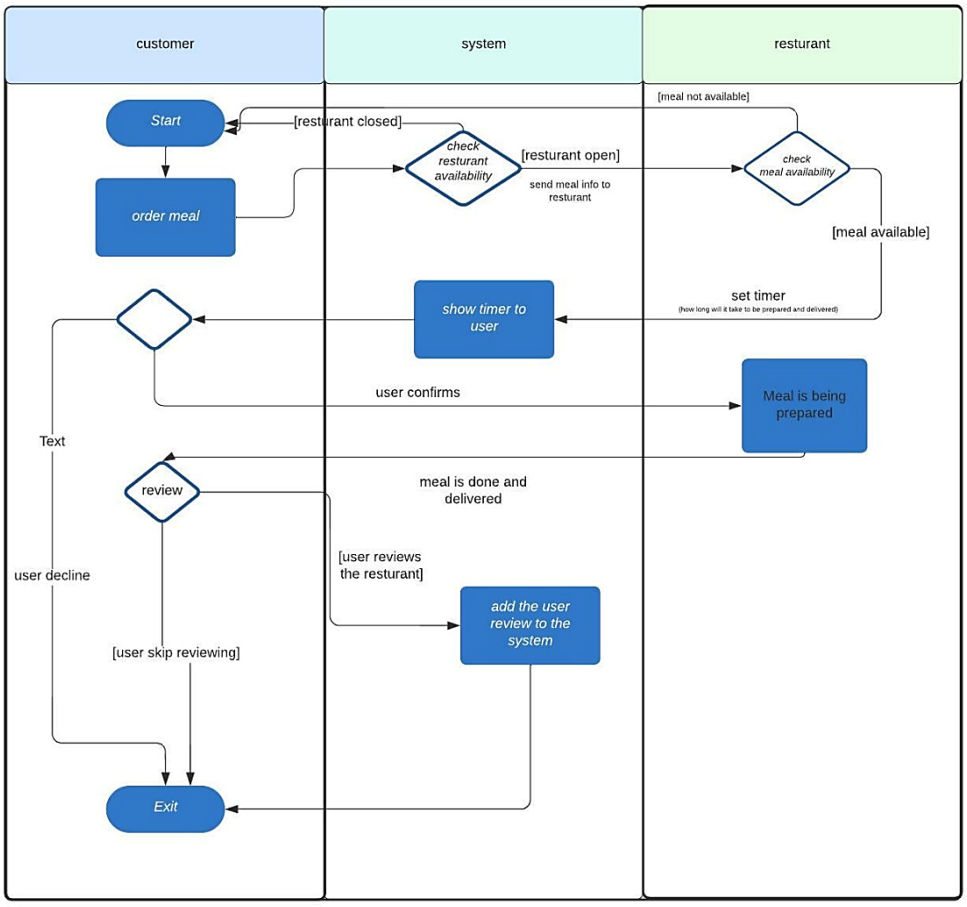

customer
system
resturant
Start
[resturant closed]
[meal not available]
check resturant availability
[resturant open]
send meal info to resturant
check meal availability
order meal
[meal available]
set timer
(how long will it take to be prepared and delivered)
show timer to user
user confirms
Meal is being prepared
Text
review
meal is done and delivered
user decline
[user reviews the resturant]
add the user review to the system
[user skip reviewing]
Exit

Figure 9

6. Interaction Diagrams

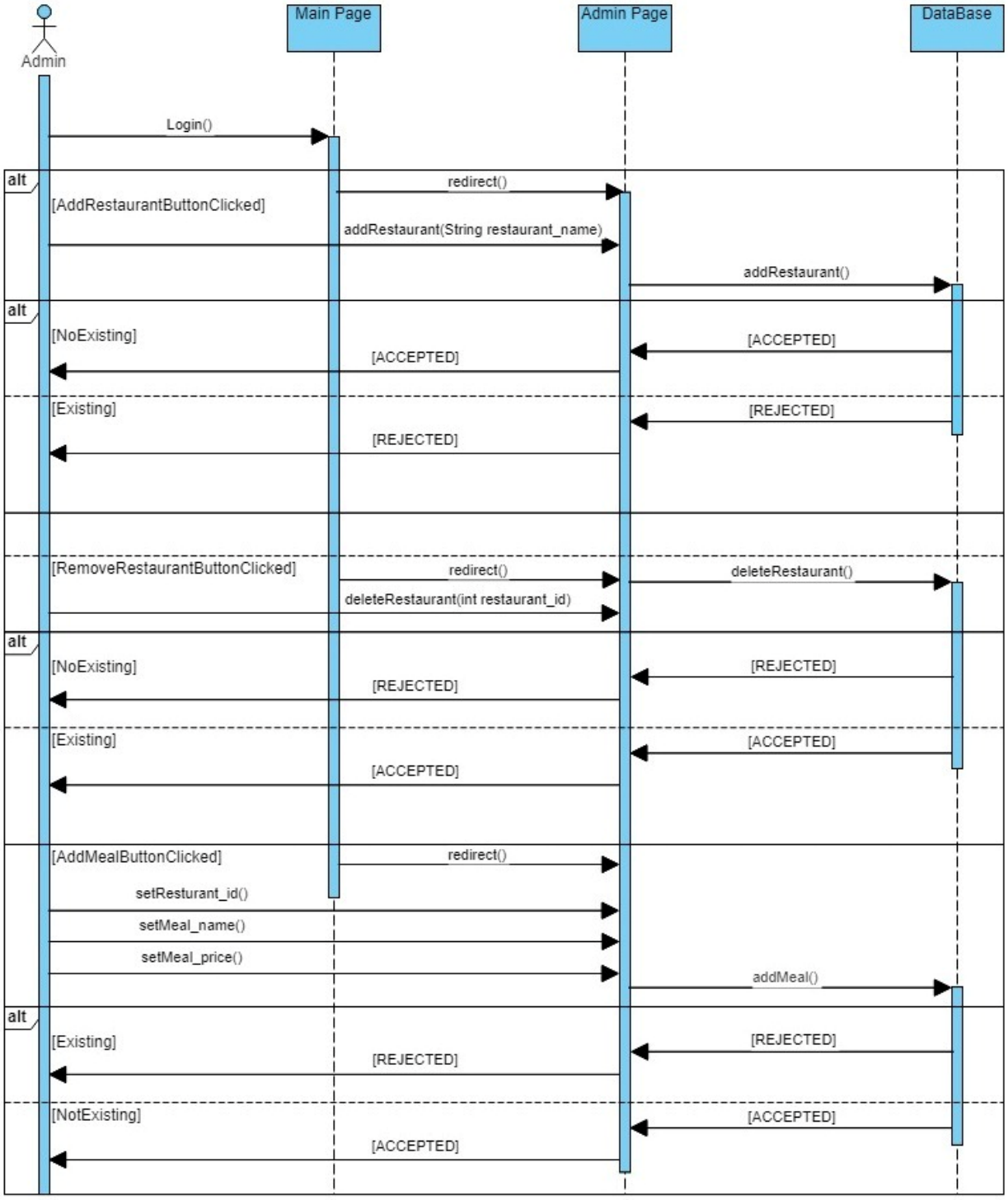


Figure 10

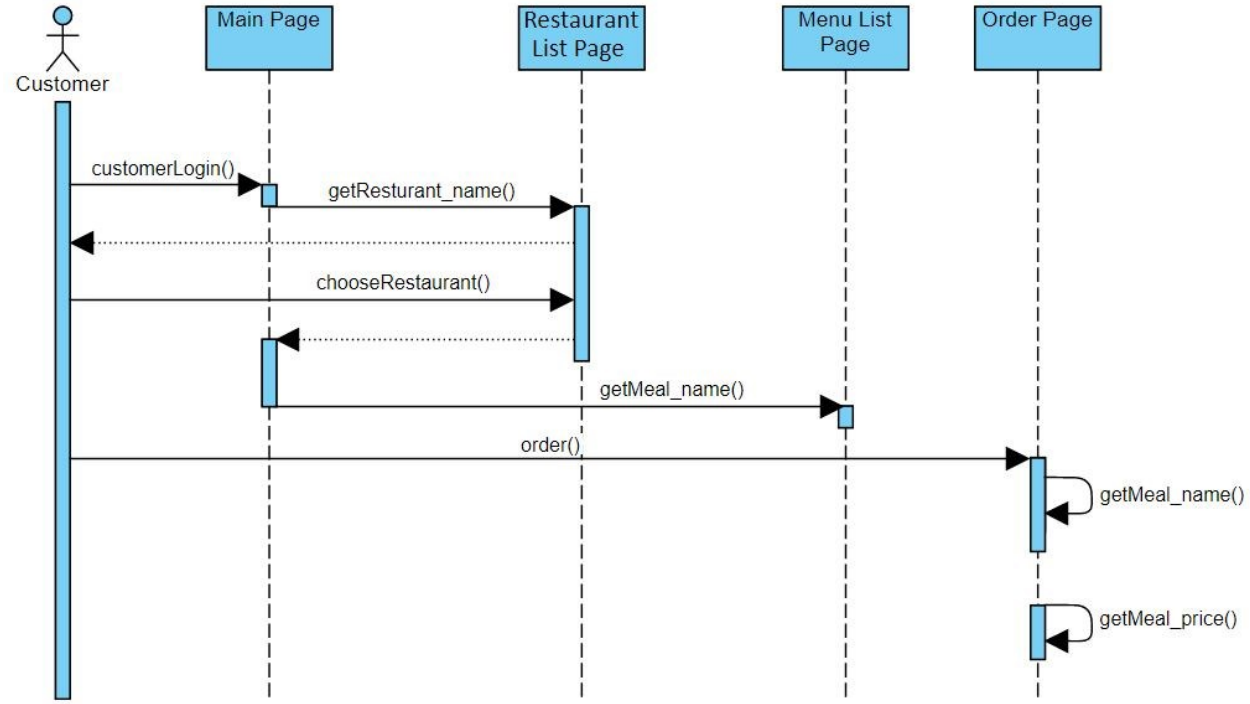


Figure 11

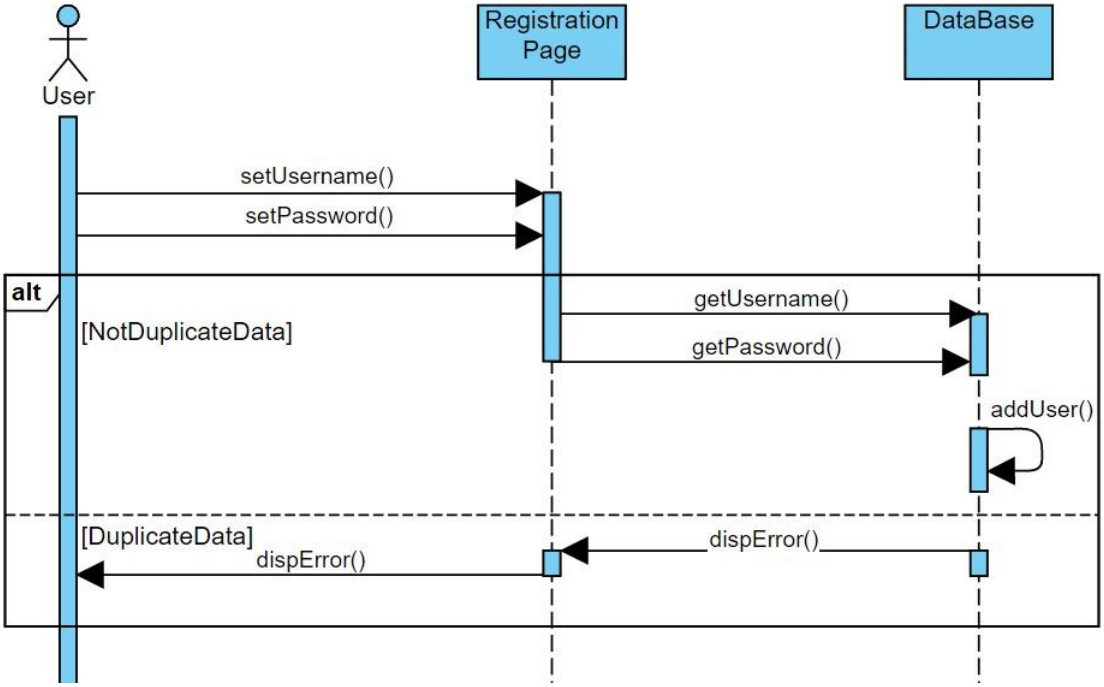


Figure 12

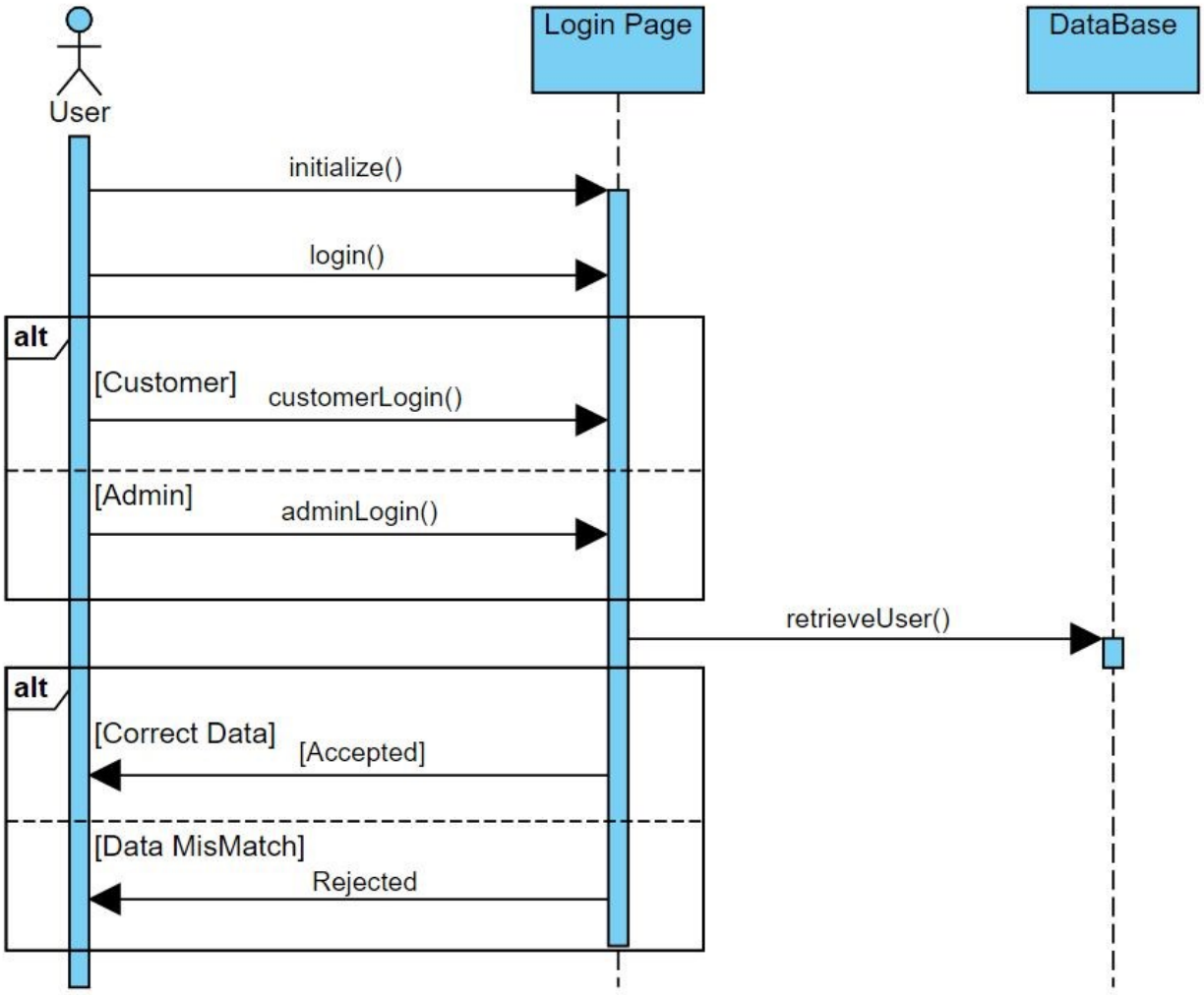

User
Login Page
DataBase
initialize()
login()
alt
[Customer]
customerLogin()
[Admin]
adminLogin()
retrieveUser()
alt
[Correct Data]
[Accepted]
[Data MisMatch]
Rejected

Figure 13

7. Noun Extraction and CRC Cards

Otlob is a meal order and delivery software. There are different types of users, with each having their own privileges. The user registers his username, password and phone no. A validation message is sent to users' cell nos. The customer is allowed to select and order meals. The restaurant owner is allowed to edit restaurant list. Only the db manager is allowed to verify an added restaurant. When an order is made, the restaurant is notified so that it checks for its availability. An invoice with the final order details is shown after order is confirmed.

Nouns:

Otlob, meal, order, delivery, software, users, privileges, username, password, phone, validation, message, customer, restaurant, owner, edit, list, db manager, availability, invoice, order details

Noun filtration:

Otlob, meal, order, users, username, password, validation, customer, restaurant, owner, db manager, invoice, order details

Otlob	
• Starts APP <<Main Class>> • Starts Landing Page	• GUI

User Admin, Customer	
• Make Order • Delete Order • Make Restaurant • Delete Restaurant • Order Meal • Cancel Order	• Connection

Resturant	
• Share List of Meals to Admin • Contain Resturant(ID, Name)	• Admin • Meal

Figure 14


Customer
User<<Parent>>
Make Order
Cancel Order
Login
Register
Contain(Customer Username, Order History)
Connection
Meal
Share Meal Price to Admin
Share Restaurant_ID to Admin
Restaurant
Admin
Admin
User<<Parent>>
Add Meal to Database
Delete Meal from Database
Add Restaurant to Database
Delete Restaurant from Database
User Login Validation
Registering Users
Connection
databaseManager
Get data from database
Connection
RegistrationController
Send User To CustomerLogin Page
Take Username/Password from User
Check Username/Password validity
Send User to home page
Admin
GUI

Figure 15

LandingPageController	
• Transfer User To Customer Login Page if he is Customer • Transfer User To Admin Login page if he is Admin • Transfer User to Registration Page if he is new user	• GUI

DeleteMealController	
• Send user to Landing Page • Delete Meal from Database	• GUI • Admin

DeleteRestaurantController	
• Check if Delete Restaurant button is pressed • Delete Restaurant from Database	• Admin • GUI

AddResturantController	
• Add Added Restaurant attributes to Database from GUI • Set Window to <<Landing Page>>	• Admin • GUI

InvoiceController	
• Show User invoice details • Add invoice to Customer order history • Add Invoice to Database	• Admin • Customer • GUI

CustomerLoginController	
• Check If customer Username/Password Correct • Change app Window <<Make Order Window>>	• Customer • GUI

Figure 16

AdminLoginController	
• Check Admin Username/Password entered validity • Show user if the Username/Password entered are valid • send user to AdminDashboard	• Admin • GUI

AdminDashboardController	
• invokes Add_Restaurant function if Add Restaurant button is pressed in Admin Dashboard • invokes Delete_Restaurant function if Delete Restaurant button is pressed in Admin Dashboard • invokes ad • invokes Add_Meal function if Add Meal button is pressed in Dashboard • invokes Delete_Meal function if Delete Meal button is pressed in Dashboard	

AddMealController	
• Add Added Meal Details to Database through Admin Class • Add Added Meal quantity from GUI to Database through Admin Class	• Admin • GUI

MakeOrderController	
• Show to user Meal Price • Check User credit card details • Show user credit card details before checkout • Take from user Address • Show to user Total Price • Ask user for Meal quantity • Check if user payment is valid	• Admin

Connection	
• Connect user to Databasemanager • Share Database to user • Share Transactions to Users • Users Share Transactions to Databasemanager • Connect databaseManager to SQL	• databaseManager • User

Figure 17

GUI	
• show graphical interface to user	• AdminDashboardController • LandingPageController • InvoiceController • RegistrationController • AdminLoginController • CustomerLoginController • MakeOrderController • AddMealController • DeleteMealController • AddResturantController • DeleteResturantController

GUI	
• show graphical interface to user	• AdminDashboardController • Admin • LandingPageController • InvoiceController • RegistrationController • AdminLoginController • CustomerLoginController • MakeOrderController • AddMealController • DeleteMealController • AddResturantController • DeleteResturantController

Figure 18

8. Class Diagram

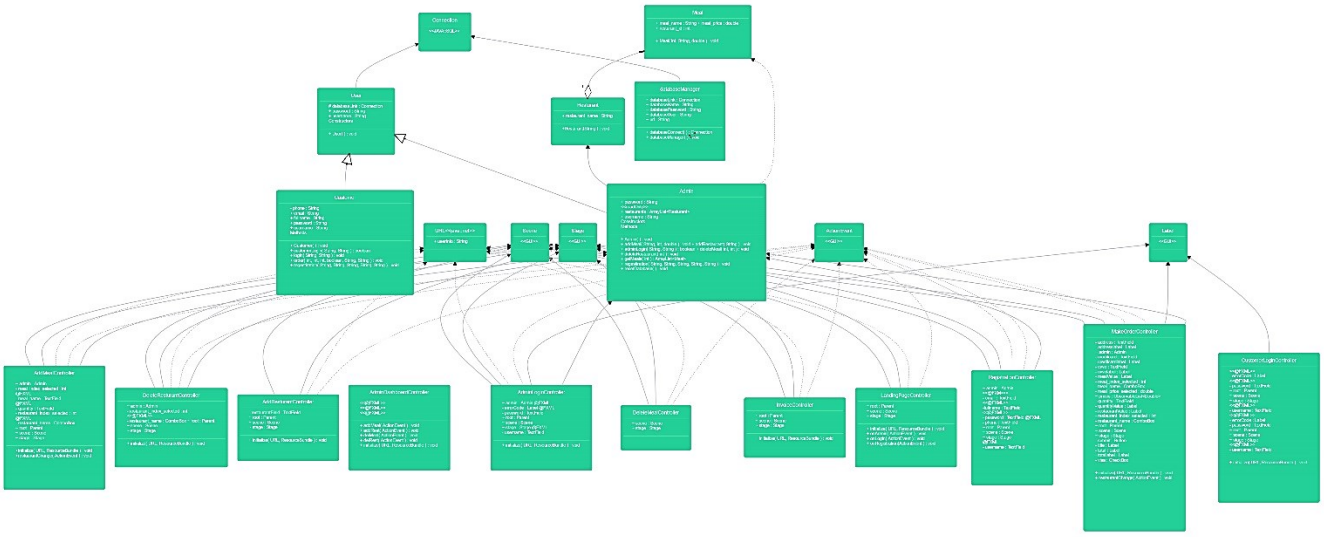


Figure 19

The class diagrams shown here are non but summaries of details attached in rest of the deliverable as well as in section 11

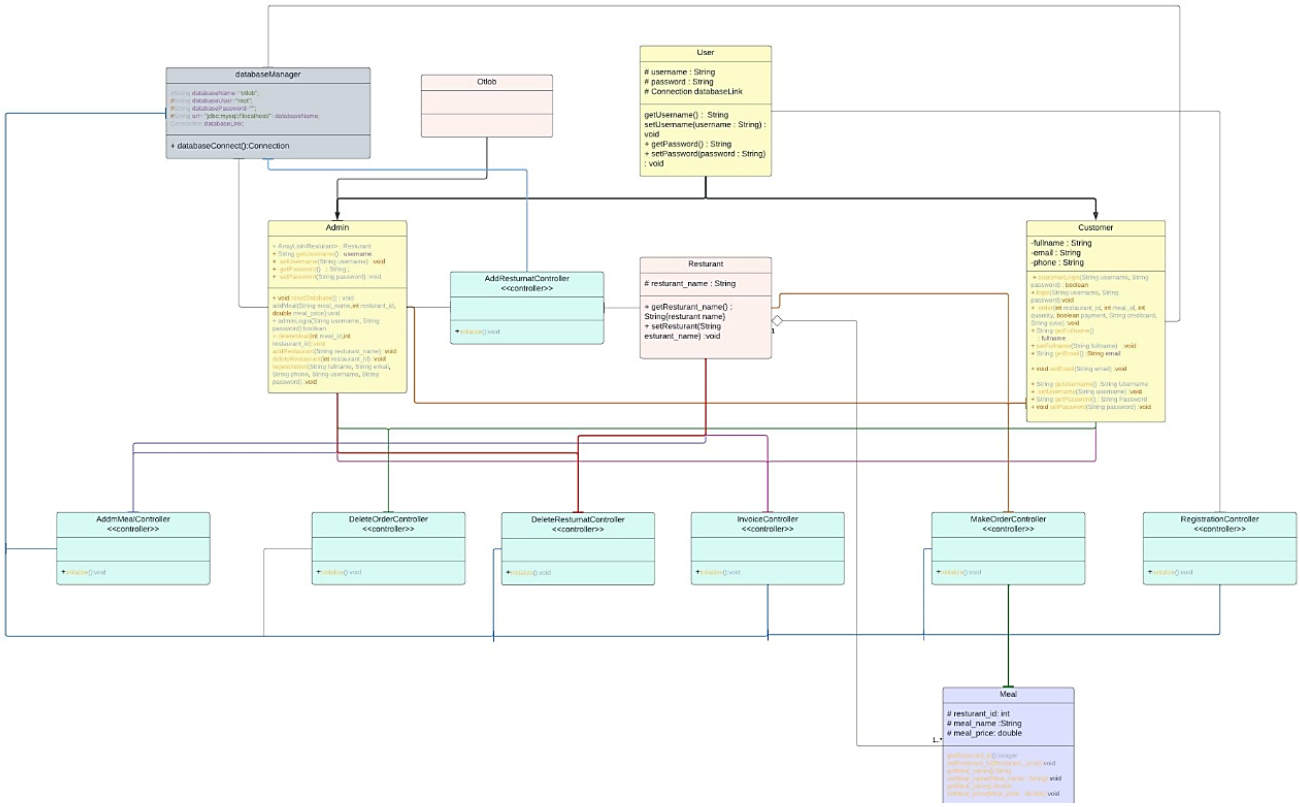


Figure 20

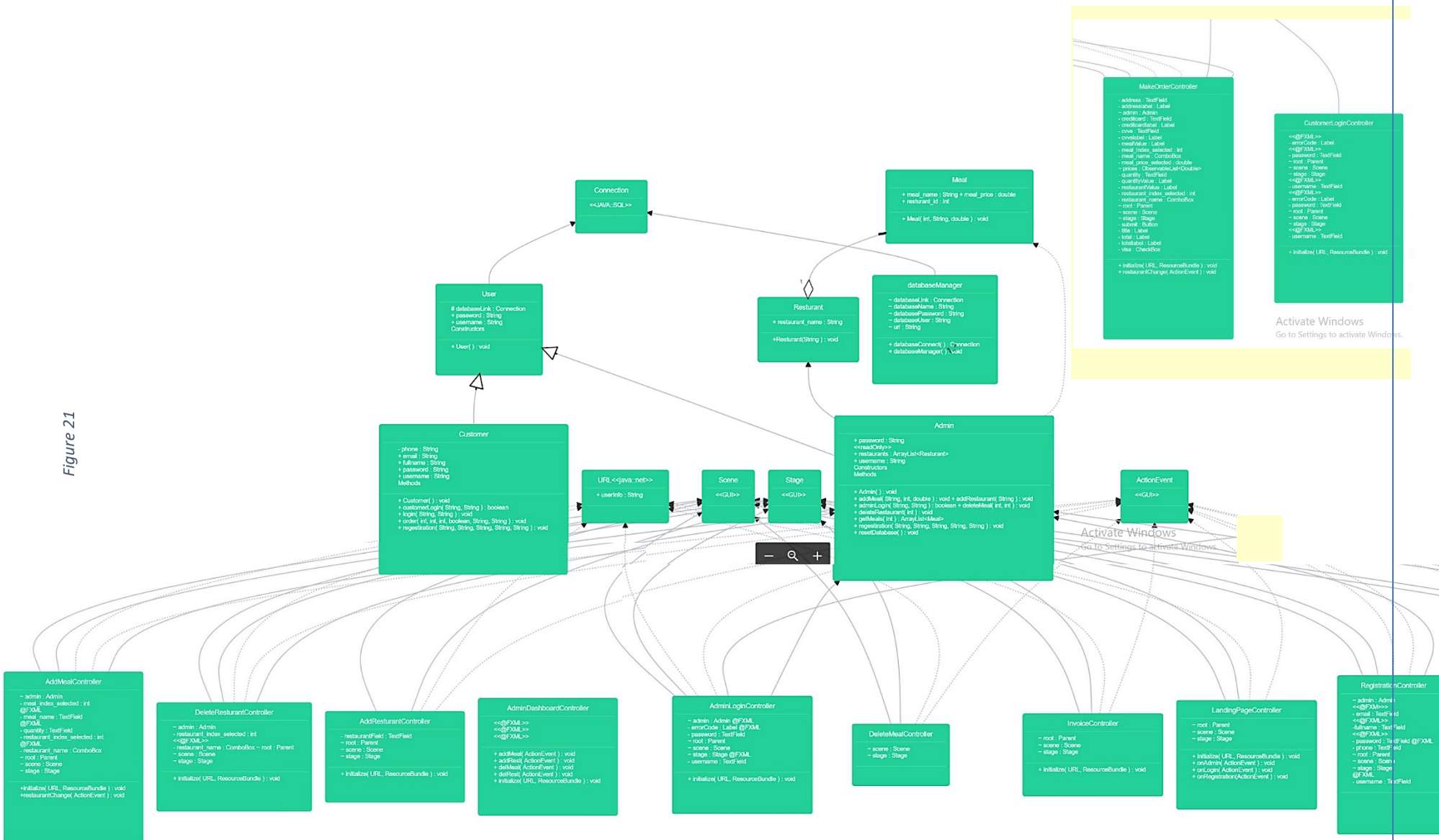

Connection
<<JAVA::SQL>>
Meal
+ meal_name : String + meal_price : double
+ resturant_id : int
+ Meal(int, String, double) : void
User
databaseLink : Connection
+ password : String
+ username : String
Constructors
+ User( ) : void
Resturant
+ restaurant_name : String
+Resturant(String) : void
databaseManager
~ databaseLink : Connection
~ databaseName : String
~ databasePassword : String
~ databaseUser : String
~ url : String
+ databaseConnect( ) : Connection
+ databaseManager( ) : void
Customer
- phone : String
+ email : String
+ fullname : String
+ password : String
+ username : String
Methods
+ Customer( ) : void
+ customerLogin(String, String) : boolean
+ login(String, String) : void
+ order(int, int, int, boolean, String, String) : void
+ registration(String, String, String, String, String) : void
URL<<java::net>>
+ userInfo : String
Scene
<<GUI>>
Stage
<<GUI>>
Admin
+ password : String
<<readOnly>>
+ restaurants : ArrayList<Resturant>
+ username : String
Constructors
Methods
+ Admin( ) : void
+ addMeal(String, int, double) : void + addRestaurant(String) : void
+ adminLogin(String, String) : boolean + deleteMeal(int, int) : void
+ deleteRestaurant(int) : void
+ getMeals(int) : ArrayList<Meal>
+ registration(String, String, String, String, String) : void
+ resetDatabase( ) : void
ActionEvent
<<GUI>>
MakeOrderController
- address : TextField
- addressLabel : Label
- admin : Admin
- creditcard : TextField
- creditcardlabel : Label
- cvva : TextField
- cvvlabel : Label
- mealValue : Label
- meal_index_selected : int
- meal_name : ComboBox
- meal_price_selected : double
- prices : ObservableList<Double>
- quantity : TextField
- quantityValue : Label
- restaurantValue : Label
- restaurant_index_selected : int
- restaurant_name : ComboBox
- root : Parent
- scene : Scene
- stage : Stage
- submit : Button
- title : Label
- total : Label
- totalLabel : Label
- visa : CheckBox
+ initialize(URL, ResourceBundle) : void
+ restaurantChange(ActionEvent) : void
CustomerLoginController
<<@FXML>>
- errorCode : Label
<<@FXML>>
- password : TextField
- root : Parent
- scene : Scene
- stage : Stage
<<@FXML>>
- username : TextField
<<@FXML>>
- errorCode : Label
- password : TextField
- root : Parent
- scene : Scene
- stage : Stage
<<@FXML>>
- username : TextField
+ initialize(URL, ResourceBundle) : void
AddMealController
~ admin : Admin
- meal_index_selected : int
@FXML
- meal_name : TextField
@FXML
- quantity : TextField
- restaurant_index_selected : int
@FXML
- restaurant_name : ComboBox
~ root : Parent
~ scene : Scene
~ stage : Stage
+initialize(URL, ResourceBundle) : void
+restaurantChange(ActionEvent) : void
DeleteResturantController
~ admin : Admin
- restaurant_index_selected : int
<<@FXML>>
- restaurant_name : ComboBox ~ root : Parent
~ scene : Scene
~ stage : Stage
+ initialize(URL, ResourceBundle) : void
AddResturantController
- restaurantField : TextField
~ root : Parent
~ scene : Scene
~ stage : Stage
+ initialize(URL, ResourceBundle) : void
AdminDashboardController
<<@FXML>>
<<@FXML>>
<<@FXML>>
+ addMeal(ActionEvent) : void
+ addRest(ActionEvent) : void
+ delMeal(ActionEvent) : void
+ delRest(ActionEvent) : void
+ initialize(URL, ResourceBundle) : void
AdminLoginController
~ admin : Admin @FXML
- errorCode : Label @FXML
- password : TextField
~ root : Parent
~ scene : Scene
~ stage : Stage @FXML
- username : TextField
+ initialize(URL, ResourceBundle) : void
DeleteMealController
~ scene : Scene
~ stage : Stage
InvoiceController
~ root : Parent
~ scene : Scene
~ stage : Stage
+ initialize(URL, ResourceBundle) : void
LandingPageController
~ root : Parent
~ scene : Scene
~ stage : Stage
+ initialize(URL, ResourceBundle) : void
+ onAdmin(ActionEvent) : void
+ onLogin(ActionEvent) : void
+ onRegistration(ActionEvent) : void
RegistrationController
~ admin : Admin
<<@FXML>>
- email : TextField
<<@FXML>>
-fullname : TextField
<<@FXML>>
- password : TextField @FXML
- phone : TextField
~ root : Parent
~ scene : Scene
~ stage : Stage
@FXML
- username : TextField
Activate Windows
Go to Settings to activate Windows.
Activate Windows
Go to Settings to activate Windows.

Figure 21

9. State Diagram

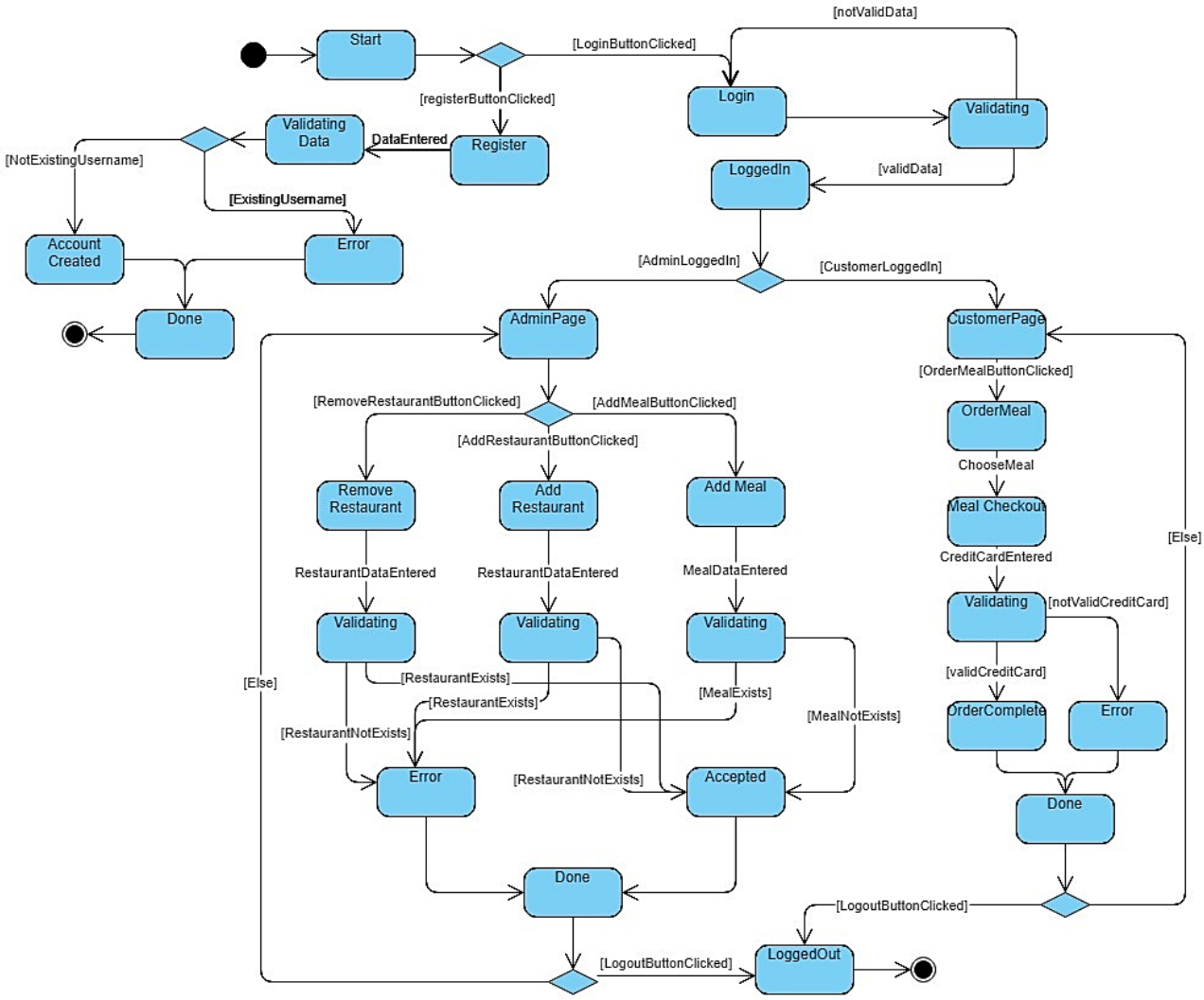


Figure 22

10. Client-Object Relation Diagram

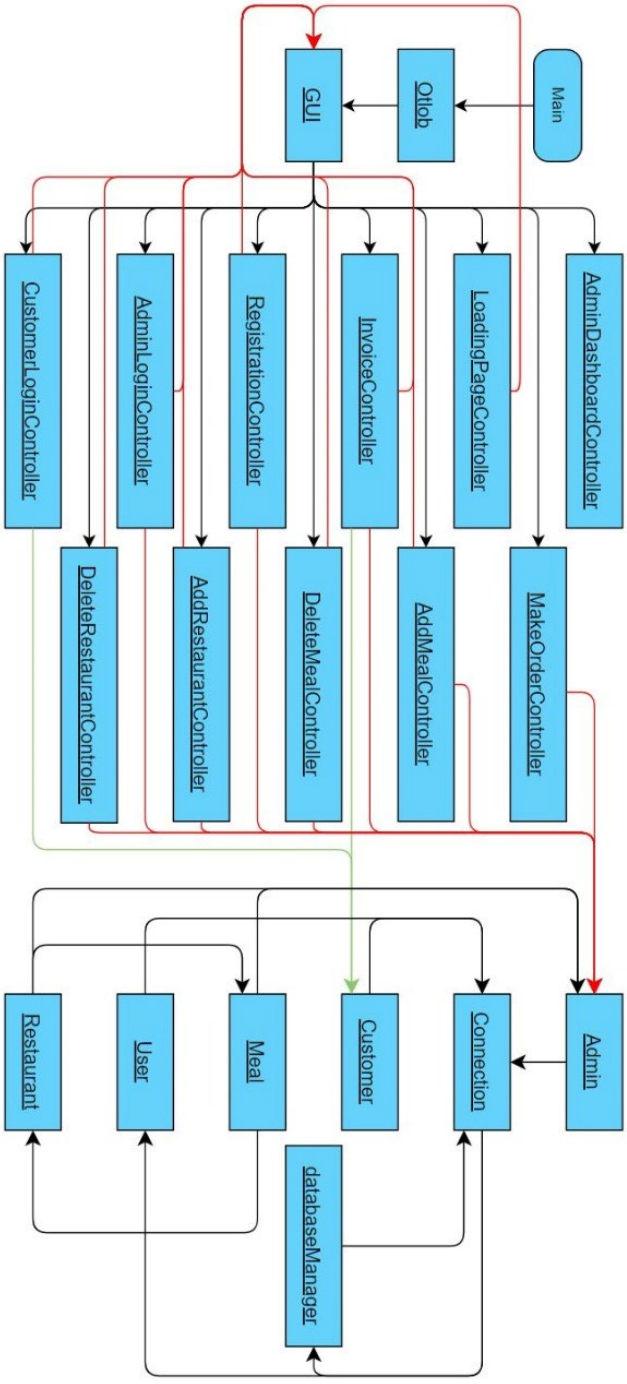


Figure 23

11. OOAD Methodologies

The Rumbaugh Methodology describes an approach for the development of scalable and usable software with easily expressible class properties without sacrificing the software reliability. The reasons for this usage are the simplified simulation of software entities and the reduction of development complexity. The stages of OMT, as otherwise named, include analysis, system design, object design and implementation.

Otlob features a relatively small-scale software and a human-managed database. This also means that easy communication between different stakeholders should be normalised and the software process should not get too overboard. The functioning of Otlob necessitates many details and has the capacity for a variety of test cases.

The Jacobson Methodology, OOSE, additionally uses more detailed requirements, use cases, testing models and PDLs. Its best feature is advancing software maintainability and adaptability through software robustness testing and quality assurance.

11.1. Rumbaugh et Al. Methodology

11.1.1. Analysis

The goal here is to provide a trustable software that eases ordering and delivery food. The analysis stage is further built on the problem statement to object, dynamic and functional models.

```
Problem Statement:

Provide a trustable software that eases ordering and delivery food
```

11.1.1.1. Object Model

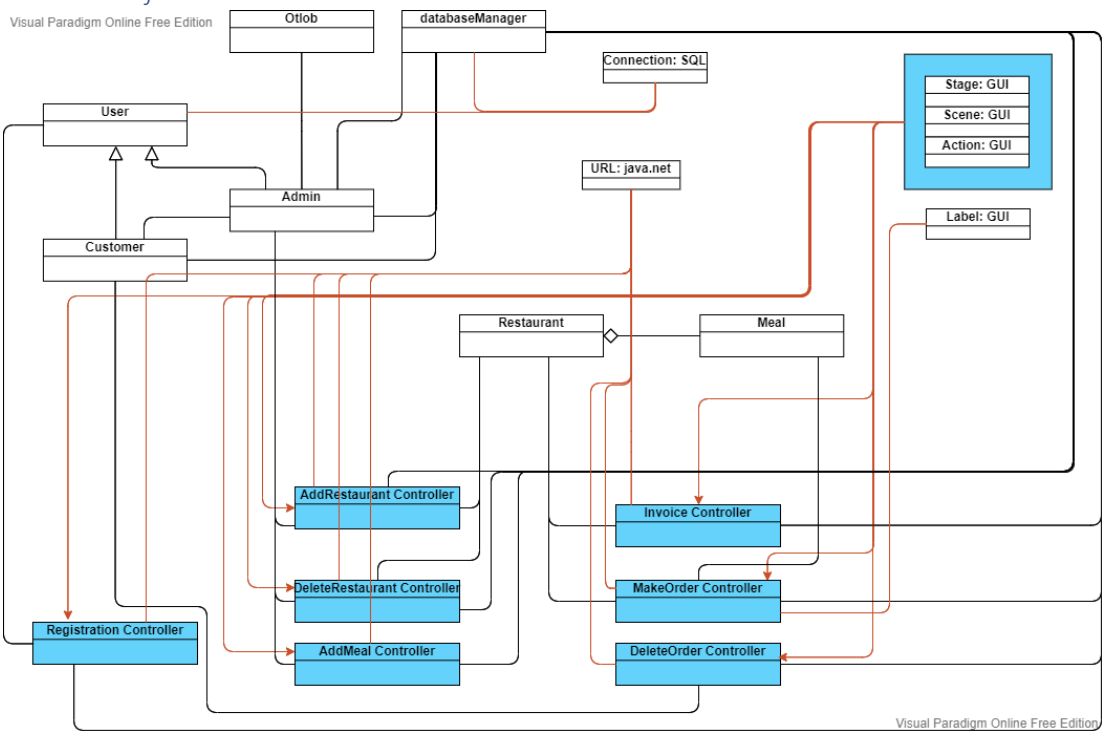


Figure 24

11.1.1.2. Dynamic Model

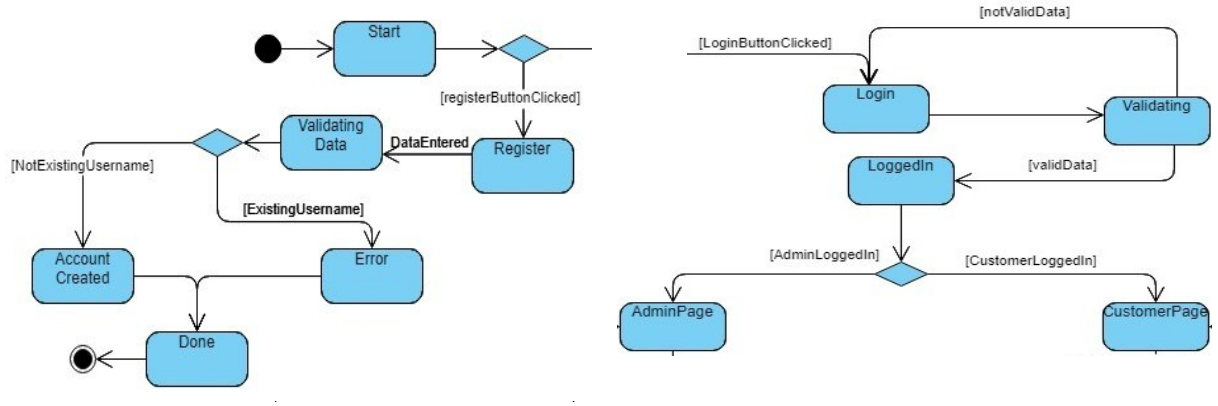


Figure 25

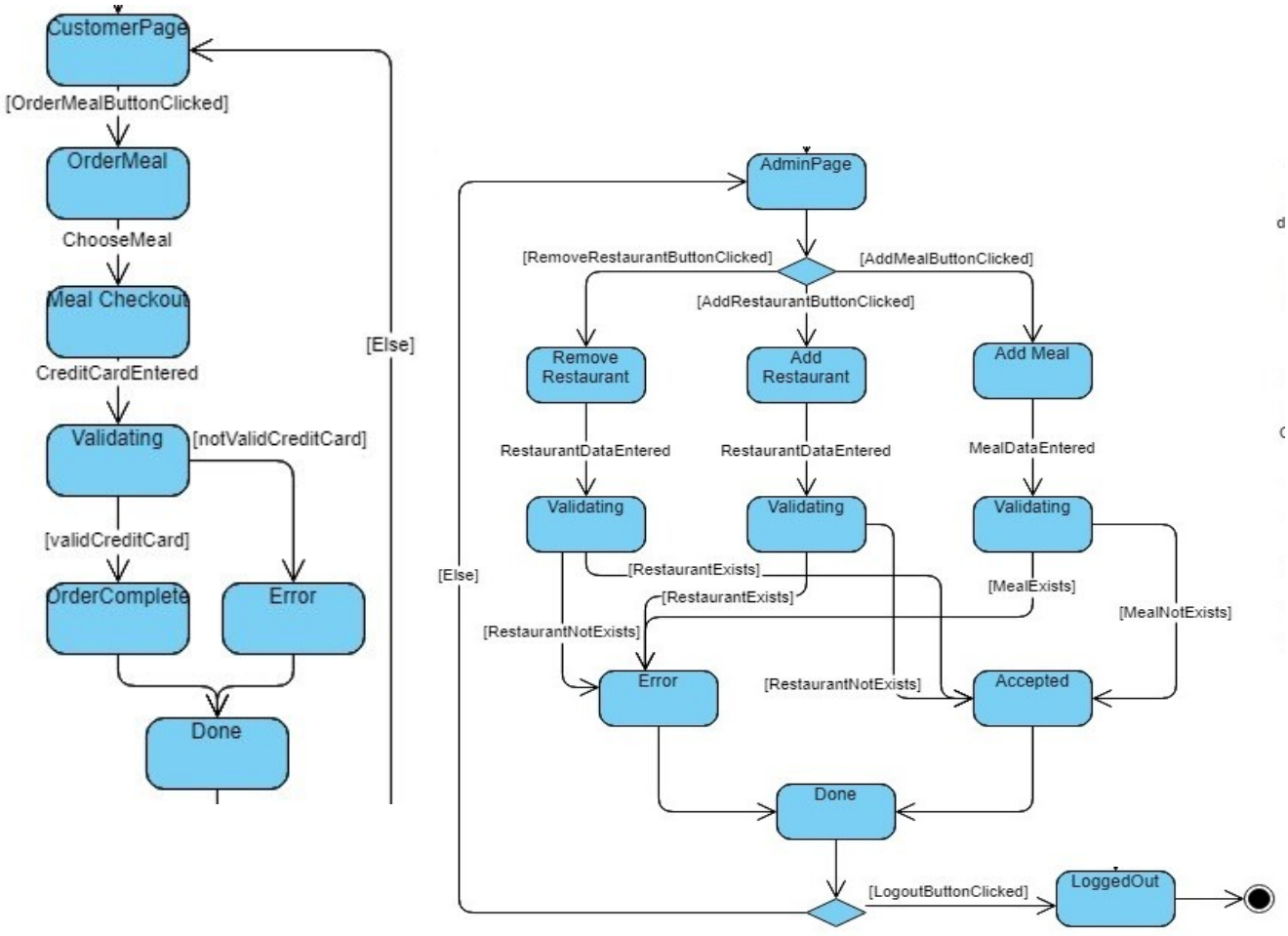


Figure 26

11.1.1.3. Functional Diagram

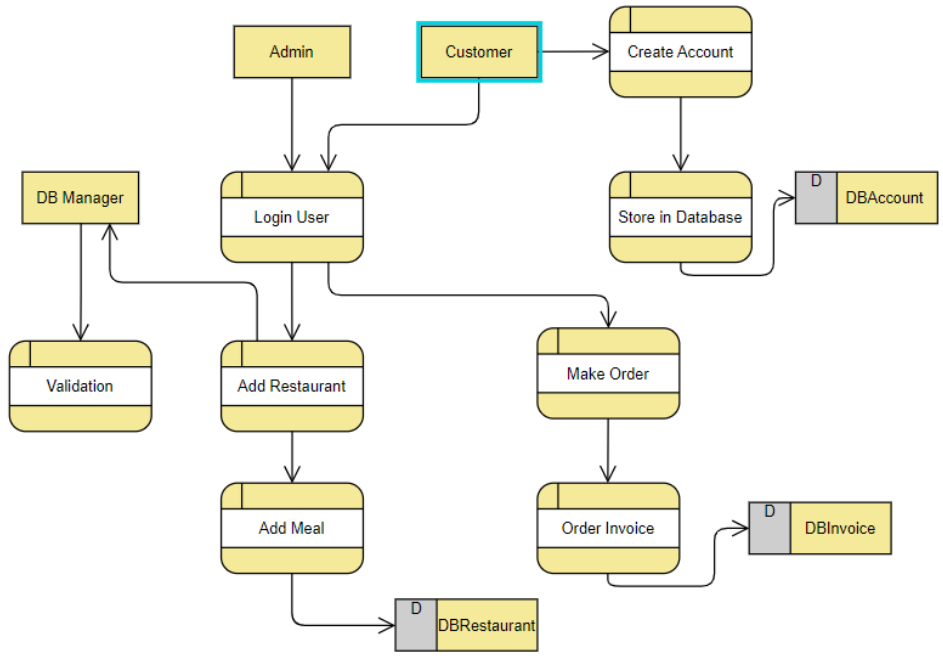


Figure 27

11.1.2. System Design

The following packages and subsystems comprise of the system design (the GUI wasn't added for simplicity):

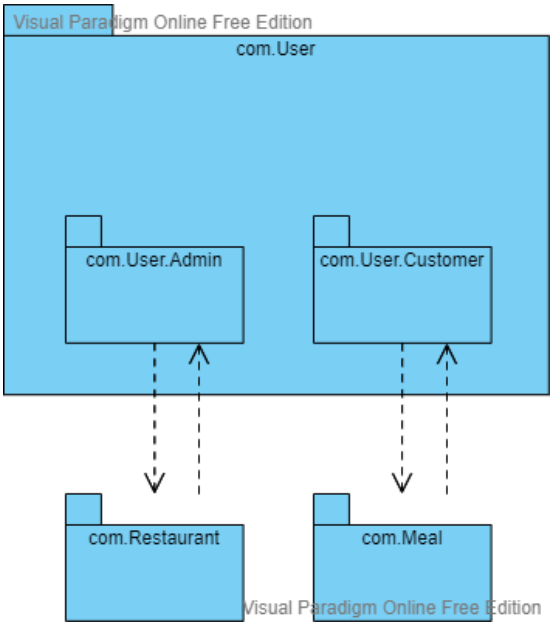


Figure 28

11.1.3. Object Design

The class, operation and attribute details shown here are those from the class diagram:

Figure 29

11.1.4. Implementation

The software was implemented using Java programming language. The details for the implementation are mentioned in the user guide and in the code deliverable.

11.2. Jacobson et Al Methodology

11.2.1. Detailed Use Cases

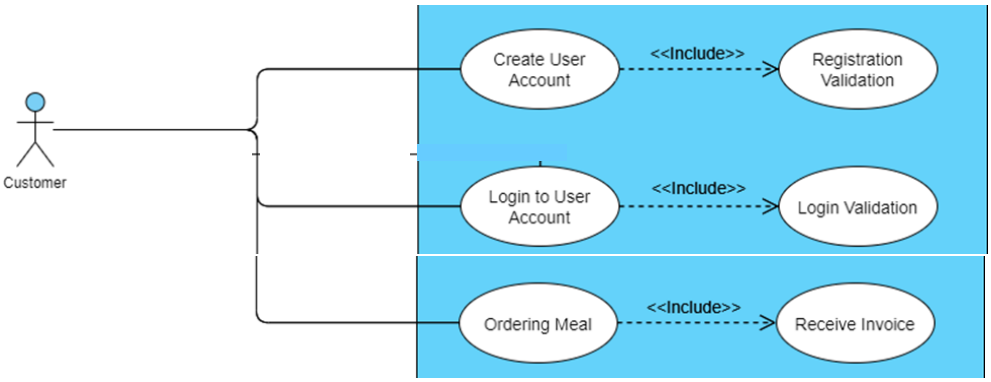


Figure 30

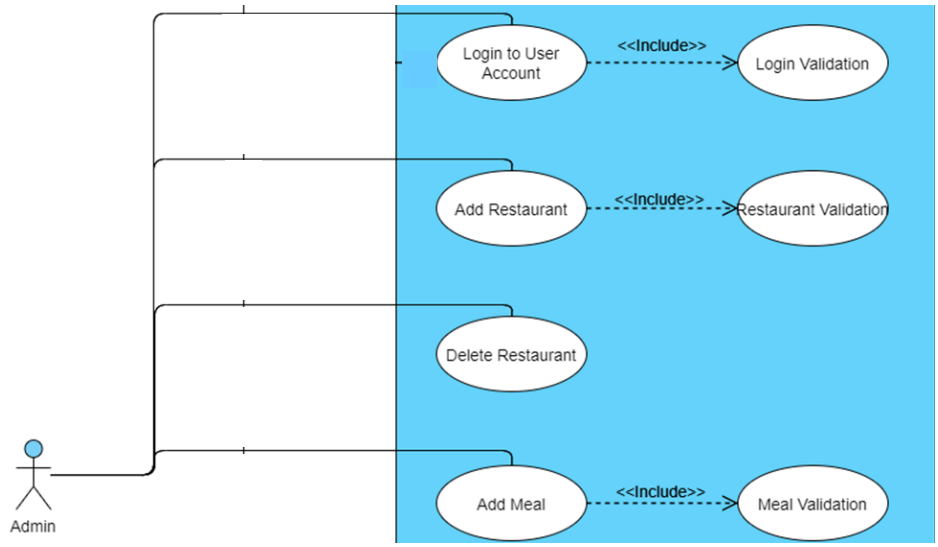


Figure 31

11.2.2. Interaction and Collaboration Diagrams

The full interaction diagrams have already been discussed; here we discuss the collaboration diagram.

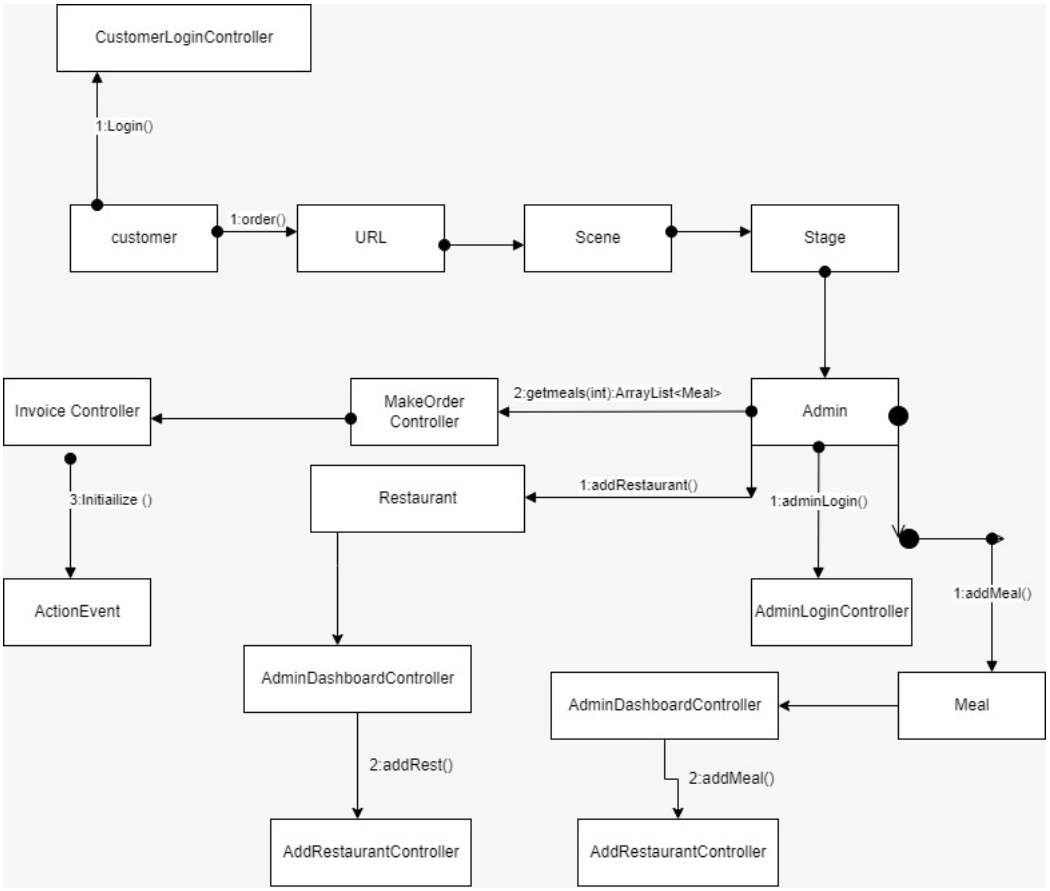


Figure 32

12. Comparative Analysis of the Output of the Adopted Methodologies

	Rumbaugh Methodology	Jacobson Methodology
Output	Detailed output in case of diagrams , analysis and every aspect of software development lifecycle	Minimises the bulk of coding into simpler diagrams
Advantages	Primary strength in object analysis, which is beneficial for large scale problems that require many diagnostics	Easy method, not many diagrams and is pluggable on informal software development lifecycles
Disadvantages	Doesn't show communication between components well	Steep curve since complexity between one step and another may not be suitable for some(detailed class diagrams and pseudocode for example)

Table 6

13. Architectural Model

The MVC or Model-View-Controller is the architectural model used in this project The Model View Controller uses a certain pattern that divides the application into 3 components: model, view and controller

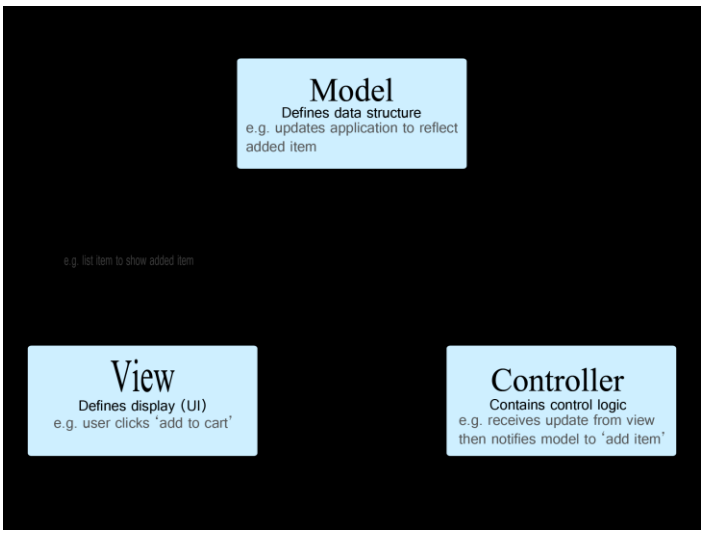


Figure 33

In this project[5], 11 controllers have been used for several purposes such as : Customer logging in, adding and deleting both restaurants and meals by the admin.

13.1. The Model

The Model is used for any data-related logic that the user can actually use. This can be any kind of data such as data transferred between Controller and View components. For example, a Customer object will

[5] See attached Class diagram

retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

13.2. View

The View component is heavily related to all UI related objects, for instance, the admin View would include text boxes, dropdowns, text fields etc.

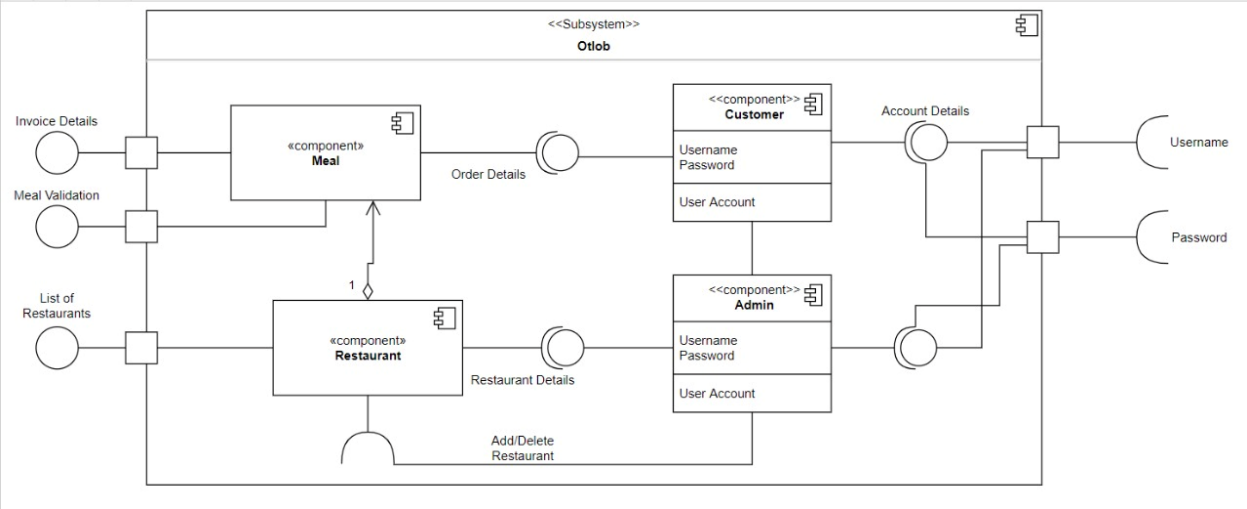
13.3. Controller

Controllers act as a bridge between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final result. For instance, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to display the Customer info.

13.4. Justification for using MVC

The MVC does have many advantages to be selected as the model for our project. First of all, The MVC separates the control(controller), data (the model) and the GUI (view), the result is that the application is easier to be maintained and also easier to detect bugs, for example: if there is a bug in the Combo boxes, this means that the bug is most probably linked to the View component. In addition to this, the fact that the MVC separates the 3 components makes any potential modifications much more easier than combining the 3 components together because any modification in any component will not interfere with other components. Also, The MVC pattern is generally easy to be tested.

14. Component Diagram



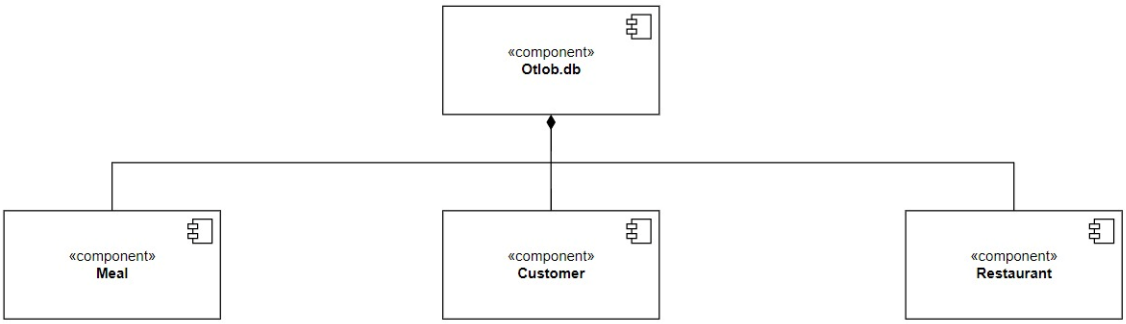


Figure 34

15. Time Plan

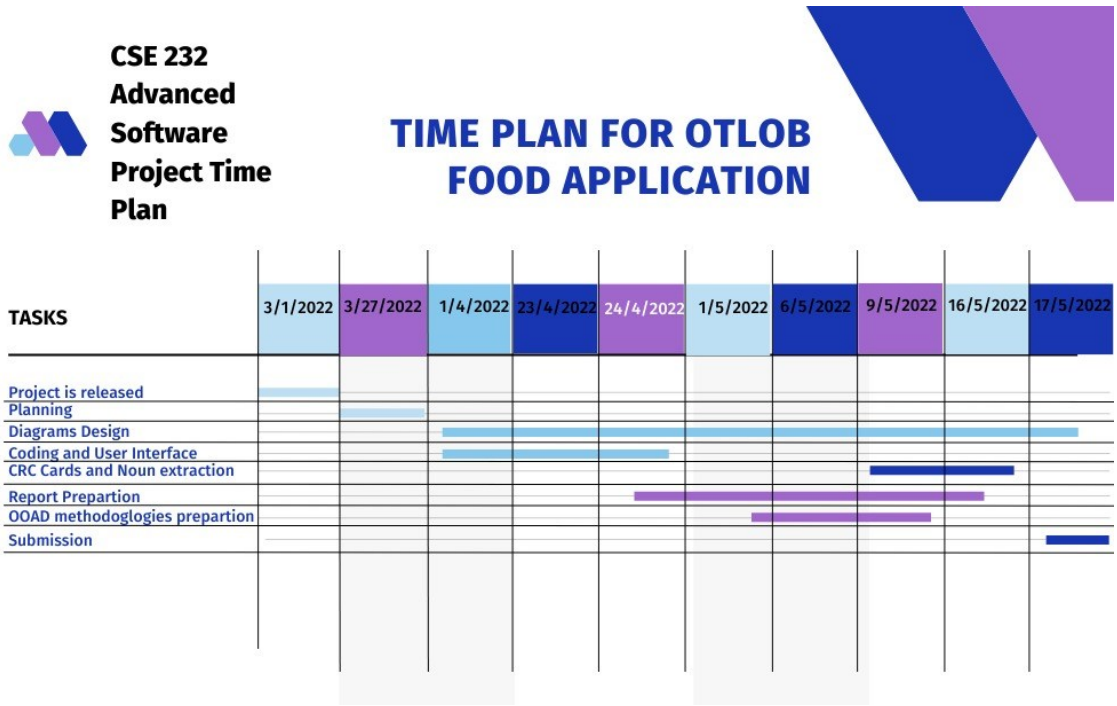


Figure 35

16. User Guide

16.1. Admin's guide

1. Make sure to start xampp, start netbeans ide 8.2, and import mysql-connector-java-8.0.26.jar before starting to avoid program crashing

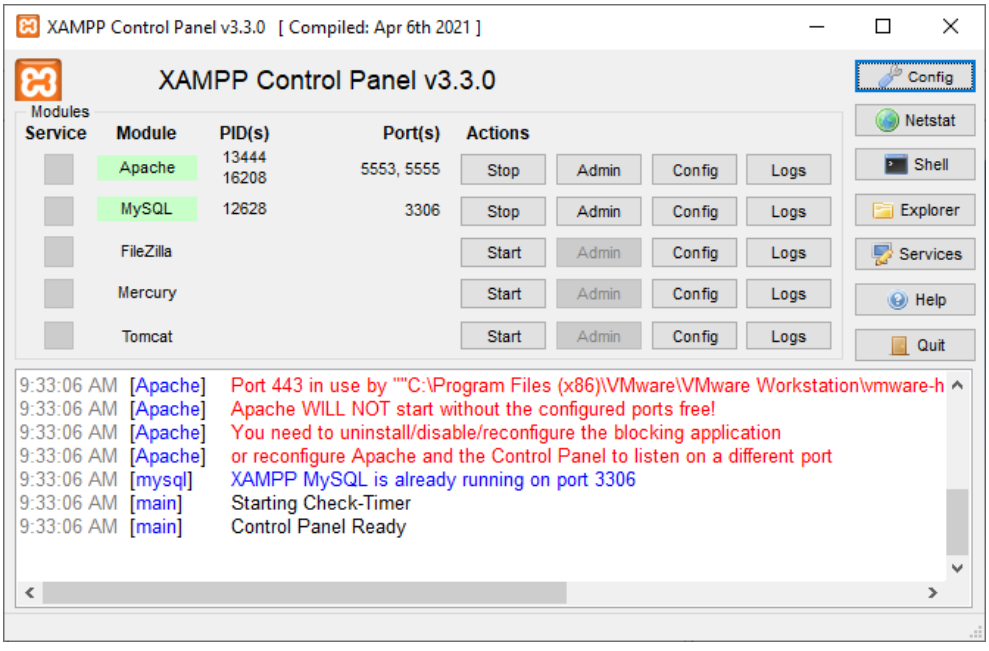
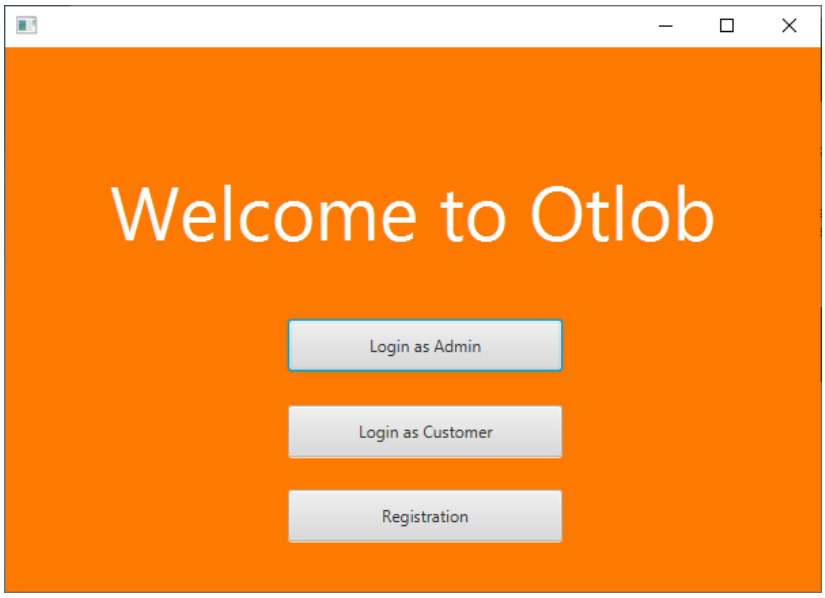


Figure 36

2. The program will start



3. You can click on Login as Admin to login as admin

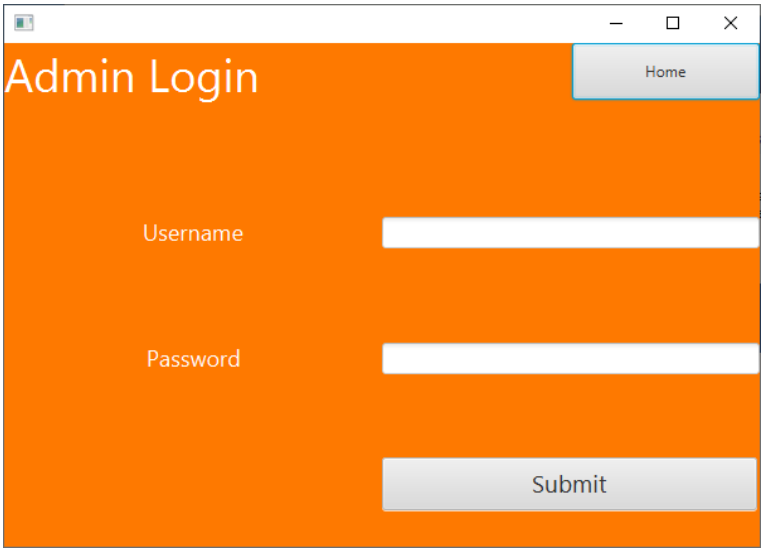
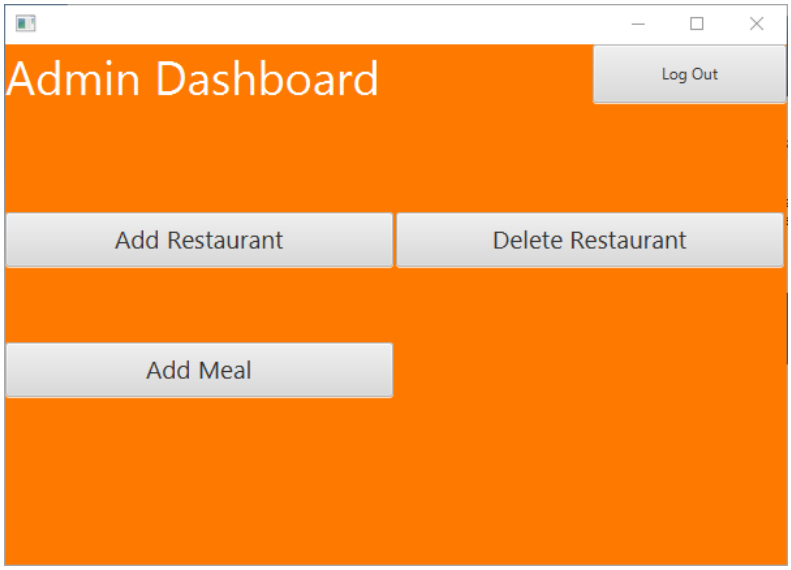


Figure 37

4. login as admin using the default credentials admin:admin and you will get into admin dashboard



5. you can add restaurant by clicking on add restaurant and fill data

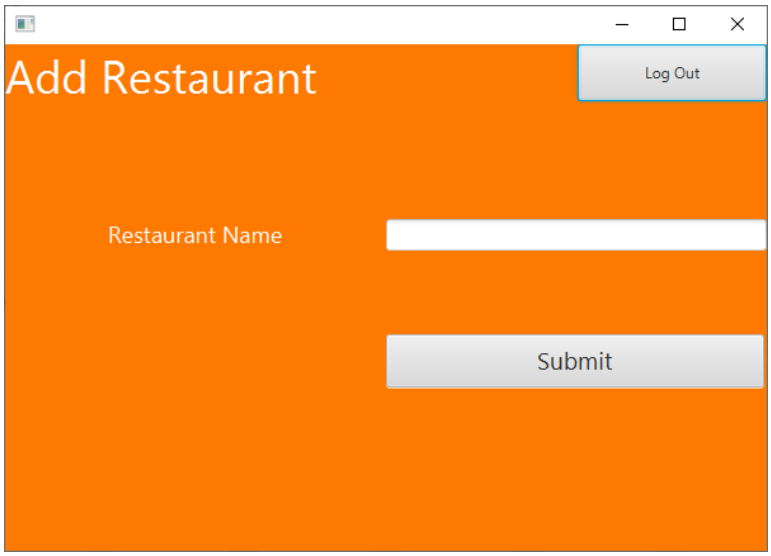
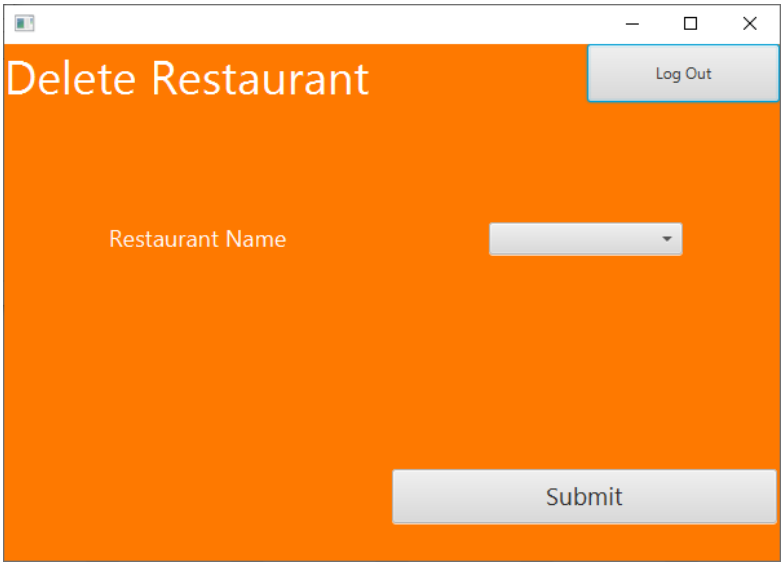


Figure 38

6. you can delete restaurant by clicking on delete restaurant and fill data



7. you can add meal by clicking on add meal and fill data

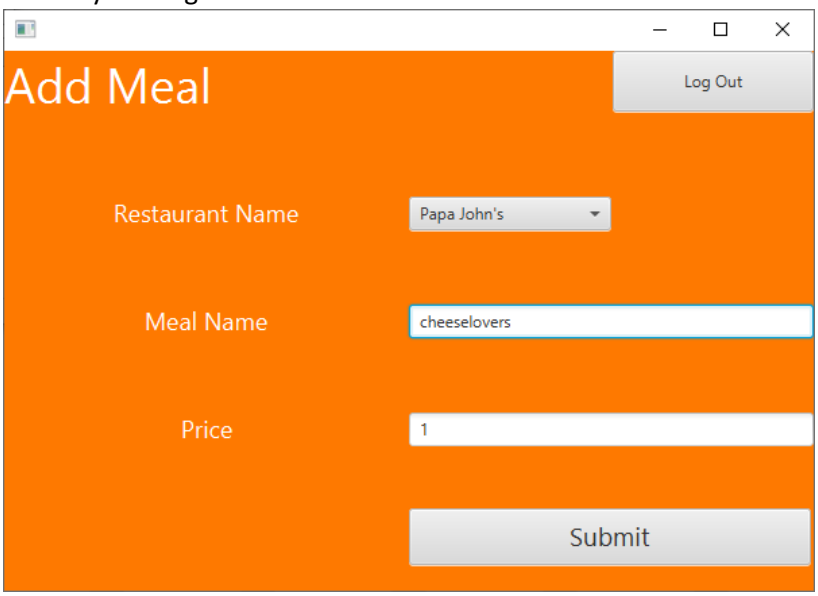
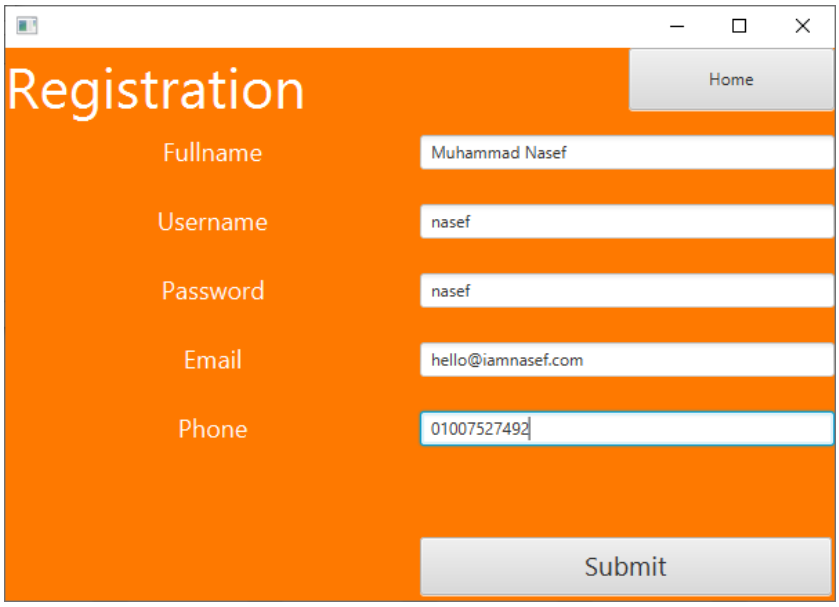


Figure 39

User Guide

1. Click on registration and fill data to create account



2. You will be redirected to login page where you can fill your credentials

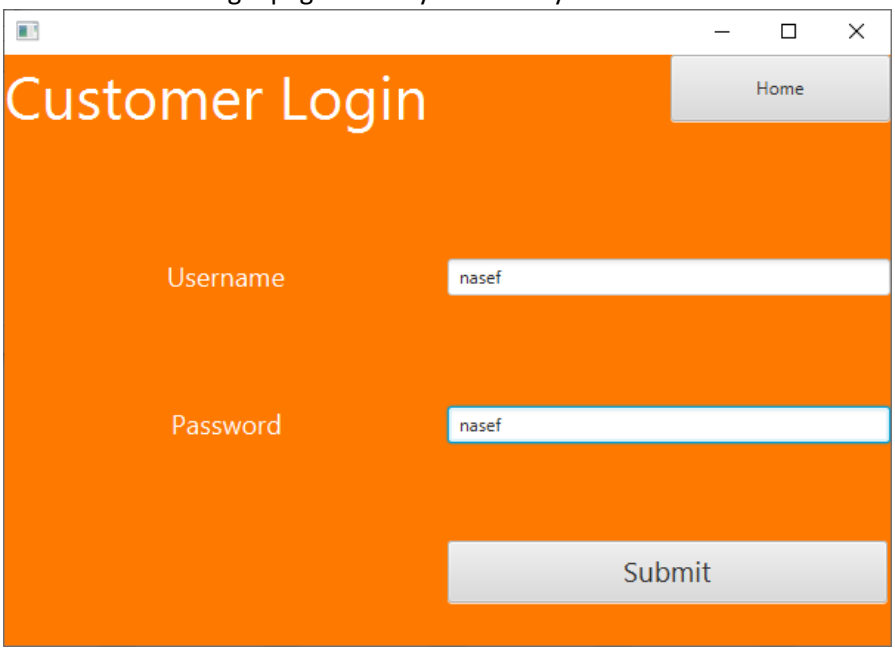
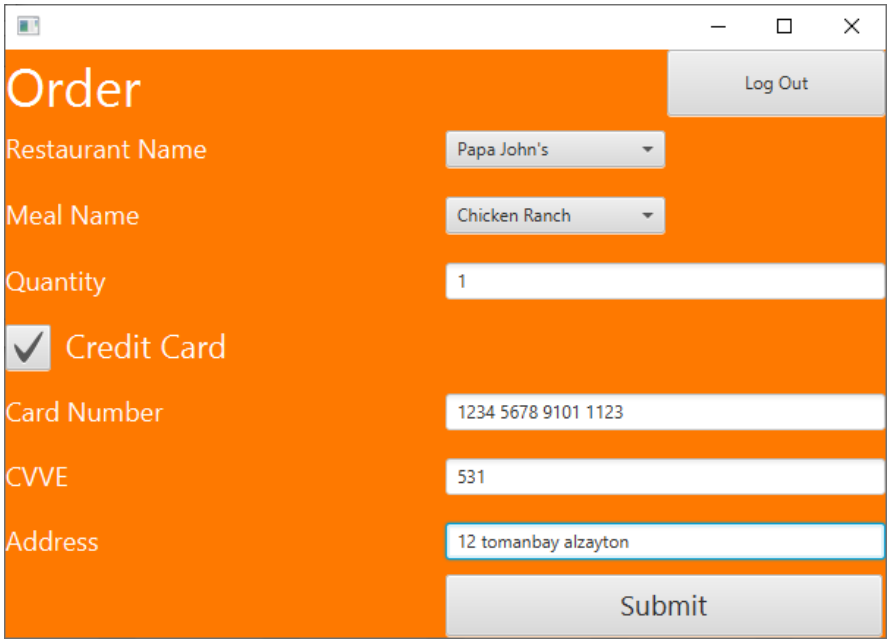


Figure 40

3. You will be redirected to order page where you will fill your order details



4. After submitting you will get the invoice which contains the total order

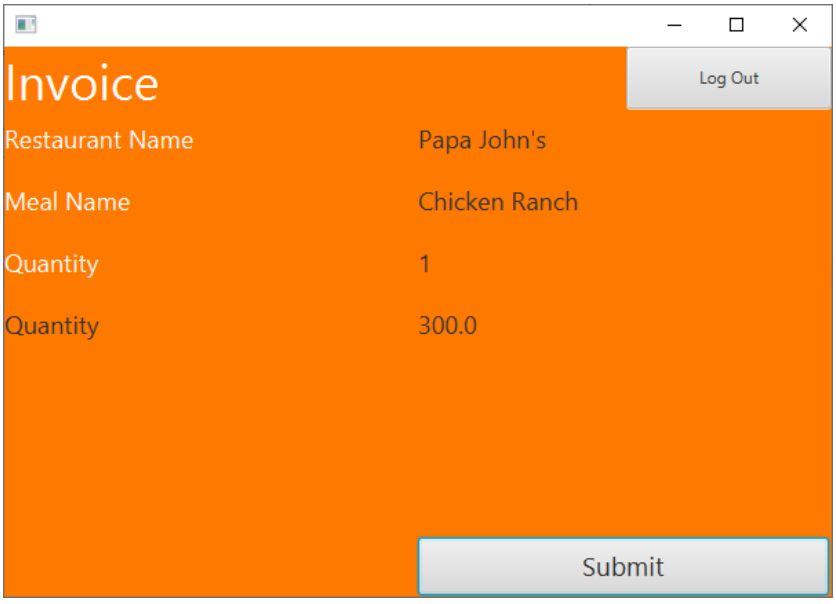


Figure 41

17. Appendices

17.1. Appendix A

17.1.1. Business study

By the end of year 2022, it's expected that the online delivery market to be worth 151 USD billion [2]. It's the online delivery only, not to mention the fast-food industry as a whole. And during covid19 13% of the us restaurant market was taken up by online delivery [3]. Combining with how fast the Egyptian market is growing as 21% of Egyptian consumers use restaurants and meal delivery [4]. This why we decided to build an online ordering system to exploit the market gap and the market demand in this area. Our application will have the following features

17.2. Appendix B

17.2.1. Default generated SQL code

```
-- phpMyAdmin SQL Dump
-- version 5.1.1
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Generation Time: Mar 18, 2022 at 03:04 PM
-- Server version: 10.4.21-MariaDB
-- PHP Version: 7.3.30

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `otlob`
--

-- --------------------------------------------------------

--
-- Table structure for table `meals`
```

```sql
--

CREATE TABLE `meals` (
  `meal_id` int(11) NOT NULL,
  `resturant_id` int(11) NOT NULL,
  `meal_name` varchar(255) NOT NULL,
  `meal_price` double NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `meals`
--

INSERT INTO `meals` (`meal_id`, `resturant_id`, `meal_name`, `meal_price`) VALUES
(1, 1, 'Family Box', 300),
(2, 1, 'Chicken Wrap', 50),
(3, 2, 'Big Mac', 60),
(4, 2, 'McFurry', 30),
(5, 3, 'Chicken Ranch', 300),
(6, 3, 'Peperoni', 320),
(7, 5, 'chicken', 500);

-- --------------------------------------------------------

--
-- Table structure for table `restaurants`
--

CREATE TABLE `restaurants` (
  `restaurant_id` int(11) NOT NULL,
  `restaurant_name` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `restaurants`
--

INSERT INTO `restaurants` (`restaurant_id`, `restaurant_name`) VALUES
(1, 'KFC'),
(2, 'McDonald\'s'),
(3, 'Papa John\'s'),
(4, 'asu'),
(5, 'yaw');

-- --------------------------------------------------------
```

```sql
--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `user_fullname` varchar(255) NOT NULL,
  `user_username` varchar(255) NOT NULL,
  `user_password` varchar(255) NOT NULL,
  `user_email` varchar(255) NOT NULL,
  `user_phone` varchar(255) NOT NULL,
  `user_id` int(11) NOT NULL,
  `user_role` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`user_fullname`, `user_username`, `user_password`,
`user_email`, `user_phone`, `user_id`, `user_role`) VALUES
('test1', 'test1', 'test1', 'test1@gmail.com', '01005365363', 1, 1),
('test2', 'test2', 'test2', 'test2@gmail.com', '0352352352', 2, 1),
('admin', 'admin', 'admin', 'admin@gmail.com', '01535352', 3, 0),
('nasef', 'nasef', 'nasef', 'nasef@nasef.nasef', '0100463463253', 4, 1);

--
-- Indexes for dumped tables
--


--
-- Indexes for table `meals`
--
ALTER TABLE `meals`
  ADD PRIMARY KEY (`meal_id`);

--
-- Indexes for table `restaurants`
--
ALTER TABLE `restaurants`
  ADD PRIMARY KEY (`restaurant_id`);

--
-- Indexes for table `users`
--
```

```
ALTER TABLE `users`
  ADD PRIMARY KEY (`user_id`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `meals`
--
ALTER TABLE `meals`
  MODIFY `meal_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;

--
-- AUTO_INCREMENT for table `restaurants`
--
ALTER TABLE `restaurants`
  MODIFY `restaurant_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `user_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

18. Research report

Object oriented analysis and design methodologies have been created from the beginning of software development to try and modulate the process of making a software system with the best efficiency and reduce the steps to the minimum throwing away all the unnecessary waste. There are lots of different methodologies that were made throughout theyears but only three stood out because they were the infrastructure of whats called not the Unified Modeling Language (UML). These three are Rumbaughs Object-Oriented-Technique (OMT) , Booch methodology and Jacobson methodology.

In our project we used Rumbaughs Object-Oriented-Technique (OMT) as it serves multiple points as we will discuss later in this section of the document. Let as just explain eachof the methodologies and the big picture of each of them.

	Rumbaugh Object-Modelling-Technique(OMT)	Booch	Jacobson
Steps	1- analysis 2- system design 3-object design 4-implementation	***-macro development process :*** 1. Conceptualization 2. Analysis and development of the model 3. Design or create the system architecture 4. Evolution or implementation 5. Maintenance ***-micro development process:*** 1. Identify classes and objects 2. Identify class and object semantics 3. Identify class and object relationships Identify class and object interfaces and implementation	1- use case scenario use case diagram
Advantages	Specifies the main parts of the system ina small number of steps.	Fully describes the system.	Uses a minimum number of steps.
Disadvantages	Describes the system in a limited point of view without looking at all the possible outcomes which mightcause some small ambiguity.	Too many steps with too many diagrams which requires technical background for developers. This costs the developers a lot of time and money just for two steps of thedevelopment.	Does not cover the main topics needed for the system which might cause some sort of ambiguity for the developers in the future implementation of the system or misunderstanding of the system.

Table 7

18.1.Justification

We chose the Rumbaugh Object-Oriented-Technique (OMT) because it has the most efficient effect in our Food ordering system as it covers the main parts of it and the lost parts whichare due to ambiguity can be clear from common sense. This means that we don’t need to assume anything so the whole system is clear. Also it had a spectacular effect in reducing thetime of searching for unknown specifications in each stage as we will discuss now.

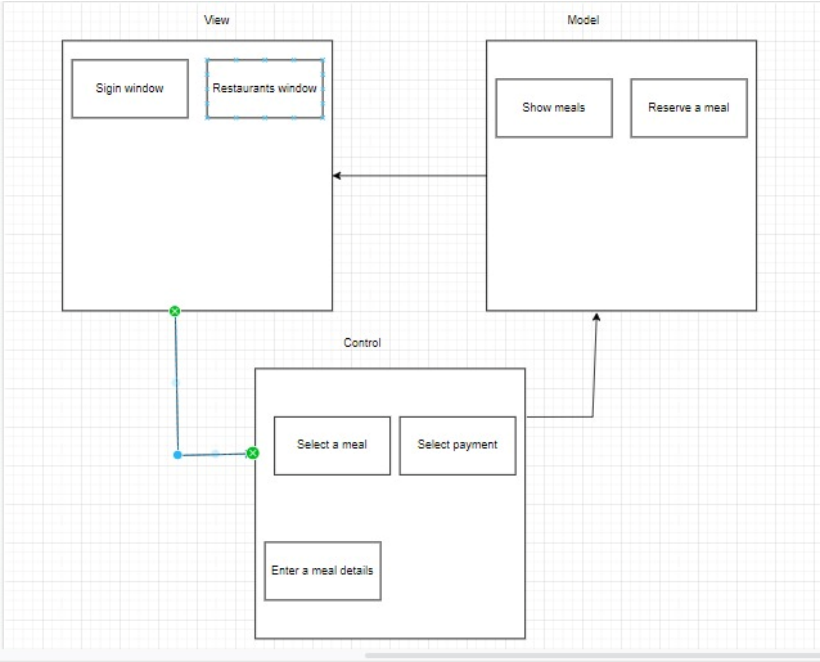


Figure 42

18.2.Analysis

In the analysis phase we discovered the main functionality of the food ordering system starting with its main responsibilities and reactions with the user in the use case diagram. Using the output of the use case diagram we started realizing the differentcomponents resulting in making the class diagram which was made in three phases seamlessly instead of starting with it. At last we extracted from both the components and their interactions the dynamic modeling by making the state diagram to show the different states of each object and the sequence of them.

18.3.System design

In the system design phase we started to see a close form of an architecture of the whole system which saved us lots of hard work that could have been spent trying to figure out vaguely. In this phase we just completed the architecture of thesystem and dividing the system into subsystems and packages.

18.4.Object design

After finishing the last two phases we nearly finished the overall look of the system all we needed to do is just looking for the details. We started importing the attributes and functions of each object from

making the dynamic and functional models. As we already extracted the dynamic model in the analysis phase we savedsome time in the progress.

18.5.Implementation

Finishing the main three phases of the OOAD we were ready to start implementing our analyses and designs into actual code which made it easy,neat and most importantly correct.

18.6.Alternative choices

18.6.1. Booch methodology

If we used Booch methodology it could have worked but it would be inefficient. Going through Booch methodology we would have started with the macro development process in which we would have had to Conceptualization, Analysis and development of the model whichis already made in the analysis phase of the Rumbaugh Object-Oriented-Technique (OMT) instead of two.

We also needed to Design or create the system architecture which are also done in a simpler way in Rumbaugh Object-Oriented-Technique (OMT) due to the information given fromthe previous phases. Evolution or implementation is in both of the methodologies.

Maintenance is one of the differences between the two methodologies but it can beimplemented after the evolution of the system with no problems.

Also while finishing the macro development process we would need to work on the micro development process which shows the real difference between the methodologies. We would have needed to identify classes and objects, Identify class and object semantics, Identify class and object relationships and Identify class and object interfaces and implementation which areall defined just in the analysis phase and the object design phase with hierarchy instead of fourdifferent phases that could cause change of ideas.

18.6.2. Jacobson methodology

Jacob methodology has some serious problems that can cause a huge misunderstandingof the system. Jacobson methodology only focuses on the lifecycle and sequence of the systemwithout focusing on the component, architecture or details of any of the objects in the system. This causes some kind of chaos implementing the system.

Jacob methodology could work on some specific software systems but in our case implementation is very important due to our need of a stable structure for the system. In Jacobmethodology we make a use case scenario and a use case diagram which are already made in the analysis phase in the in Rumbaugh Object-Oriented-Technique (OMT). So it is better to avoid it as it is does not meet our requirements.