Online Food Ordering System (*OFS*) Project Documentation

List of Contents:

1. **Software Requirements Specification**

Table of Contents

1. Introduction

Home delivery of fast via phone calls is popular. The process seems easy to use but at times there is miscommunication sometimes. The main drawback of placing orders via phone call is that there is no visual menu shown, the employees have to repeat a lot of things again and again to the customers. It's a time consuming process which at times irritates customers. It would be much more comfortable for the customers to have an online ordering system.

1.1 Purpose

This SRS defines External Interface, Performance and Software System Attributes requirements Online Food Ordering System. it will also describe how the system will perform and under which it must operate.

This document is intended for the following group of people:-

Developers for the purpose of maintenance and new releases of the software.
Management of the Restaurant Owner.
Documentation writers.
Testers.

1.2 Scope

This system will help to manage and run the restaurant business systematically. In this n1anagen1ent systen1, we will provide an app that can be used by the customers to order food. Customers can also give feedback through this app. So that the owner of the restaurant can evaluate the whole system. This will ultimately lead to hiring less waiters and creating an opportunity to appoint n1ore chefs and better kitchen places to serve food faster. Customers can also n1ake payment through debit or credit cards using POS which will be integrated with the n1anagen1ent software. Customers can see the current discount facility of the restaurant. Customers can also see the calorie chart which will increase consciousness about their health. All the information about daily expenses and profit will be saved in the system. Also the required information 's about employees will be saved in the system which can only be accessed by the system admin.

1.3 Definitions, Acronyms, and Abbreviations.

We will also use bold letters to emphasize main topics and for all the major functions of the system. Underline will represent hyperlink. Italic will represent acronyms and useful notes.
We will use some acronyms through this document. Abbreviations and definitions of some useful terms we will use are given below:

TERM	DEFINITION
System Admin	System admin is a person who is responsible for managing the whole system and who has full access to the system.
System User	A person who is using or operating the system but with a limited privilege.
Database	Collection of all the information monitored by this system.
Field	A cell within a form.
Software Requirements Specification (SRS)	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document.
Stakeholder	Any person who is involved in the development process of the software.
Point of Sale (POS)	A point of sale system is either a stand-alone machine or a network of input and output devices used by restaurant employees to accomplish their daily activities including food and beverage orders, transmission of tasks to the kitchen and other remote areas, guest-check settlement, credit card transaction processing, and charge posting folios.
OFS	Online Food Ordering System

1.4 References

www.google.con1-the world 's information.
www .wikipedia.con1-free online encyclopedia.
www .cnet.con1-technology portal.
www .slideshare.net-the world 's largest professional content sharing community.
IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5 Document Overview

This document is intended for different types of readers such as restaurant owner, system designer, system developer and tester. By reading this document a reader can learn about what the project is implemented for and how it will present it's basic ideas.
This document has a sequential overview of the whole project so if a reader reads the document from top to bottom , he will get a clear idea about the project.

Section 1.0	Discusses the purpose and scope of the software.
Section 2.0	Describes the overall functionalities and constraints of the software and user characteristics.
Section 3.0	Details all the requirements needed to design the software.

2. The Overall Description

2.1 Product Perspective

The Online food ordering System helps the restaurant manager to manage the restaurant more effectively and efficiently by computerizing meal ordering, billing and inventory control.
The system processes transactions and stores the resulting data. Reports will be generated from these data which help the manager to make appropriate business decisions for the restaurant. For example, knowing the number of customers for a particular time interval, the manager can decide whether more waiters and chefs are required. Moreover, easily calculate daily expenditure and profit.
The whole management system is designed for a general Computerized Digital Restaurant. So that any restaurant owner can get it and can start an automated process to his restaurant.

2.2 Product Functions

The major functions that OFS performs are described as follows:-

1.Registration:	Application provides a link for the Users/Client Registration.
2.Log In:	Administrator and Client can log in by entering username and password and manage their work on the website.
3.Save information:	Client enter all its necessary information by filling personal Information and system save that information.
4.Change requirements:	Customers can change any of their information any time.
5.Food Menu:	Admin can insert, update and delete the food items from the menu list.

6.Show Food Menu: There is a list of all types of food the company is dealing with the available themes.
7.Record Order Details: Customer can select food items from menu and can add the desired food items toThe cart. Customer can place the order and gets the confirmation against that Order in the form of order number.
8.Show Order Status: Customer can check the status of his/her placed order.
9.View Orders: Admin can view the placed order and delivered orders.

2.3 User Characteristics

The Online food ordering System has five active actors and one cooperating system.

The customer can access the system through wifi connection and order food .

The Chef can see the order and after preparing the food he will tell the system that the food is ready.

The waiter can get the confirmation of food from the chef through the system and deliver it to the right table.

The cashier can access the system and receive the payment from customers.

The Admin can edit the price, count total earnings and expenditure.

2.4 Constraints

There are some constraints which cost more for the system. If those constraints can be overcome then this whole system will perform best. They are-

1. IOS App and Windows App.
2. Information flow or data flow can be controlled and n1ore effective.
3. Minimum amount for ordering food is 500 .
4. Minimum orders 2.
5. Java can be used for more security.

2.5 Assumptions and Dependencies

If this system has an IOS and Windows app then customers who use such a smartphone will be more benefited. If there are more Tablets for each table the whole system performance will be better. For a more secure system it is beneficial to use a CC camera and TV.

3. External Interface Requirements

There are many types of interfaces as such supported by this software system namely; User Interface, Software Interface and Hardware Interface.

3.1.1 User Interface Requirements

The user interface will be implemented using any android smartphone app browser. This interface will be user friendly. So that every kind of customer can place the food order easily. Customers can also give feedback through it easily with some demo
comment or if they are keen to write their review on their own they can do it.

3.1.2 Hardware Interfaces

There shall be a logical address of the system in IPv6 format.

3.1.3 Software Interfaces

The system shall communicate with the Configurator to identify all the available components to configure the product.
The system shall communicate with the content manager to get the product specifications.

3.1.4 Communications Interfaces

Communication function required the Internet protocol version 6 and it will follow HTTPS. It will use FTP for the whole system with a local server. And email
communication to device to device of the system.

3.2 Behaviour Requirements

3.3.1Use Case View

The use cases for each of the actors are described in this section.

Customer Use Case

Use case: Order Food

Description

The Customer can order food and see their payment receipt and pay.

Chef Use Case

Use case: Prepare Food

Description

The chef can see the orders of customers and checks whether this order can be taken or not and then confirms the order and starts preparing the food. When the food is ready the chef alerts the waiter to serve the food. He can also edit what ingredients are available and what ingredients are demanded.

Waiter Use Case

Use case:Serve Food

Description

The waiter can see the food orders and the ready foods in the kitchen to be served. After serving the food the waiter will mark the order as served.

Cashier Use Case

Use case:	Take Payment

Description

The Cashier can only take payment from the customer and save it into the system database with respect to the food item and also check if the customer is eligible for a discount. If yes then take the payment accordingly.

Admin Use Case

Use case:	Maintain System

Description

The Admin has full access to the system. He maintains the whole system to ensure better and secure service and solves any error appeared in the system.

4. Other Nonfunctional Requirements

### 4.1	Performance Requirements

- The product will be based on a local server.
- The product will take initial load time.
- The performance will depend upon hardware components.
- Payment system will be fully secure through the POS system.

- Different databases for employees.

4.2 Safety and Security Requirements

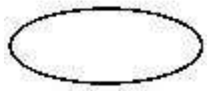
- The source code developed for this system shall be maintained in configuration management tools.
- The whole system is secured. Only Admin can access all the data.
- The password shall be 6-14 characters long.
  Passwords shall not contain name of customers as they are easy to be
  Hacked.
  Passwords can contain digits, hyphen and underscore.

- This system will use HTTPS. Because of this protocol this is more secure.
- This system will use a secured POS system.

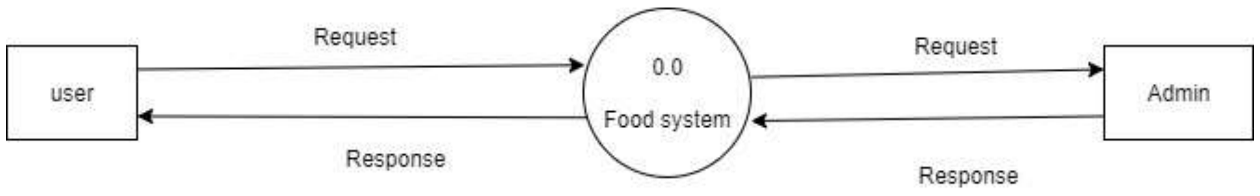
5 Other Requirements

None.

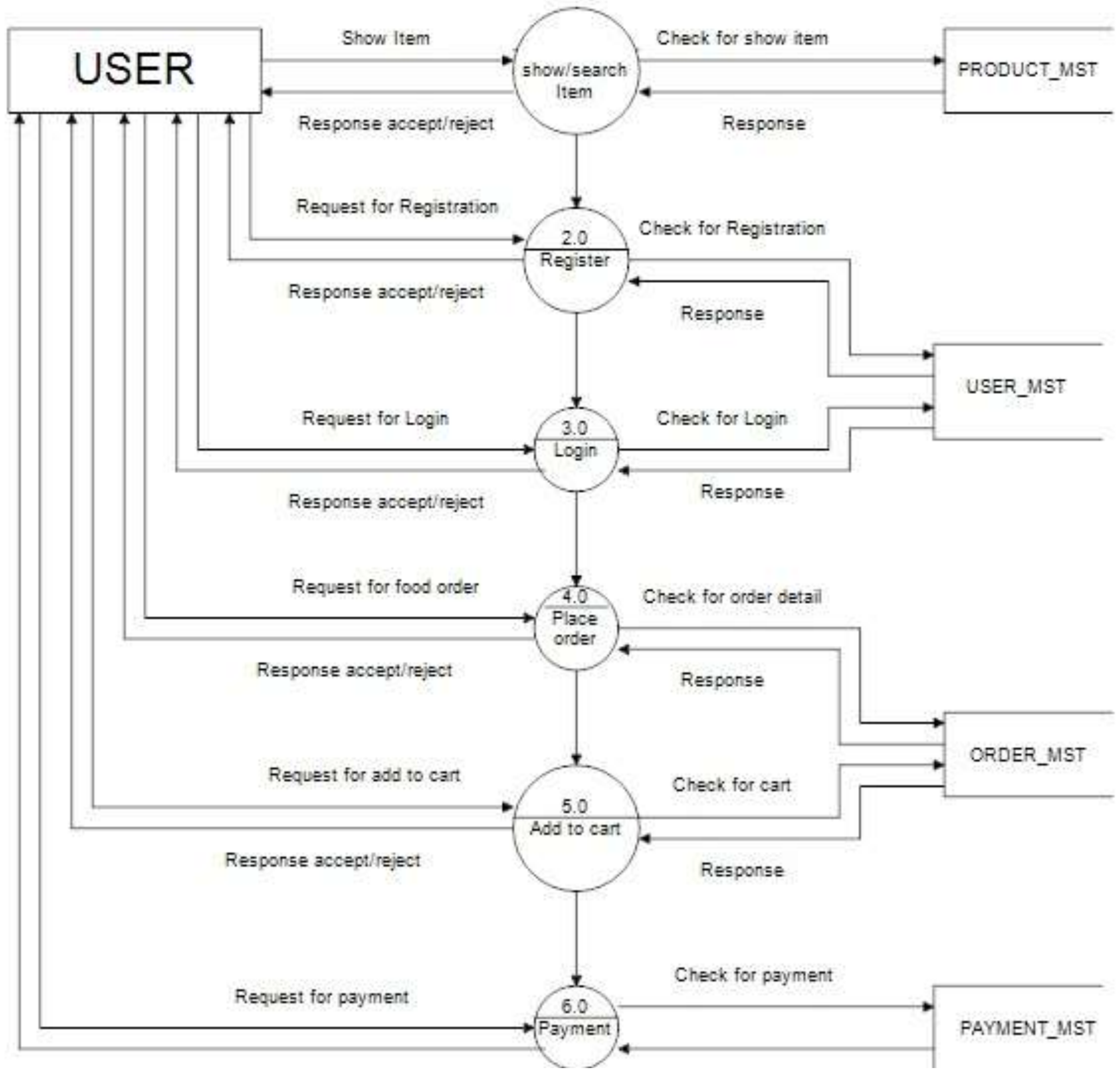
2. **Data Flow Diagrams**

Symbol	**Description**
	Data Flow – Data flow are pipelines through the packets of information flow.
	**Process :** A Process or task performed by the system.
	Entity : Entity are objects of the system. A source or destination data of a system.
	Data Store : A place where data to be stored.

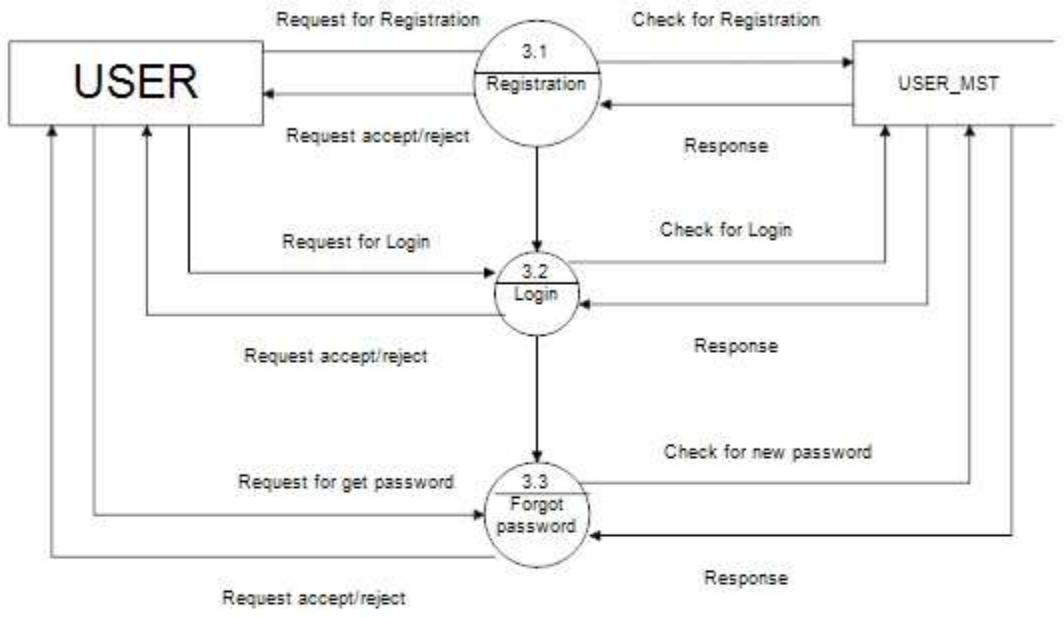
Context level DFD :



Level 0 DFD:

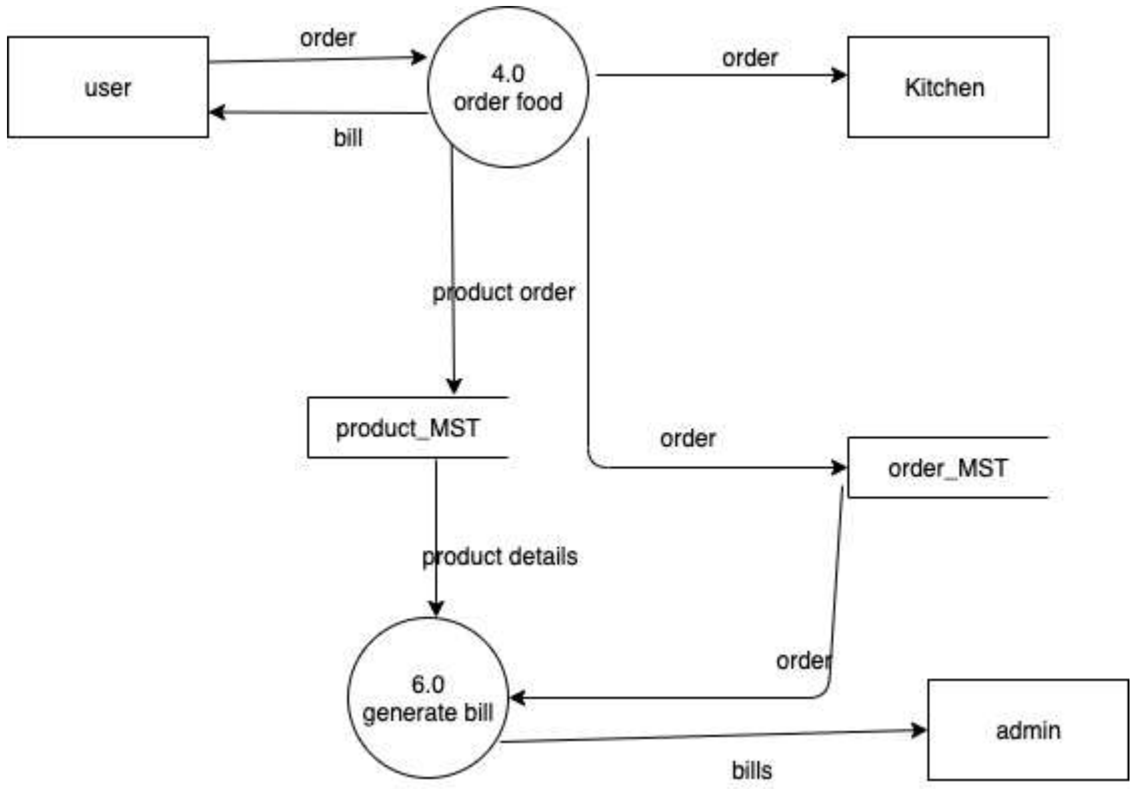

USER
Show Item
show/search Item
Check for show item
PRODUCT_MST
Response accept/reject
Response
Request for Registration
2.0
Register
Check for Registration
Response accept/reject
Response
USER_MST
Request for Login
3.0
Login
Check for Login
Response
Response accept/reject
Request for food order
4.0
Place order
Check for order detail
Response accept/reject
Response
ORDER_MST
Request for add to cart
5.0
Add to cart
Check for cart
Response accept/reject
Response
Check for payment
Request for payment
6.0
Payment
PAYMENT_MST

Level 1 DFD:


Request for Registration
Check for Registration
USER
3.1
Registration
USER_MST
Request accept/reject
Response
Request for Login
Check for Login
3.2
Login
Request accept/reject
Response
Check for new password
Request for get password
3.3
Forgot password
Response
Request accept/reject

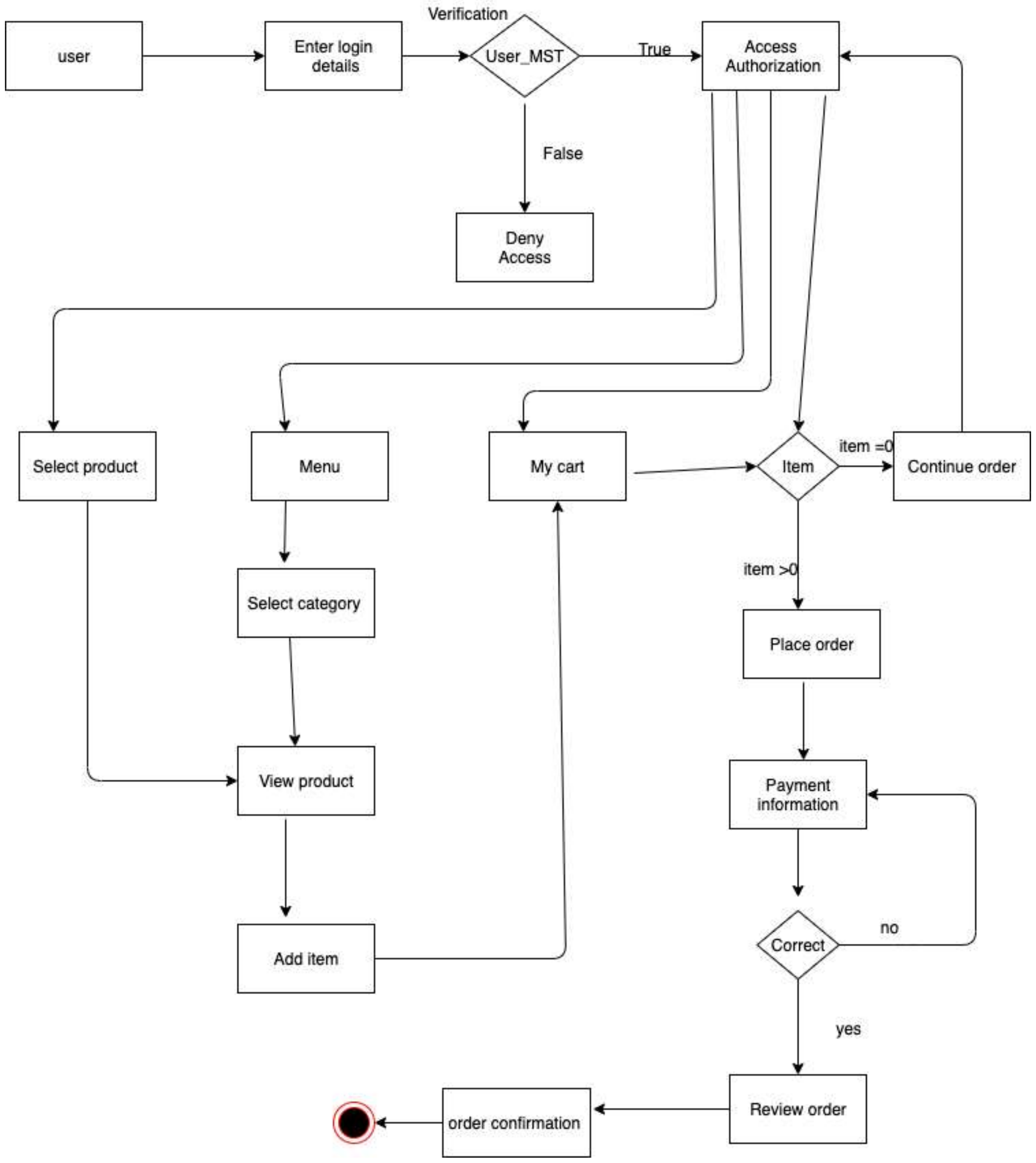
3. **The function oriented diagram**

DFD



Product order = category of product + Product name
Order = order number + order id
Bills = order id + order cost

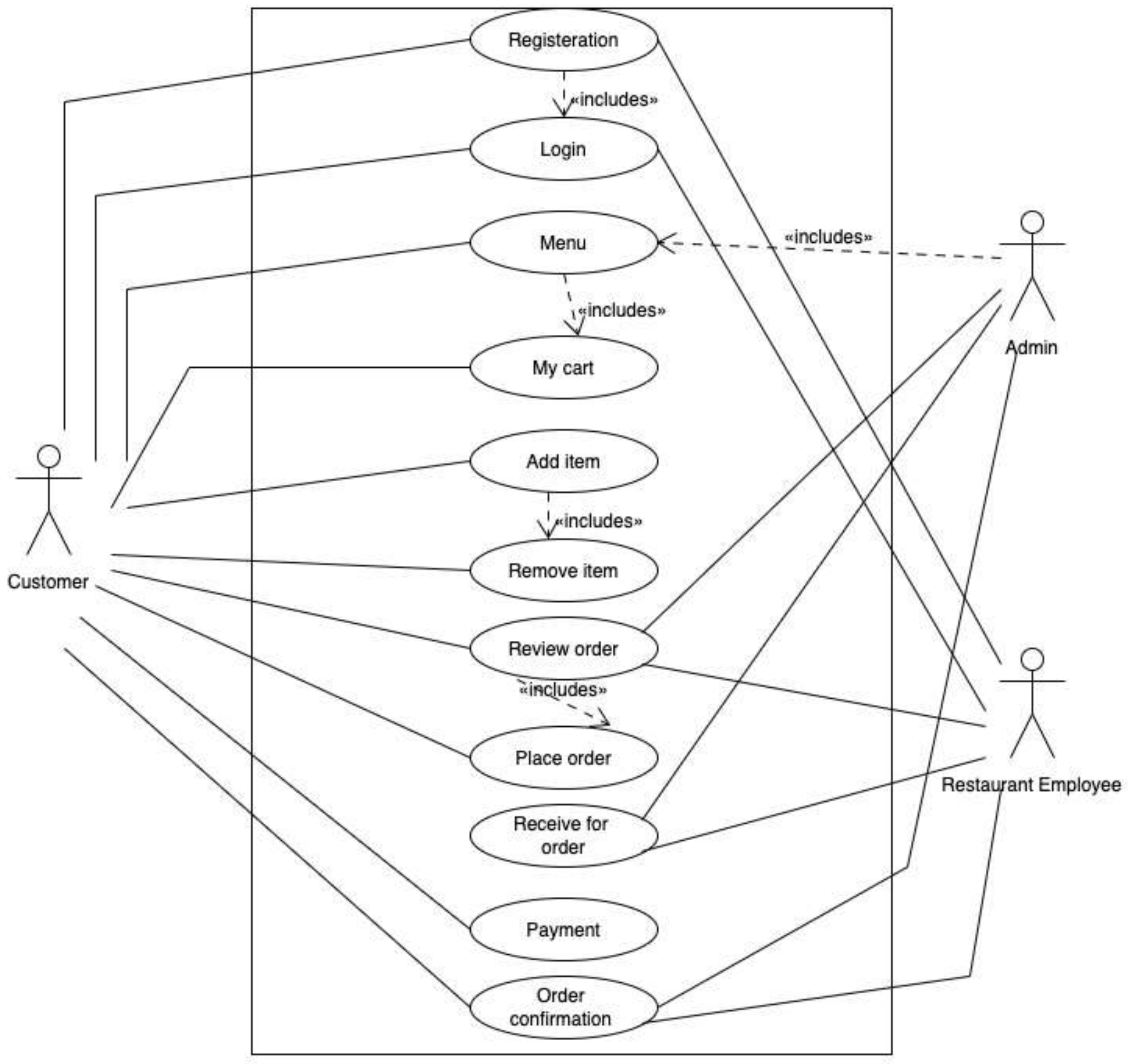
Structured chart



4. **Use case diagram**

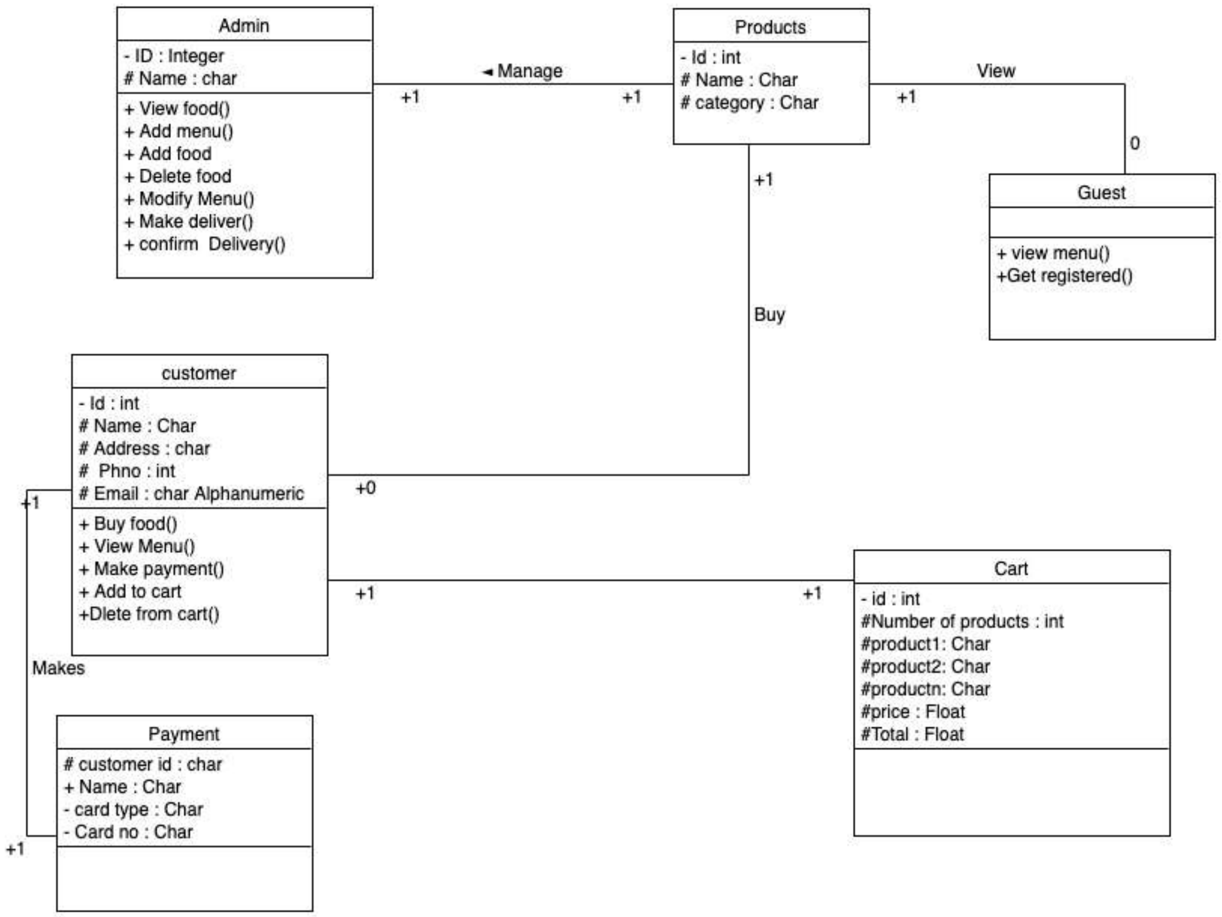
Use cases specify desired behavior.

An actor represents a set of roles that users of use case play when interacting with these use cases.
Include - use cases that are included as parts of other use cases. Enable to factor common behavior.



5. **Class diagram for online food system**

UML class diagram: a picture of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other



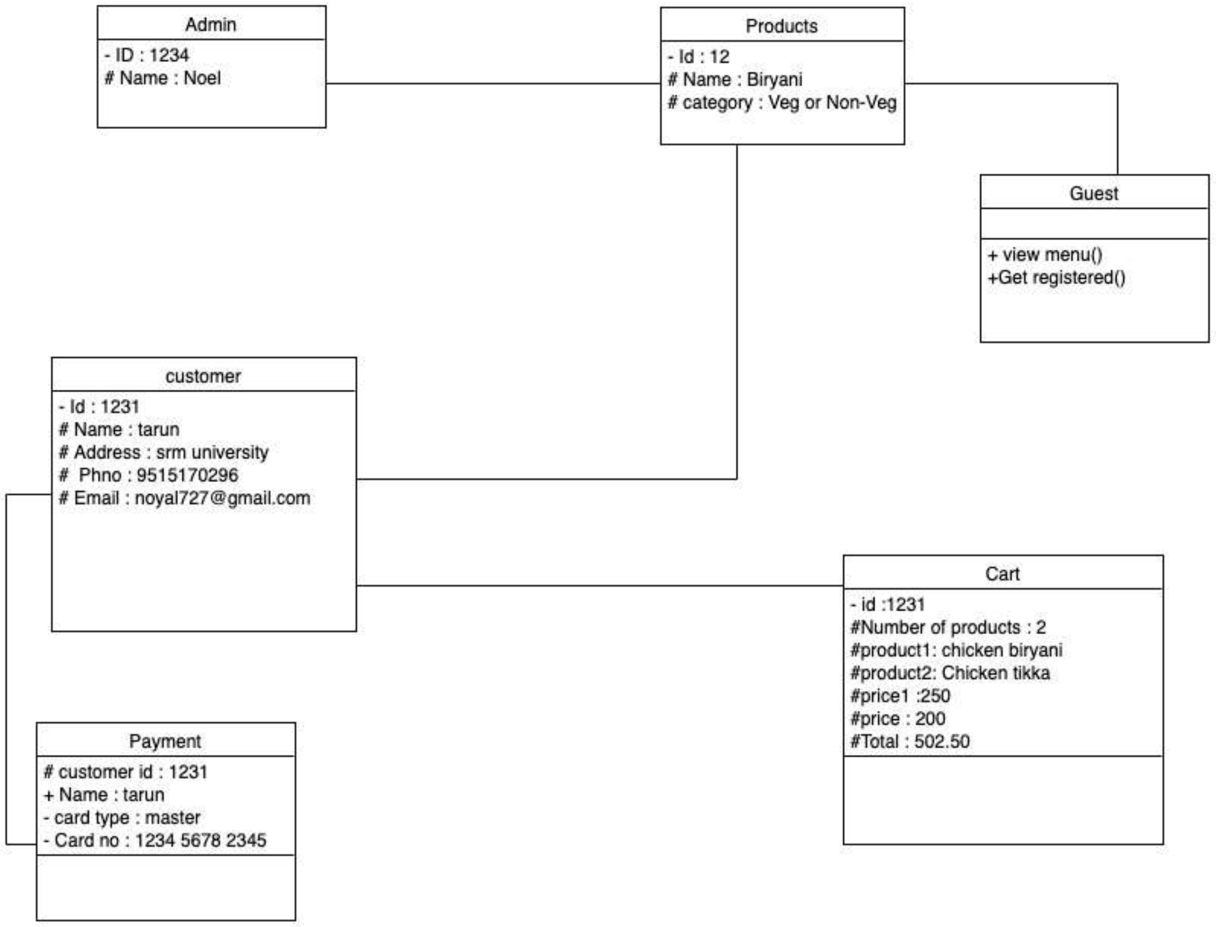
Object diagram online food system

It shows object relationships in a system.

Understand object behavior and their relationships.

Shows a static view of a system.

Forward and reverse engineering.



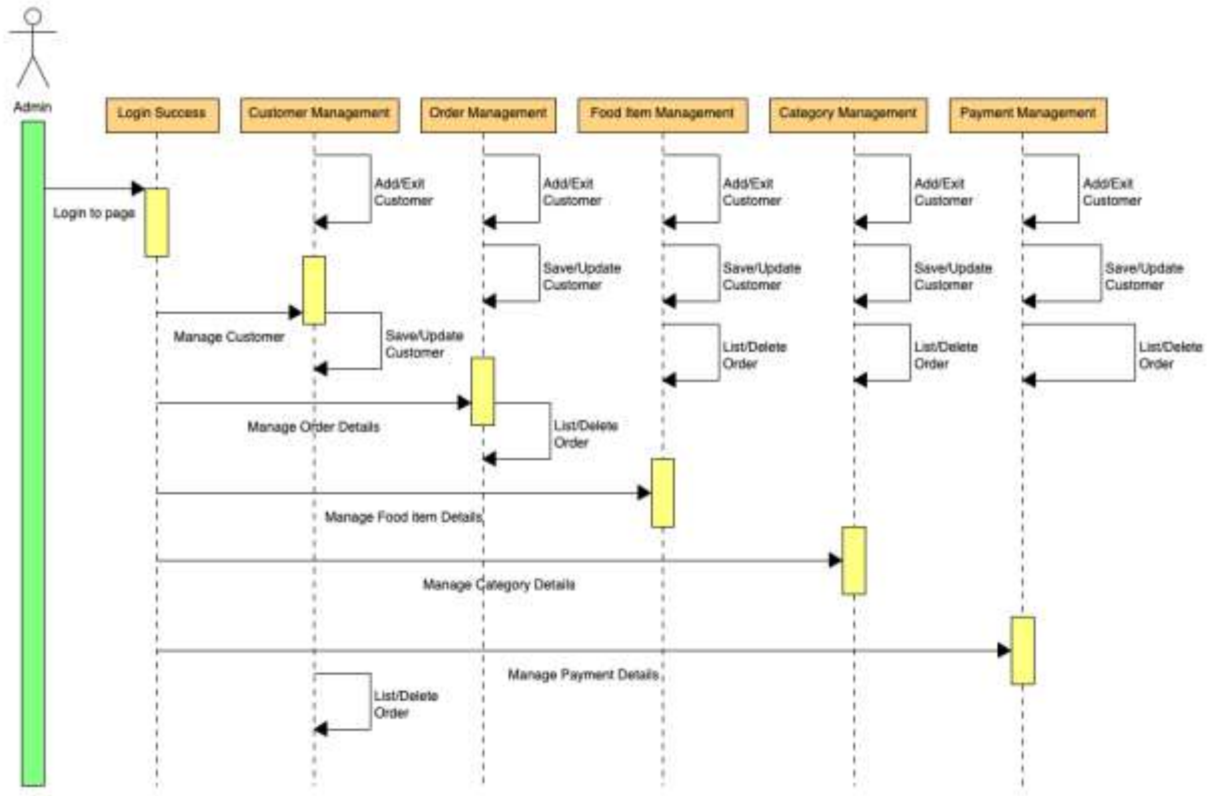
6. **Sequence diagram for online food ordering system**

A Sequence diagram is an interaction diagram that details about the operation that is carried out.

The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system.

The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.

A sequence diagram is used to capture the behavior of any scenario.



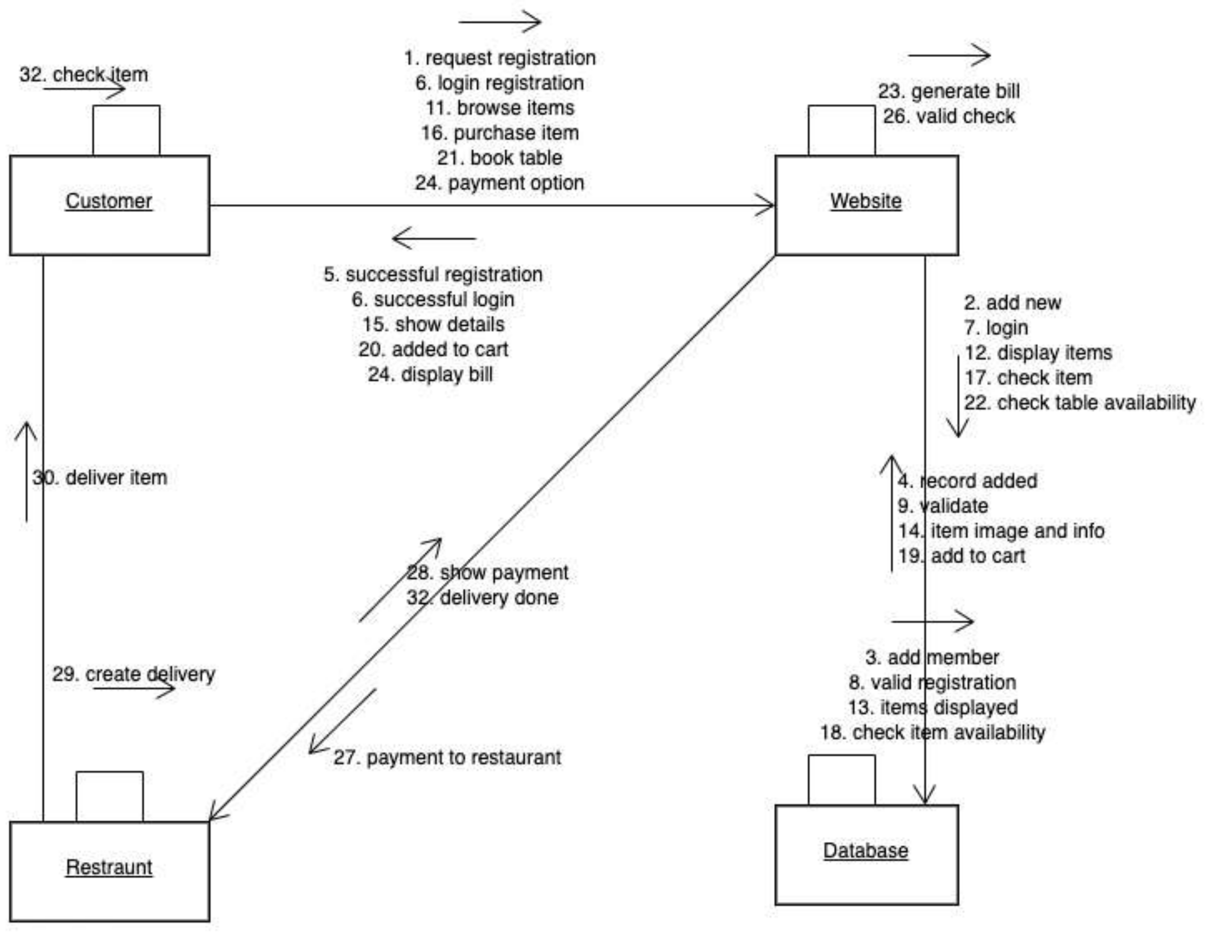
Collaboration diagram for online food ordering system

Collaboration diagrams (known as Communication Diagram in UML 2.x) are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case.

A Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.

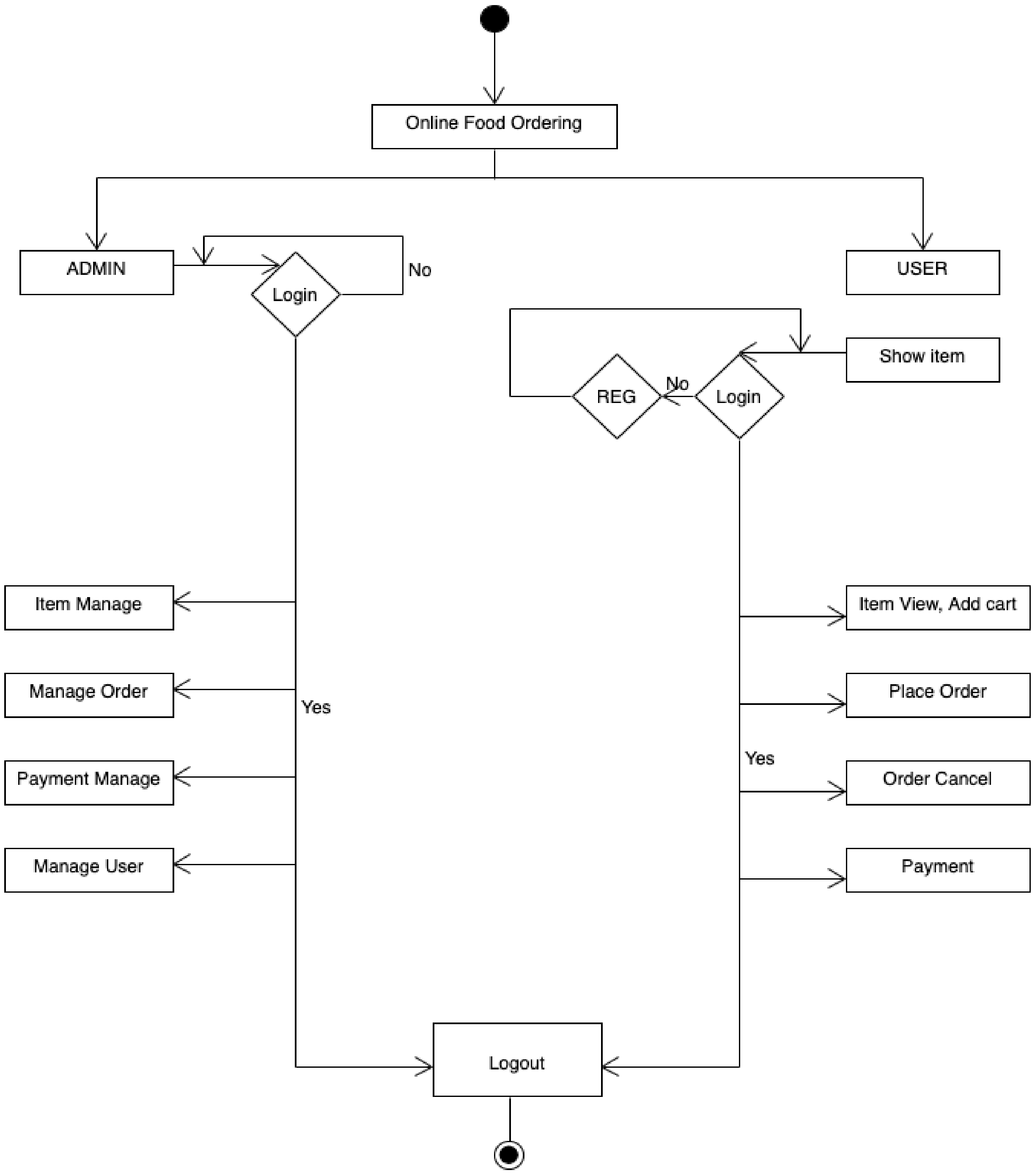
A Collaboration defines a set of participants and relationships that are meaningful for a given set of purposes.

A Collaboration between objects working together provides emergent desirable functionalities in Object-Oriented systems.



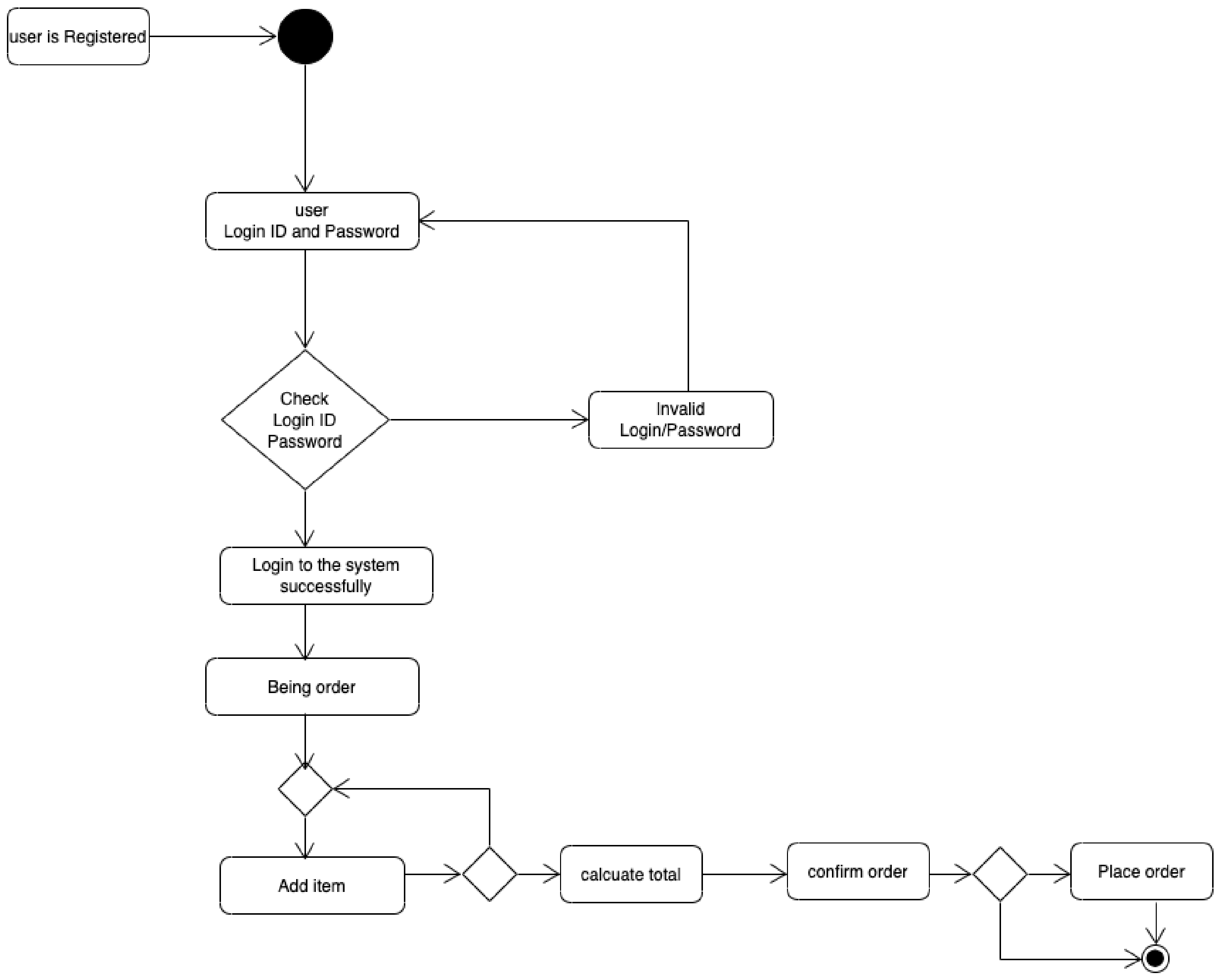
7. **State-chart diagram for online food ordering system**

A statechart diagram shows a state machine, which specifies the sequences of states that an object can be in, the events and conditions which cause the object to reach those states, and the actions which take place when those states are reached.



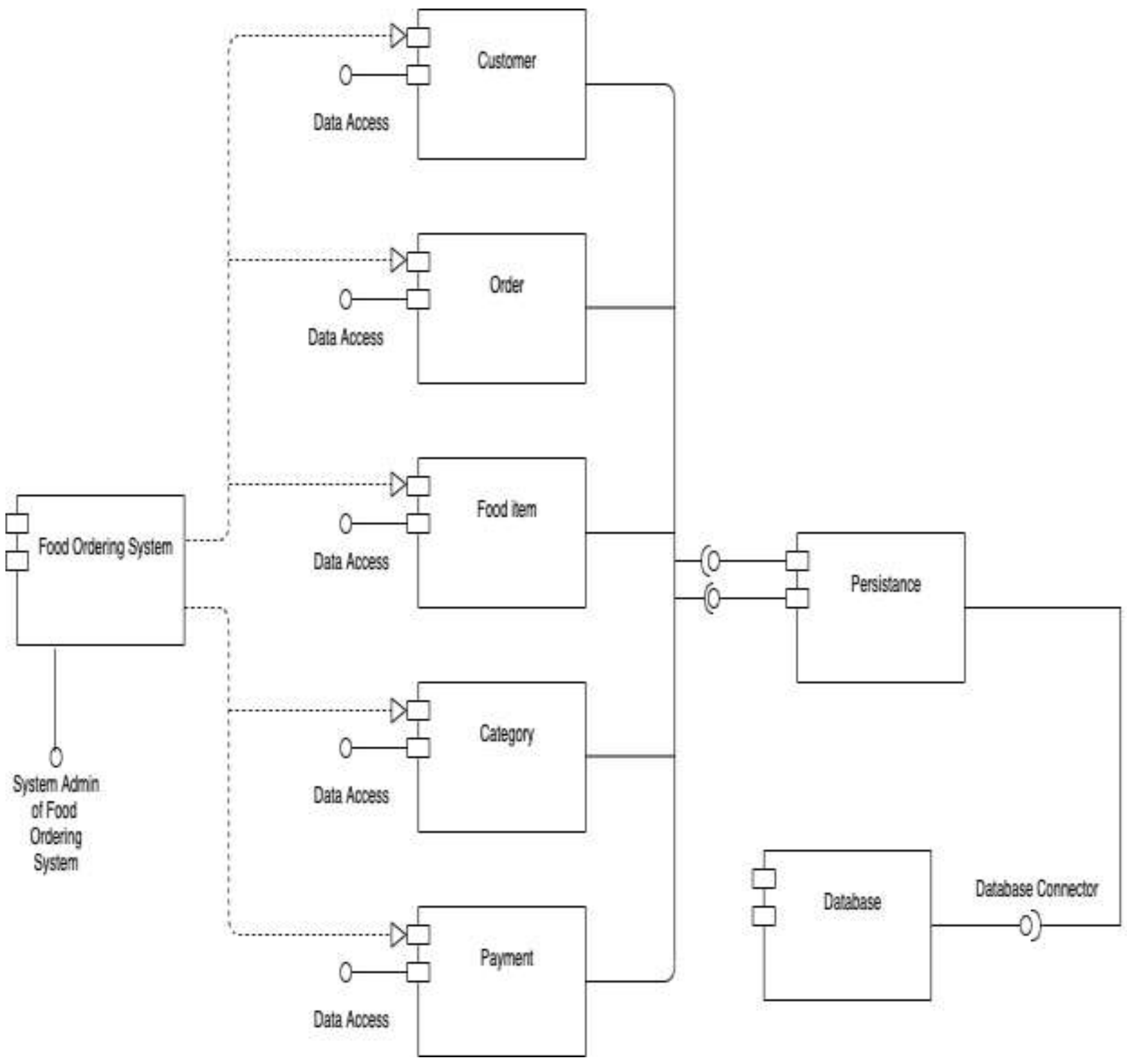
Activity diagram for online food ordering system

Activity diagram is UML behavior diagram which shows flow of control or object flow with emphasis on the sequence and conditions of the flow.



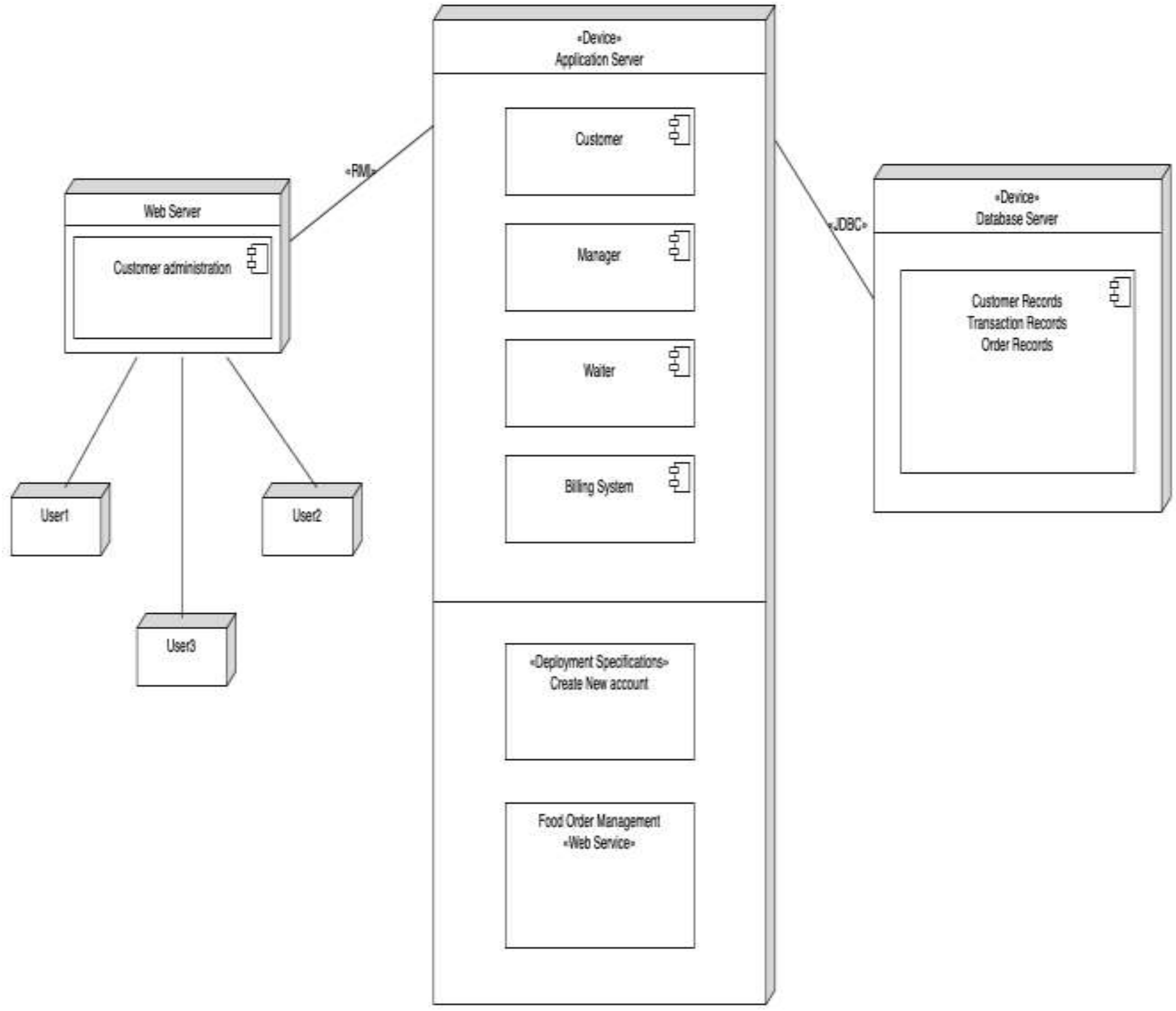
8. **Component diagram for online food ordering system**

Component diagram shows components, provided and required interfaces, ports, and relationships between them.



9. **Deployment diagram for online food ordering system**

Deployment diagram is a structure diagram which shows architecture of the System as deployment (distribution) of software artifacts to deployment targets.



10. To perform various testing tools unit testing and integration testing

10A. **Unit Testing**

```
public class Complex {
int real_part; int imaginary_part;
public Complex(int r, int i) {
real_part=r;
imaginary_part=i;
}
public Complex() {
real_part=0;
imaginary_part=0;
}
public boolean Equal(Complex c) {
boolean result = false;
if ((real_part==c.get_r()) && (imaginary_part==c.get_i())) result=true;
return result;
}
public Complex Add(Complex c) {
Complex result = new
Complex(c.get_r()+real_part,c.get_i()+imaginary_part);
return result;
}
public int get_r() { return real_part;}
public int get_i() { return imaginary_part; }
}
```

JUnit test cases for the above complex class

```
package javaapplication4;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
```

```
}



public class ComplexTest {

    Complex c1;
    Complex c2;

    public ComplexTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
        c1 = new Complex(7,3);
        c2 = new Complex(12,6);
    }


    @After
    public void tearDown() {
    }

    @Test
    public void testEqual() {
        System.out.println("Equal");
        assertTrue(!c2.Equal(c1));

        assertTrue(c1.Equal(new Complex(7,3)));
    }

    @Test
    public void testAdd() {
        System.out.println("Add");
        Complex result = c1.Add(new Complex(5,3));

        assertEquals(result.get_r(),c2.get_r());
        assertEquals(result.get_i(),c2.get_i());
    }
```

```
 @Test
 public void testGet_r() {
   System.out.println("get_r");
   Complex instance = new Complex(5,10);
   int expResult = 5;
   int result = instance.get_r();
   assertEquals(expResult, result);

 }
 @Test
 public void testGet_i() {
   System.out.println("get_i");
   Complex instance = new Complex(5,10);
   int expResult = 10;
   int result = instance.get_i();
   assertEquals(expResult, result);

 }

}
```

```
Testsuite: javaapplication4.ComplexTest
Add
Equal
get_i
get_r
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.09 sec

------------- Standard Output ---------------
Add
Equal
get_i
get_r
------------- ---------------- ---------------
```

10B. **Integration Testing**

CalculatorService.java

```
public interface CalculatorService {
  public double add(double input1, double input2);
  public double subtract(double input1, double input2);
  public double multiply(double input1, double input2);
  public double divide(double input1, double input2);
}
```

MathApplication.java

```
public class MathApplication {
  private CalculatorService calcService;

  public void setCalculatorService(CalculatorService calcService){
    this.calcService = calcService;
  }

  public double add(double input1, double input2){
    return calcService.add(input1, input2);
  }

  public double subtract(double input1, double input2){
    return calcService.subtract(input1, input2);
  }

  public double multiply(double input1, double input2){
    return calcService.multiply(input1, input2);
  }

  public double divide(double input1, double input2){
    return calcService.divide(input1, input2);
  }
}
```

Integration Test cases

MathApplicationTest.java

```
import static org.mockito.Mockito.when;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class MathApplicationTest {

  @InjectMocks
  MathApplication mathApplication = new MathApplication();
  @Mock
  CalculatorService calcService;

  @Test
  public void testAdd(){
    when(calcService.add(10.0,20.0)).thenReturn(30.00);
    Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
  }
}
```

output

```
Testsuite: javaapplication4.MathApplicationTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.416 sec

test:

BUILD SUCCESSFUL (total time: 2 seconds)
```