

ONLINE CAKE ORDERING SYSTEM

A PROJECT REPORT

Submitted By

T.DIVYA

Register Number : 511921622007

In partial fulfillment of award of the degree of

MASTER OF COMPUTER APPLICATIONS

FACULTY OF INFORMATION AND COMMUNICATION

ENGINEERING

DEPARTMENT OF COMPUTER APPLICATIONS

PRIYADARSHINI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI - 600 025.



SEPTEMBER 2023

CERTIFICATE OF EVALUATION

COLLEGE NAME : PRIYADARSHINI ENGINEERING COLLEGE
VANIYAMBADI.

BRANCH & NAME : MCA, 4TH SEMESTER

SUBCODE & NAME : MC4411, PROJECT WORK

MONTH & YEAR : SEPTEMBER 2023

Name of the student who have done the project	Registration Number	Title of the Project	Name of the Supervisor with Designation
DIVYA T	511921622007	ONLINE CAKE ORDERING SYSTEM	MR.S.SENTHILMURUGAN,MCA ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER APPLICATIONS

The report of the project work submitted of the fulfillment of Master of Computer Applications of Anna University was evaluated and confirmed to be reports of the work done by the above student.

INTERNAL GUIDE

Mr. S. Senthilmurugan, MCA.,
Associate Professor & HOD,
Department of Computer Application,
Priyadarshini Engineering College,
Vaniyambadi - 635 751.

HEAD OF THE DEPARTMENT

Mr. S. Senthilmurugan, MCA.,
Associate Professor & HOD,
Department of Computer Application
Priyadarshini Engineering College,
Vaniyambadi - 635 751.

ABSTRACT

ONLINE CAKE ORDERING SYSTEM

An online cake ordering system allows the users to show the various cake products, services and the quality of the cakes provided to the customers. And also there is login facilities for Admin, Customers and to maintain their account with the cake shop. Cakes are now available in online store and the customers can purchase through online. The project consists of list of cake products displayed in various categories. The user may browse through these items as per categories. If the user likes a product he may add it to his shopping cart. And also customers can follow, Order. Here we use user friendly interface to make the entire frontend. The middle tier or code behind model is designed for fast processing. And MYSQL serves as a backend to store cake products lists data. Thus, the project brings an entire cakeshop through online and makes it easy to know more about the cakes.

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE OF THE PROJECT

The main objective of the project is to design and develop a user friendly system.

- Easy to use and an efficient computerized system.
- To develop an accurate and flexible system, it will eliminate data redundancy.
- Computerization can be helpful as a means of saving time and money.
- Less chances of information leakage.
- Provides Security to the data by using login and password method.

1.1 ABOUT THE PROJECT

It is a system that allows users to check for various cakes available at the online store and purchase online. The project consists of list of Cakes and bakery products displayed in various categories. The user may browse through these items as per categories. If the user likes a product he may add it to his shopping cart. He may even pay through a credit card or cash on delivery. User has also option for ordering custom cakes according to their requirements like cake's flavor, size, shape and so on. Thus the online Cake shopping project brings an entire cake shop online and makes it easy for both buyer and seller.

CHAPTER 2

ORGANIZATION PROFILE

2.1 ABOUT THE ORGANIZATION

Zeboto is one of the prominent software companies that helps other organizations to transform into digital enterprises. It differentiates you from competitors in the market and provides better engagement with customers, partners and employees. We clearly understand that all customer landscapes are not of same kind. That's why Zeboto sourcing methodology encompasses a clear solution especially crafted to address the clients' issues. We offer a phased approach towards your business drivers and help IT organizations to align their goals towards the overall vision of the business. Our main aim is to develop in a constant manner and become a leading performer in this competitive global marketplace. Fortunately, we have been able to gather a crew of professionals that can shape and mold their collective experiences, all of them possess outstanding talent that can help to accelerate your organization.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Existing system used in the company is manual. The organization maintained the details in the form of records. It should improve efficiency and effectiveness. The purpose of the project is to build an application to reduce the manual work for managing the customer details, project details, order details, etc.

To maintain in records is very difficult because more manual power can consume. More workers to be engaged in the process. It is very difficult to obtain details immediately or within the particular time. If they want to get any payment details, it requires very long time to take it. The seller doesn't know that the customer is satisfied in their product or not. The seller doesn't know whether their product is reached in the market or not.

3.2 PROPOSED SYSTEM

The proposed system means the process that should be going to implement. The new features that should be included to maintain the details and an easy contact with the customer. There is a feedback option from the customer about their product. It makes a good relationship between the product seller and the customer.

3.3 METHODOLOGY

The online cake ordering system project can be developed using an agile model methodology that emphasizes collaboration, flexibility, where the web developer works on small chunks of project. The Agile methodology allows for frequent feedback and adaptation, which can help ensure that the project meets the requirements and expectations of the stakeholders.

The agile methodology involves the following phase

Let's observe the lifecycle of the Agile methodology:

1.Understanding the client's requirement:

The project is initiated by gathering a part of the information about the project from the client. The initial requirements are identified, prioritized and the necessary resources are selected. An estimate is calculated for the entire work. A demo is created or selected as to how the work will be demonstrated.

2.Sprint Planning:

A sprint is nothing but a period during which the task is completed and ready for review. To make it more target based on the end user, the web developer firstly decides the duration of the sprint. Hence, the client's feedback will be received on a regular basis. This helps in debugging the bugs and fixing every minor problem on time.

3.Designing the product:

Now the designing of the website begins. The tasks are allocated to the respective developers. Once started, a track is kept on the tasks in the form of a board. The list of tasks moves from "To Do" list to "In Progress". This, in turn, gives expertise, continuous delivery, domain knowledge, and faster feedback. When a part of the website is ready, the next phase called "Testing" is done.

4. Testing:

The web development agency is involved in content and navigation right from the beginning. The web developer entails the small delivery sets to the client. The client gives rapid feedback. The designs are tested simultaneously, and the results are demonstrated immediately.

5. Feedback and preparing for the next set of work:

Once the feedback is received from the client at each stage, the results are discussed within the team. How the development process can be improved and what further steps can be taken, becomes the deciding factor for the next set of work.

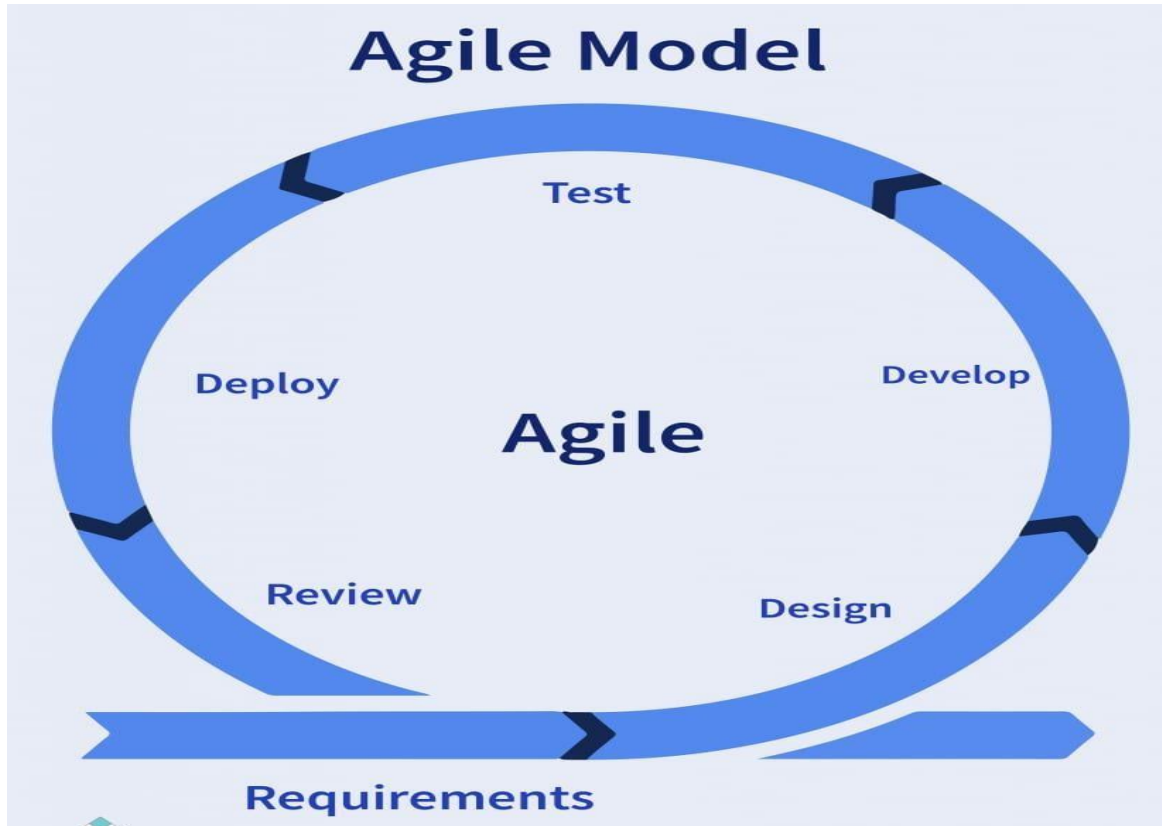


Fig No: 3.3 Agile methodology

CHAPTER 4

SYSTEM CONFIGURATION

4.1 HARDWARE SPECIFICATION

Processor : INTEL(R)Core(TM)i5-6006U CPU @2.00GHz.
System Type : 64-bit operating system, X64 based processor
Hard Disk : 400GB
Monitor : 14' Color Monitor.
Mouse : Optical Mouse.
Ram : 8GB

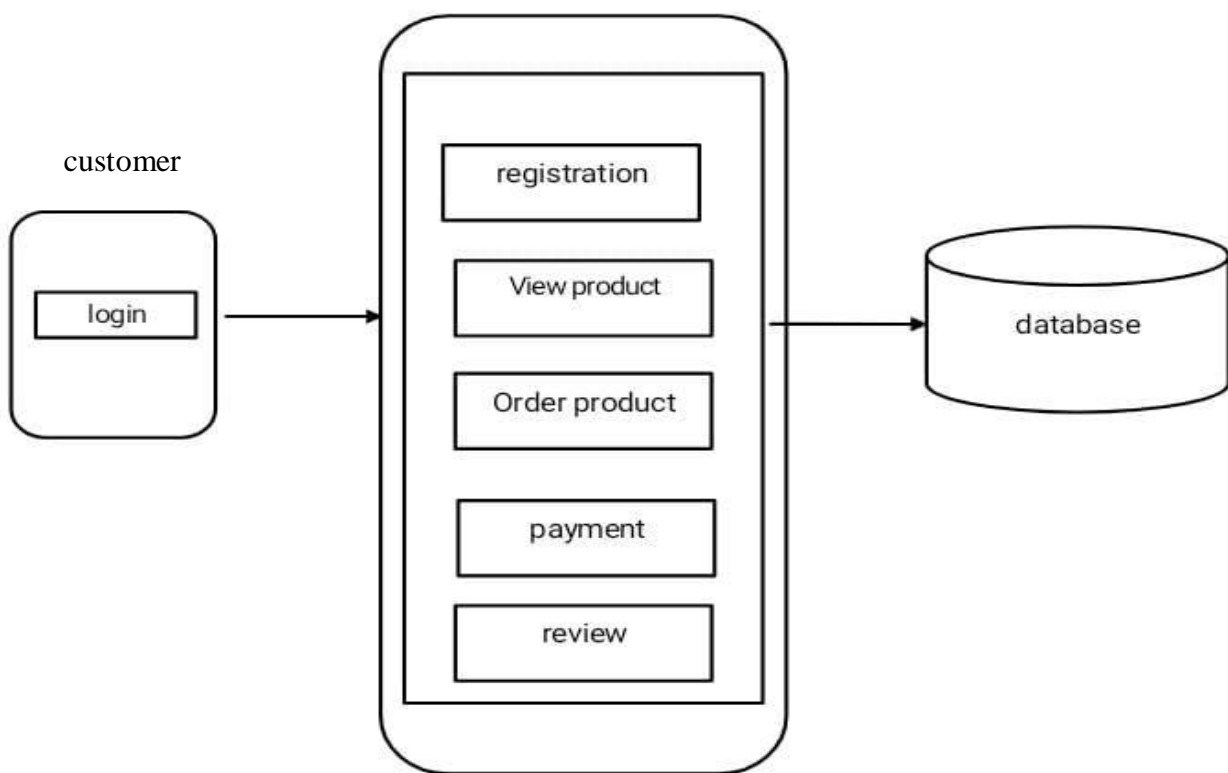
4.2 SOFTWARE SPECIFICATION

Operating System : Windows 11
Frontend : HTML, CSS,JAVASCRIPT
Backend : PYTHON,MYSQL.
Framework : Django

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE



5.1.2 ARCHITECTURE DIAGRAM

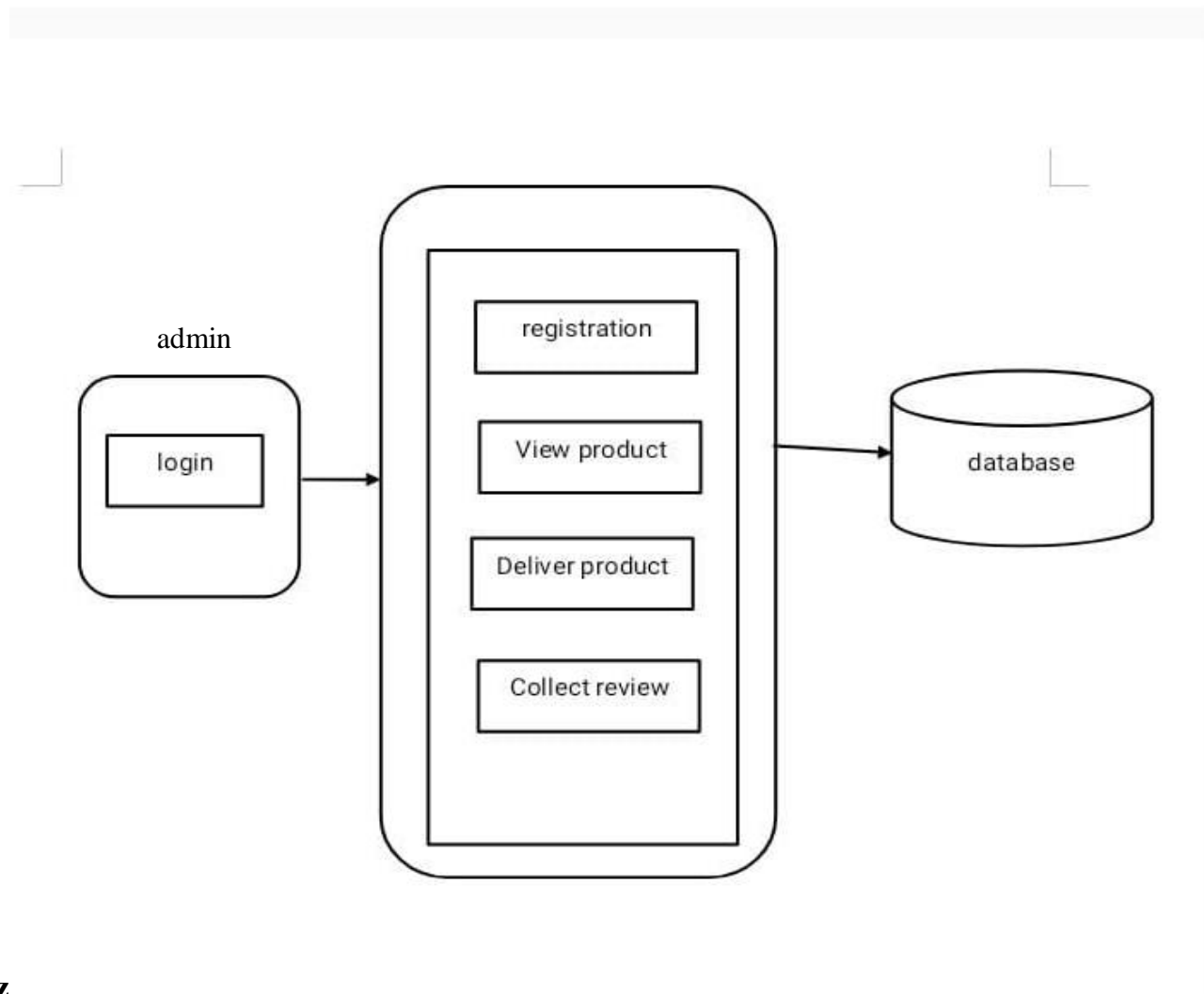


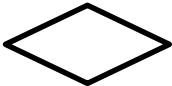


Fig No: 5.1.2 Architecture diagram

5.3 E - R DIAGRAM

Entity-Relationship Diagram is a graphical representation of entities and their relationship to each other's. It describes how data is related to each other. An entity is a piece of data - an object or a concept about which data is stored. A relationship is how the data is shared between entities

symbol	Name	Description
	Entity	An entity can be any object, place, person or anything.
	Attribute	An Attribute Describes a property or characteristics of an entity.
	Relationship	A Relationship Describes relation between entities.

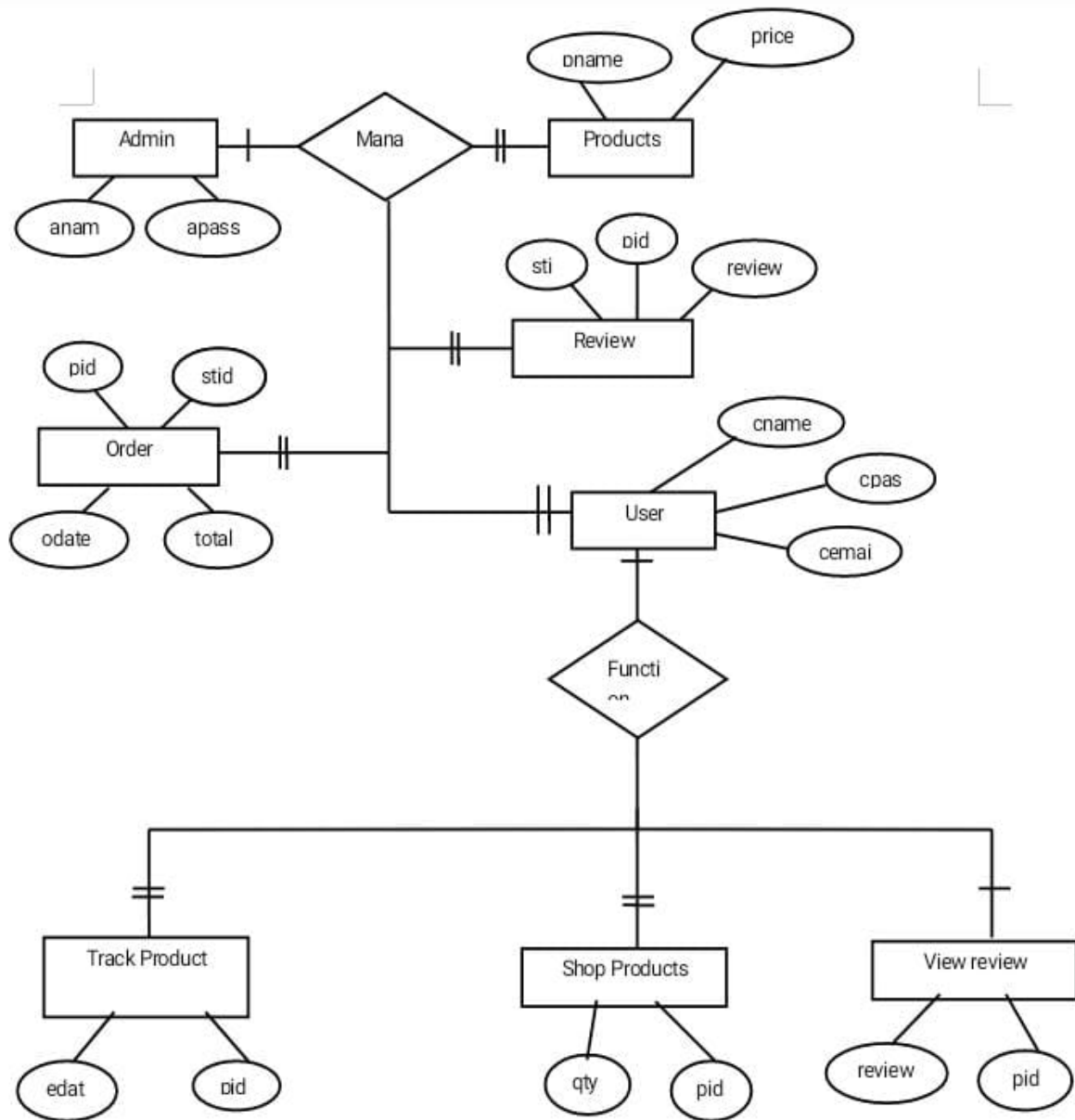


Fig No: 5.3.1 Use case diagram

5.4 UML DIAGRAM

5.4.1 USECASE DIAGRAM

Use case diagram model behavior within system and helps the developer understand of what the user requires. The stick man represents what's called an actor. it is a behavioral diagram that shows a use cases and actors and their relationship. it is an associated between the use cases and actors. an actor represents a real world object.

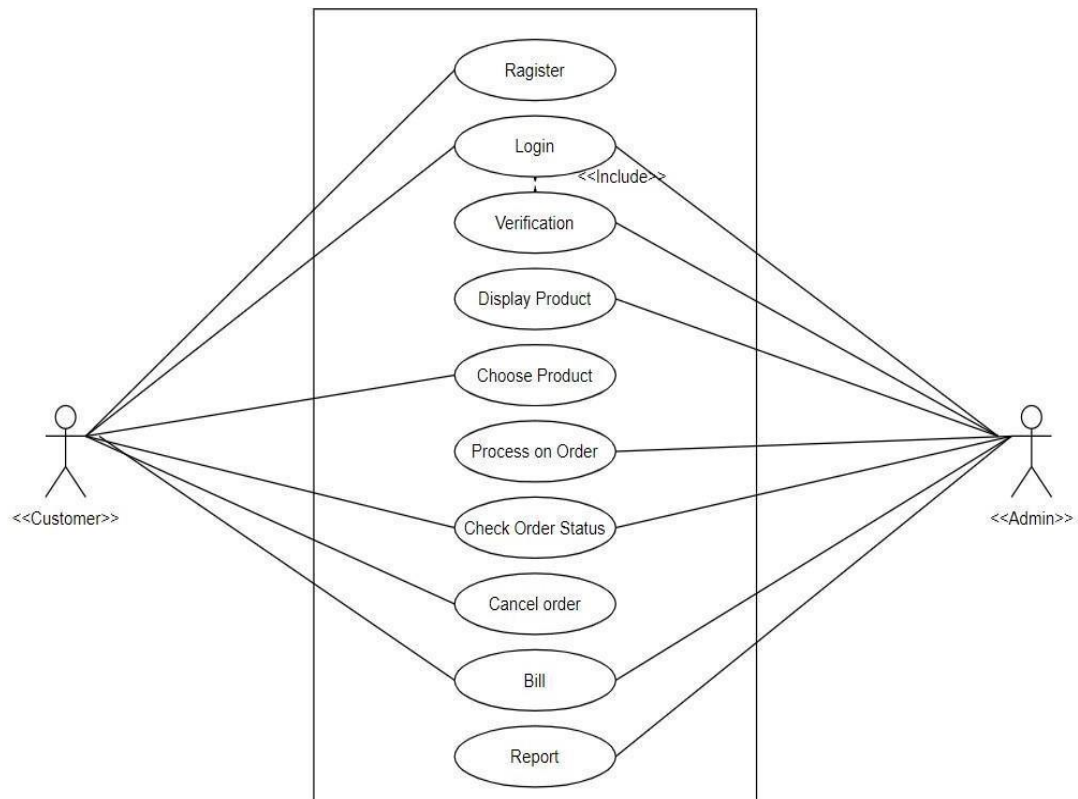


Fig No: 5.4.1 Use case diagram

5.4.2 CLASS DIAGRAM

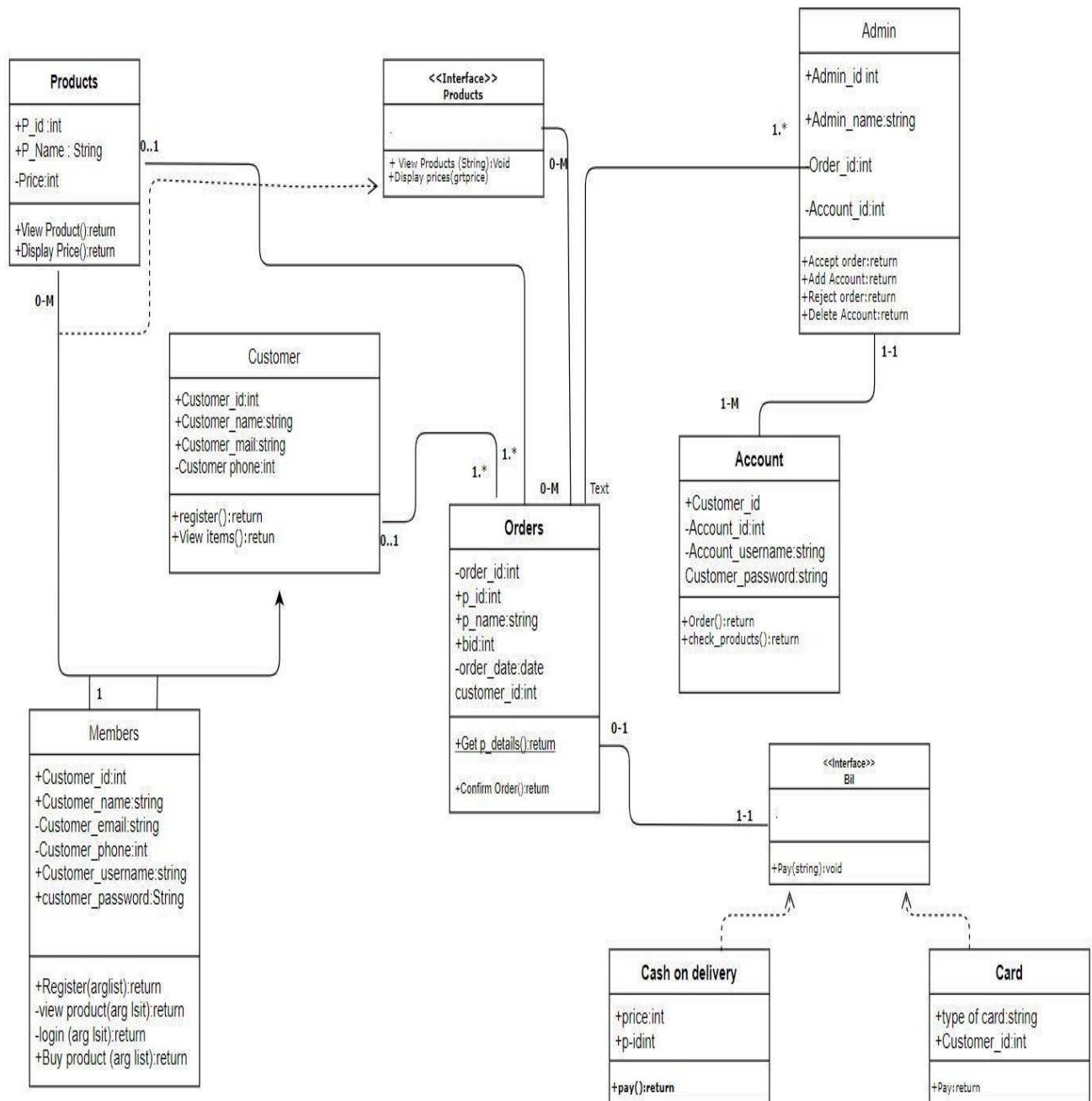


Fig No: 5.4.2 Class diagram

5.4.3 SEQUENCE DIAGRAM

A] CUSTOMER

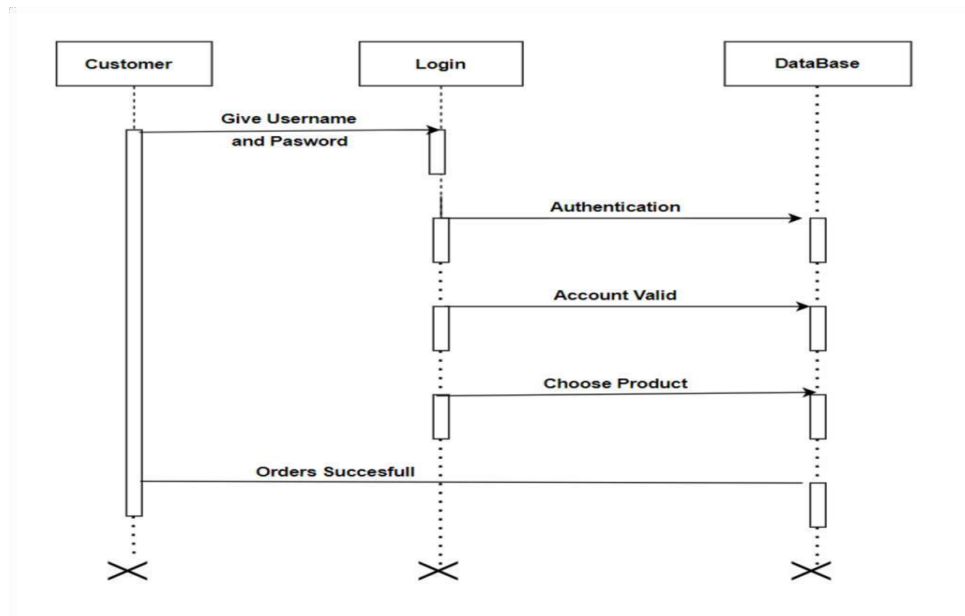


Fig No: 5.4.3 Sequence diagram

B] ADMIN

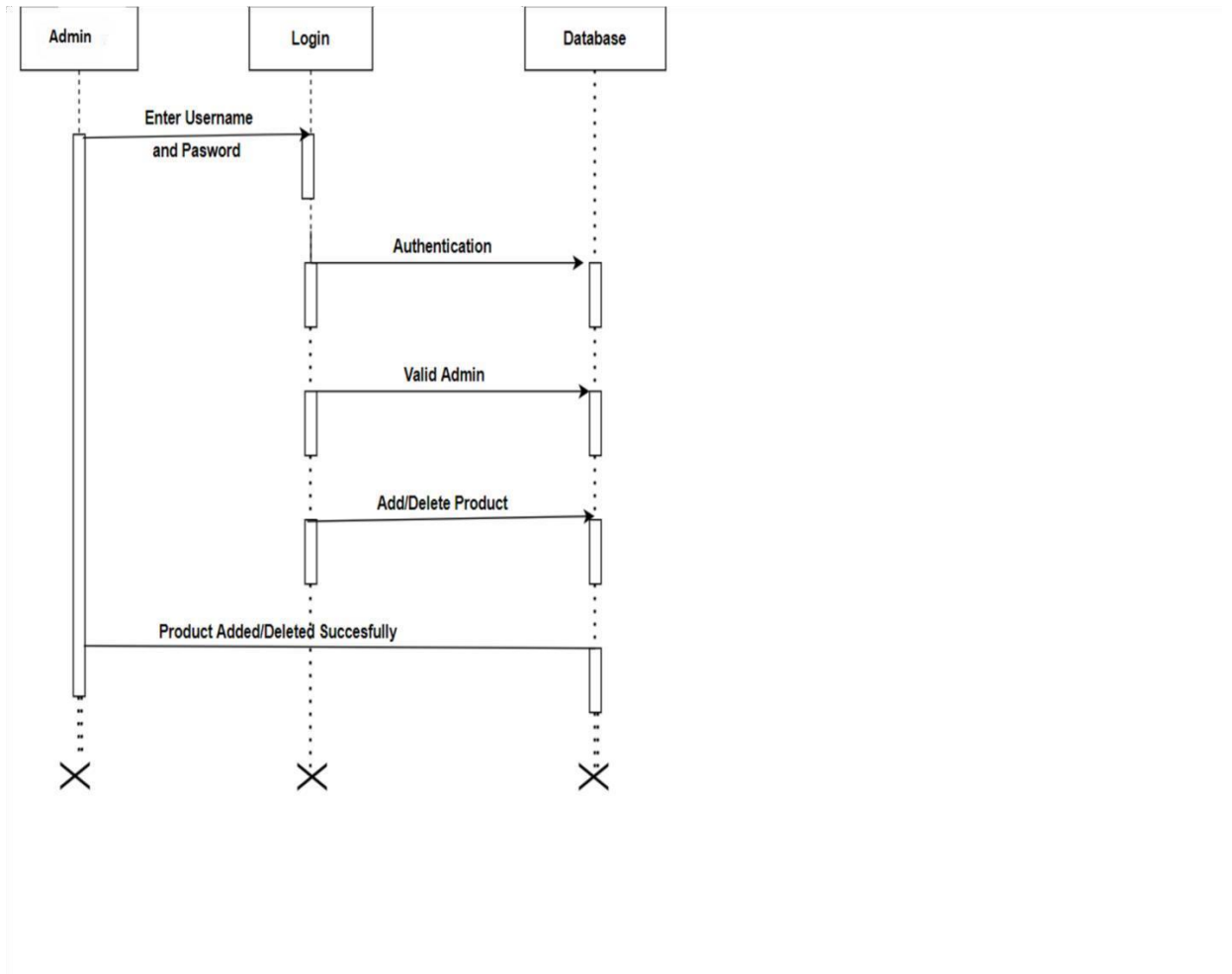


Fig No: 5.4.4 Sequence diagram

5.4.4 ACTIVITY DIAGRAM

Activity diagram is another important diagram in uml to describe the dynamic aspects of the system. It basically a flowchart to represent the flow from one activity to another activity. This flow can be sequential, branched, or concurrent.

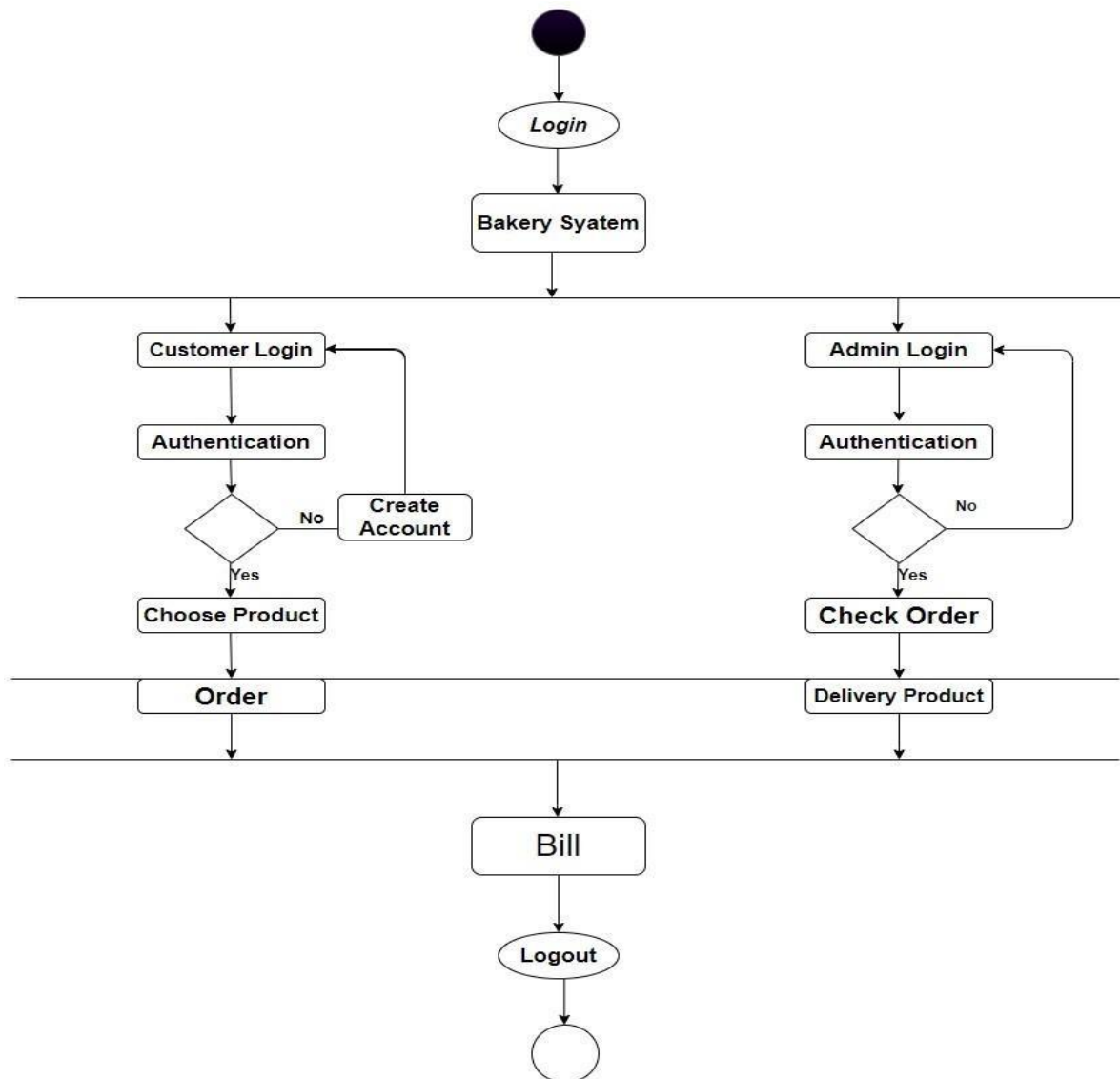


Fig No: 5.4.4 Activity diagram

5.5 DATABASE DESIGN

TABLE 1: Admin table

aid: primary key

S.NO	FIELD	TYPE	SIZE	KEY CONSTRAIN	DISCRIPTION
1	aid	int	(5)	primary	Defines the Id of the admin
2	aname	varchar	(30)	Not Null	Defines the Name of the admin
3	ausername	varchar	(30)	Not Null	Defines the username of the admin
4	apass	varchar	(30)	Not Null	Defines the password of the admin

Table No: 5.1 Admin

TABLE 2: Product table

pid: primary key

S.NO	FIELD	TYPE	SIZE	KEY CONSTRAIN	DISCRIPTION
1	pid	Int	(11)	Primary	Defines the Id of the product
2	pname	varchar	(50)	None	Defines the Name of the product
3	price	Int	(11)	None	Defines the Price of the product
4	pimage	Int	(11)	none	Defines the Image of the product

Table No: 5.2 Product

TABLE 3: User table

Uid: primary_key

S.NO	FIELD	TYPE	SIZE	KEY CONSTRAIN	DISCRIPTION
1	uid	Int	(5)	Primary	Defines the Id of the user
2	ufname	varchar	(50)	Not Null	Defines the First name of the user
3	ulname	varchar	(50)	Not Null	Defines the Last name of the user
4	uemail	varchar	(50)	Not Null	Defines the Email of the user
5	upass	varchar	(30)	Not Null	Defines the Password of the user
6	uphno	Int	(10)	Not Null	Defines the Phone no of the user
7	ugender	varchar	(10)	Not Null	Defines the gender no of the user
8	uadd	varchar	(200)	Not Null	Address of the customer
9	udob	datetime		Not Null	Date of birth of the customer

Table No: 5.3 User

TABLE 4: ORDER TABLE

oid: primary key

S.NO	FIELD	TYPE	SIZE	KEY CONSTRAIN	DISCRIPTION
1	cid	Int	(5)	primary	Defines the Order id
2	Uid	Int	(5)	Primary	Defines the User id
3	Pid	Int	(5)	Primary	Defines the Product id
4	qnty	Int	(5)	Primary	Defines the Quantity
5	date	datetime		Not null	Defines the Date
6	Status	Varchar	(20)	Not null	Defines the status
9	payment	varchar	(20)	Not null	Defines the payment mode

Table No: 5.4 order

TABLE 5: REVIEW TABLE

rid: primary key

S.NO	FIELD	TYPE	SIZE	KEY CONSTRAIN	DISCRIPTION
1	Rid	Int	(5)	primary	Defines the Id of the review
2	Uid	Int	(5)	primary	Defines the Id of the user
3	cid	Int	(5)	primary	Defines the Id of the order
4	Review	Text	(200)	Not null	Defines the Feedback
5	date	Datetime		Not null	Defines the Date
6	pid	Int	(5)	primary	Defines the Product id

Table No: 5.5 Review

CHAPTER 6

PROJECT DESCRIPTION

6.1 PROBLEM DEFINITION

The Online cake ordering System is working manually. The current system is very time consuming and costly, because it involves a lot of paper work. To manually handle such a system is a very difficult task. But now-a-days because of computerization this job is becoming easier. Following are the reasons why the current system should be computerized.

- ▶ To increase efficiency with reduced cost.

- ▶ To reduce the burden of paper work. To save the time of recording details of every work undertaken by Bakery. To check that the request for particular product is available. To generate reports easily.

6.1.1 NEEDS

- Efficiently maintains the details about the customer order .
- Simultaneously updates the changes made to any data or item in the database.
- It is faster than manual system

6.1.2 ADVANTAGE

- The system reduce manual work
- Allows for faster service
- Save time and cost
- Easy user friendly GUI

6.1.3 DISADVANTAGE

- Its reduce employment as the human efforts are being automated by this system

APPLICATION

- Reduce human error.
- To provide a better Graphical User Interface.
- To generate reports easily.

6. 2 MODULE DESCRIPTION

1.Admin module

In Admin, it has unique username and password. Administration is the only authorized person to access module, so other user doesn't get rights to access admin module. Admin maintain company and customer details.

2.Customer Module

In customer module, it has customer details such as name, phone number, email address, phoneno etc., Each customer has unique no. The customer details can also be viewed in order tracking and company module. Admin can also get customer details by using this customer unique number.

- Customer can register first
- After register user can login to proceed ahead.
- User can search product.
- User can place order and paid payment processed.
- View status of order.

CHAPTER 7

SOFTWARE TESTING AND MAINTENANCE

7.1 INTRODUCTION

Software testing is widely used technology because it is compulsory to test each and every software before deployment. As we know, software testing is a process of analyzing an application's functionality as per the customer prerequisite. If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free. Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects. Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

7.2 BLACK BOX TESTING

Another type of manual testing is black-box testing. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.

Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team. Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement. The main objective of implementing the black box testing is to specify the business needs or the customer's requirements.

7.3 UNIT TESTING

Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called unit testing. The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of a software or an application. And it is verified throughout the specified application development phase.

7.4 INTEGRATION TESTING

Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called integration testing. The purpose of executing the integration testing is to test the statement's accuracy between each module.

7.5 SYSTEM TESTING

Whenever we are done with the unit and integration testing, we can proceed with the system testing. In system testing, the test environment is parallel to the production environment. It is also known as end-to-end testing. In this type of testing, we will undergo each attribute of the software and test if the end feature works according to the business requirement. And analysis the software product as a complete system.

7. 6 WHITE BOX TESTING

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers. Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as WBT (White Box Testing). In other words, we can say that the developer will execute the complete white-box testing for the particular software and send the specific application to the testing team. The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.

7.7 ALPHA TESTING

Alpha testing is conducted in the organization and tested by a representative group of end-users at the developer's side and sometimes by an independent team of testers. Alpha testing is simulated or real operational testing at an in-house site. It comes after the unit testing, integration testing, etc. Alpha testing used after all the testing are executed. It can be a white box, or Black-box testing depends on the requirements - particular lab environment and simulation of the actual environment required for this testing.

7.8 BETA TESTING

Beta testing is a type of user acceptance testing among the most crucial testing, which performed before the release of the software. Beta Testing is a type of Field Test. This testing performs at the end of the software testing life cycle. This type of testing can be considered as external user acceptance testing. It is a type of salient testing. Real users perform this testing. This testing executed after the alpha testing. In this the new version, beta testing is released to a limited audience to check the accessibility, usability, and functionality, and more.

7.9 VALIDATION TESTING

Validation testing is testing where tester performed functional and non-functional testing. Here functional testing includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and non-functional testing includes User acceptance testing (UAT). Validation testing is also known as dynamic testing, where we are ensuring that "we have developed the product right." And it also checks that the software meets the business needs of the client.

7.12 OVERVIEW

An online cake ordering system allows the users to show the various cake products, services and the quality of the cakes provided to the customers. And also there is login facilities for Admin, Customers and to maintain their account with the cake shop. Cakes are now available in online store and the customers can purchase through online. The project consists of list of cake products displayed in various categories. The user may browse through these items as per categories. If the user likes a product he may add it to his shopping cart. And also customers can follow, Order. Here we use user friendly interface to make the entire frontend. The middle tier or code behind model is designed for fast processing. And MYSQL serves as a backend to store cake products lists data. Thus, the project brings an entire cakeshop through online and makes it easy to know more about the cakes.

7.13 TEST CASES

Initial Functional Test Cases

Use Case	Function Being Tested	Initial System State	Input	Expected Output
System startup	System is started when the switch is turned "on"	System is off	Activated the "on" switch	Project request to login
Project Startup	System accept sand leads to login page	Project is requesting Login	Enter a valid user name and password	Project is on
Project Startup	Connection to the home page is established	Project has been turned on	Enter valid data	Project output should demonstrate that a connection has been established to the project
Enter Login	System allows admin to login	System is asking to enter the user name And password	Enter login details	System displays the home page of SMS.

Login	System allows admin to login	System is asking for entry of user name password	Enter invalid login details	System displays an error message. "you are unauthorized user" system is ready to start a new login
Enter User Registration	System reads Admin and enter Register information	Project is on	Insert valid Data in Register page	System display Register and data inserted successfully

Test Case

Use Case	Function Being Tested	Initial System State	Input	Expected Output
Enter dealer registration	System reads Admin and Enter Invalid data	Project is on	Insert valid	System display register and data inserted successfully
Enter product registration	System reads Admin and enter product information	Project is On	Insert valid	System display Product Registration Available
Enter Student details	System reads Admin and Enter Invalid data	Project is on	Insert valid	System display Student available
Enter Student Information details	System reads Admin and Enter Order information	Project is on	Insert valid Data	System display Enter Student and data Received successfully

Enter Category details	System reads Admin and Enter Invalid data	Project is on	Insert valid Data	System display category and data Received
------------------------------	--	------------------	-------------------------	---

SAMPLE TEST CASE

PROJECT NAME: Online Cake ordering System

Test Case Name: Login (valid data)

PROJECT NAME: Online Cake Ordering System	
TEST CASE: LOGIN	
Test case ID: function_1	Test designed by: Divya. T
Test priority(low/med/high): med	Test designed date: 1.6.2023
Module name: login screen	Test executed by: Divya. T
Test title: verify user name and password	Test execution date: 1.6.2023
Description: Test the login page	

Pre-condition: User has to enter the valid user's name and password

STEP	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/FAIL)
1	Navigate to login page				
2	Provide valid username	Username =admin			
3	Provide valid password	Password =admin			
4	Click on login button		User should be able to login	A valid user can login into home page	Pass

Pre-condition: user has to validate the correct data with the database

Test Case Name: User Details (valid data)

PROJECT NAME: Online Cake Ordering System	
TEST CASE: Customers Details	
Test case ID: function-2	Test designed by: Divya. T
Test priority(low/med/high): med	Test designed date: 16.6.2023
Module name: Customer Details screen	Test executed by: Divya. T
Test Title: Name, email, address, phone no,	Test execution date: 16.6.2023
Description: Test the Customer Details page	

Pre-condition: User has to enter the invalid username and password

Dependencies:

Pre-condition: User has to enter the valid correct data with the database

Dependencies:

STEP	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/FAIL)
1	Navigate to Test Details page		User should be able to login	A valid user can login in to homepage	Pass
2	Provide valid Test name	Test name =diagnosis			
3	Provide valid Test type	Test type =urine			
4	Click on login button				

Post-condition: User is validated with database and successfully login to homepage.

Test Case Name: product ordering (valid data)

PROJECT NAME: Online Cake Ordering System	
TEST CASE: product ordering	
Test case ID: function_3	Test designed by: Divya. T
Test priority(low/med/high): med	Test designed date: 16.6.2023
Module name: product ordering	Test executed by: Divya. T
Test Title: product id, product name,price	Test execution date: 16.6.2023
Description: Test the product ordering	

Pre-condition: User has to enter the valid correct data with the database

Dependencies:

STEP	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/FAIL)
1	Navigate to booking page		User should be able to login	A valid user can login in to homepage	Pass
2	Provide valid username	Test id =01			
3	Provide valid password	Test date =10/06/2023			
4	Click on login button				

Post-condition: User is validated with database and successfully login to homepage.

CHAPTER 8

CONCLUSION

The researchers create a Cake Ordering System for Cake Shop businesses. The developed system was made available for input to its target users. According to the study's findings, the built system meets the demands and needs of the respondents and intended users. The majority of project respondents praised the system's usefulness and dependability in quickening the buying and selling of cakes online. As a result, the researchers came to the conclusion that the devised method is a useful tool for cake businesses. A cake ordering system is a computerized system that enables customers to place orders for cakes online. This system can be used by bakeries to take orders from customers and manage their inventory. The system can also be used by customers to track their orders and get updates on the status of their order.

The benefits of a cake ordering system include increased efficiency and customer satisfaction. With a cake ordering system, customers can easily place orders online and track the status of their order. This system can also help bakeries manage their inventory and improve their business operations . The challenges faced during the development of a cake ordering system include designing a user-friendly interface and creating an efficient database structure. Creating a user-friendly interface is important because it ensures that customers can easily use the system. Creating an efficient database structure is necessary to ensure that the system can handle high volumes of orders.

Overall, cake ordering systems are beneficial for both customers and bakeries. They help improve business efficiency and customer satisfaction. However, designing a user-friendly interface and creating an efficient database structure are important challenges that must be overcome. If these challenges are successfully addressed, then the cake ordering system will be a success.

CHAPTER 9

FURTHER ENCHANCEMENT

In this project “Online cake ordering System” helps customers to place orders through online Also eases the workload on the staff of cake shop. This system will make things easier for staff as whole ordering process is done by customer only. As most of the things will be performed online, It will reduce the usage of paper for the cake shop. It would be much more comfortable for the customers to have an online cake order. It would be hassle free for users as they can select the cake they want and make payment for it. Also it will reduce the purchasing time for customers.

CHAPTER 10

APPENDICES

SAMPLE CODE

Let's Move ahead with this amazing project.

1. Open the terminal from the folder where we want to create the project.
Right-click on mouse -> open in terminal -> write “code .” (space is mandatory between code and “.”)

2. Then go to the project folder -> urls.py and inside urls.py of project, do this -> add “include” in import as shown in the code below and add “path(“”, include(“app.urls”))”

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include("app.urls"))
]
```

3. Create urls.py in the app of the Online Cake Ordering System project (urls.py is mandatory in both project and app).

4. In setting.py, add the ‘app name’. In my case, it is an app only, as below.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app'
]
```

5. Now run server to check if everything is working or not. If you see the below image, then we are done with the installation part. To runserver, run the command in the terminal as follows “py manage.py runserver”.(if you see the rocket image, then it’s working fine).
6. Now, create the models.py using the ORM technique as follows.

```
from django.db import models

from django.contrib.auth.models import User

# Create your models here.

class Cake(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    image = models.ImageField(upload_to='cakes')
    description = models.TextField()

    def __str__(self):
        return self.name

class CartItem(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    cake = models.ForeignKey(Cake, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)

    def __str__(self):
        return f'{self.user.username}'s CartItem - {self.cake.name}"
```

To create the above field in a database, run the following commands as follows:

Py manage.py makemigrations

Py manage.py migrate

7. To insert data in the database

templates/app -> home.html

```
<form method="POST" action="{% url 'caking-ordering' %}">
    {% csrf_token %}

    <div class="search-bar">

        <input type="text" name="search_query" placeholder="Search..." value="{ {
            request.POST.search_query } }">
```

```

<button class="search-button" type="submit">Search a Cake</button>
</div>
</form>
<div class="product-row">
    {% for cake in cakes %}
    <div class="product-card">
        
        <h3>{{ cake.name }}</h3>
        <p>{{ cake.description }}</p>
        <form action="{% url 'add_to_cart' pk=cake.pk %}" method="post">
            {% csrf_token %}
            <button class="yellow-button">Add to Cart</button>
        </form>
    </div>
    </div>

```

Views.py

```

@login_required(login_url="login_view")
def cake_ordering_view(request):
    search_query = request.POST.get('search_query')
    if search_query:
        filtered_cakes = Cake.objects.filter(Q(name__icontains=search_query))
        context = {'cakes': filtered_cakes, 'search_query': search_query}
    else:
        cakes = Cake.objects.all()
        context = {'cakes': cakes}
    return render(request, 'app/home.html', context)

```

Urls.py

```

path("caking-ordering", cake_ordering_view, name="caking-ordering"),

```

8. Show the Cart Data

template/app -> Cart.html

```

<div class="cart-items">

```

```
{% for cart_item in cart_items %}

<div class="cart-item">

{% if request.user == cart_item.user %}

<h3>{{ cart_item.cake.name }}</h3>

<p>{{ cart_item.cake.description }}</p>

<div class="cart-buttons">

<select class="payment-dropdown">

<option value="">-----Select Payment-----</option>

<option value="">Online Payment</option>

<option value="">Cash on Delivery</option>

</select>

<a class="yellow-button" href="{% url 'remove_from_cart' pk=cart_item.pk %}">Remove
from Cart</a>

</div>

{% empty %}

<p>Your cart is empty.</p>

{% endfor %}

</div>
```

Views.py

```
@login_required(login_url="login_view")

def add_to_cart(request, pk):

if request.method == 'POST':

try:

cake = Cake.objects.get(pk=pk)

except Cake.DoesNotExist:

return HttpResponseRedirect("Invalid cake ID")

# Create a new CartItem

cart_item = CartItem.objects.create(user=request.user, cake=cake)

cart_item.save()
```

```

return redirect('cart_view')

return HttpResponseRedirect("Invalid request")

@login_required(login_url="login_view")

def cart_view(request):
    cart_items = CartItem.objects.filter(user=request.user)
    return render(request, 'app/cart.html', {'cart_items': cart_items})

@login_required(login_url="login_view")

def remove_from_cart(request, pk):
    try:
        cart_item = CartItem.objects.get(pk=pk)
        cart_item.delete()
        return redirect('cart_view')
    except CartItem.DoesNotExist:
        return HttpResponseRedirect("Invalid cart item ID")

```

Urls.py

```

path("add_to_cart/<int:pk>", add_to_cart, name="add_to_cart"),
path("cart_view", cart_view, name="cart_view"),
path("remove_from_cart/<int:pk>", remove_from_cart, name="remove_from_cart"),
path("cart", Index, name="cart"),

```

9. Login.html

```

<form action="{ % url 'login_view' % }" method="POST">
    { % csrf_token % }
    { % if messages % }
    { % for message in messages % }
    <p class="error-message">{ { message } }</p>
    { % endfor % }
    { % endif % }
    <div class="form-group">
    <label for="username">username:</label>

```

```

<input type="text" id="username" name="username" required>
</div>
<div class="form-group">
<label for="password">Password:</label>
<input type="password" id="password" name="password" required>
</div>
<input type="submit" value="Login"><br><br>
You can now login here: <a href="{ % url 'register' % }">Register here</a>
</form>

```

Views.py

```

def login_view(request):
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect("caking-ordering")
        else:
            messages.error(request, "Invalid email or password.")
            return render(request, "app/login.html")

```

Urls.py

```

path("login_view", login_view, name="login_view"),

```

10. Register.html

```

<form action="{ % url 'register' % }" method="POST">
    { % csrf_token % }
    { % if messages % }
    { % for message in messages % }
    <p class="error-message">{{ message }}</p>

```

```

{% endfor %}

{% endif %}

<div class="form-group">

<label for="username">Username:</label>

<input type="text" id="username" name="username"
value="{ {request.POST.username} }" required>

</div>

<div class="form-group">

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

</div>

<div class="form-group">

<label for="password-confirm">Confirm Password:</label>

<input type="password" id="password-confirm" name="password-confirm" required>

</div>

<input type="submit" value="Register">

</form>

```

Views.py

```

def Register(request):
    if request.method == "POST":
        username = request.POST.get("username")
        email = request.POST.get("email")
        password = request.POST.get("password")
        confirm_password = request.POST.get("password-confirm")
        if password == confirm_password:

```



```

# Check if username or email already exists
if User.objects.filter(username=username).exists():
    messages.error(request, "Username already exists.")
elif User.objects.filter(email=email).exists():
    messages.error(request, "Email already exists.")
else:
    # Create new user
    user = User.objects.create_user(
        username=username, email=email, password=password
    )
    messages.success(request, "Registration successful. You can now login.")
    return redirect("login_view")
else:
    messages.error(request, "Passwords do not match.")
    return render(request, "app/register.html")

```

Urls.py

```
path("", Register, name="register"),
```

11. For Logout Process

Views.py

```

@login_required(login_url="login_view")
def logout_view(request):
    logout(request)
    return redirect('login_view')

```

Urls.py For Above:

```

from django.urls import path
from .views import *
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [

```

```

path("cart", Index, name="cart"),
path("", Register, name="register"),
path("login_view", login_view, name="login_view"),
path("caking-ordering", cake_ordering_view, name="caking-ordering"),
path("add_to_cart/<int:pk>", add_to_cart, name="add_to_cart"),
path("cart_view", cart_view, name="cart_view"),
path("remove_from_cart/<int:pk>", remove_from_cart, name="remove_from_cart"),
path("logout_view", logout_view, name="logout_view")
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Explanation for the above code:

The provided code consists of Django's views for an online cake ordering system. Let's understand how the views handle various functionalities of the system and interact with HTML templates.

- 1. Index View:** The Index view renders the “cart.html” template. This serves as the entry point for the cart page.
- 2. Register View:** The Register view handles user registration. When a user submits the registration form, the view checks if the provided username and email are unique. If the registration is successful, the user is created, and appropriate success messages are displayed. If there are errors (e.g., duplicate username or email), the view shows corresponding error messages using Django's messages framework.
- 3. Login View:** The login_view handles user login. When the user submits the login form, the view authenticates the user's credentials using Django's built-in authenticate method. If successful, the user is logged in and redirected to the “cake_ordering_view.” If login fails, an error message is shown.
- 4. Cake Ordering View:** The cake_ordering_view displays available cakes to users. It supports searching for specific cakes based on the “search_query” parameter. If a search query is submitted, the view filters and displays cakes accordingly. If no search query is provided, it shows all cakes.
- 5. Add to Cart View:** The add_to_cart view allows users to add cakes to their shopping carts. When a user selects a cake to add, a new CartItem object is created, linking the logged-in user and the selected cake. The user is then redirected to the “cart_view.”
- 6. Cart View:** The cart_view displays the shopping cart items for the logged-in user. It fetches all CartItem objects associated with the user and renders them in the “cart.html” template.

7. Remove from Cart View: The `remove_from_cart` view enables users to remove items from their shopping cart. When a user chooses to remove an item, the corresponding `CartItem` object is deleted, and the user is redirected back to the cart view.

8. Logout View: The `logout_view` handles user logout. When a user clicks on the logout link, the view logs out the user and redirects them to the login view.

The views interact with Django's templating engine to render HTML templates. The HTML templates are responsible for presenting data and providing user interfaces for different functionalities.

templates/app -> home.html

```
<!DOCTYPE html>

<html>

<head>

<title>PythonGeeks Cake Ordering Portal</title>

<style>
body {
font-family: Arial, sans-serif;
background-color: #f1f1f1;
margin: 0;
padding: 0;
}
.container {
max-width: 800px;
margin: 0 auto;
padding: 20px;
background-color: #fff;
border: 1px solid #ccc;
border-radius: 5px;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
h2 {
```

```
text-align: center;
color: #333;
}

.welcome {
text-align: center;
margin-bottom: 20px;
}

.welcome p {
font-size: 18px;
}

.options-bar {
display: flex;
justify-content: space-between;
align-items: center;
margin-bottom: 20px;
}

.cart-logo {
width: 40px;
height: 30px;
background-color: #4CAF50;
border: none;
border-radius: 4px;
color: white;
cursor: pointer;
font-size: 16px;
}

.logout-button {
background-color: red;
border: none;
```

```
color: white;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
}

.search-bar {
text-align: center;
margin-bottom: 20px;
}

.search-bar input[type="text"] {
width: 300px;
padding: 8px;
font-size: 16px;
border: 1px solid #ccc;
border-radius: 4px;
}

.search-button {
background-color: #4CAF50;
color: #fff;
border: none;
padding: 8px 20px;
font-size: 16px;
cursor: pointer;
border-radius: 4px;
}
```

```
.product-row {  
  display: flex;  
  justify-content: space-between;  
  margin-bottom: 20px;  
}  
  
.product-card {  
  width: calc(50% - 10px);  
  margin: 0;  
  padding: 10px;  
  background-color: #f9f9f9;  
  border: 1px solid #ccc;  
  border-radius: 4px;  
}  
  
.product-card img {  
  width: 100%;  
  height: 150px;  
  object-fit: cover;  
  border-radius: 4px;  
}  
  
.product-card h3 {  
  text-align: center;  
  margin-top: 10px;  
}  
  
.product-card p {  
  text-align: center;  
  margin-top: 5px;  
}  
  
.green-button {  
  background-color: #4CAF50;
```

```

border: none;
color: white;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
}

.yellow-button {
background-color: #FFD700;
border: none;
color: black;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
}

</style>
</head>
<body>
<div class="container">
<h2>PythonGeeks Cake Ordering Portal</h2>
<div class="welcome">
<p>Welcome, <span id="username">{{ user.username }}</span></p>

```

```

</div>

<div class="options-bar">

<div>

<form action="{ % url 'cart_view' % }">

{ % csrf_token % }

<button class="cart-logo">Cart</button>

</form>

</div>

<div>

<a href="{ % url 'logout_view' % }">

<button class="logout-button">Logout</button>

</a>

</div>

</div>

<form method="POST" action="{ % url 'caking-ordering' % }">

{ % csrf_token % }

<div class="search-bar">

<input type="text" name="search_query" placeholder="Search..." value="{ {
request.POST.search_query } }">

<button class="search-button" type="submit">Search a Cake</button>

</div>

</form>

<div class="product-row">

{ % for cake in cakes % }

<div class="product-card">



<h3>{ { cake.name } }</h3>

<p>{ { cake.description } }</p>

<p>Buy for ₹{ { cake.price } }</p>

```



```

<form action="{ % url 'add_to_cart' pk=cake.pk % }" method="post">
  { % csrf_token % }
  <button class="yellow-button">Add to Cart</button>
</form>
</div>
{ % endfor % }
</div>
</div>
</body>
</html>

```

templates/app -> login.html

```

<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
<style>
body {
font-family: Arial, sans-serif;
background-color: #f1f1f1;
}
.container {
max-width: 400px;
margin: 0 auto;
padding: 20px;
background-color: #fff;
border: 1px solid #ccc;
border-radius: 5px;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
margin-top: 10%;

```

```
}  
h2 {  
  text-align: center;  
  color: #333;  
}  
.form-group {  
  margin-bottom: 20px;  
}  
label {  
  display: block;  
  font-weight: bold;  
  margin-bottom: 5px;  
}  
input[type="text"],  
input[type="password"] {  
  width: 96%;  
  padding: 8px;  
  font-size: 16px;  
  border: 1px solid #ccc;  
  border-radius: 4px;  
}  
input[type="submit"] {  
  background-color: #4CAF50;  
  color: #fff;  
  border: none;  
  padding: 10px 20px;  
  font-size: 16px;  
  cursor: pointer;  
  border-radius: 4px;
```

```

}

input[type="submit"]:hover {
background-color: #45a049;
}

.error-message {
color: red;
margin-top: 5px;
}

</style>
</head>
<body>
<div class="container">
<h2>Login</h2>
<form action="{ % url 'login_view' % }" method="POST">
{ % csrf_token % }
{ % if messages % }
{ % for message in messages % }
<p class="error-message">{{ message }}</p>
{ % endfor % }
{ % endif % }
<div class="form-group">
<label for="username">username:</label>
<input type="text" id="username" name="username" required>
</div>
<div class="form-group">
<label for="password">Password:</label>
<input type="password" id="password" name="password" required>

```

```

</div>
<input type="submit" value="Login"><br><br>
You can now login here: <a href="{ % url 'register' % }">Register here</a>
</form>
</div>
</body>
</html>

```

templates/app -> register.html

```

<!DOCTYPE html>
<html>
<head>
<title>Registration Page</title>
<style>
body {
font-family: Arial, sans-serif;
background-color: #f1f1f1;
}
.container {
max-width: 400px;
margin: 0 auto;
padding: 20px;
background-color: #fff;
border: 1px solid #ccc;
border-radius: 5px;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
margin-top: 10%;
}
h2 {
text-align: center;

```

```
color: #333;
}
.form-group {
margin-bottom: 20px;
}
label {
display: block;
font-weight: bold;
margin-bottom: 5px;
}
input[type="text"],
input[type="email"],
input[type="password"] {
width: 96%;
padding: 8px;
font-size: 16px;
border: 1px solid #ccc;
border-radius: 4px;
}
input[type="submit"] {
background-color: #4CAF50;
color: #fff;
border: none;
padding: 10px 20px;
font-size: 16px;
cursor: pointer;
border-radius: 4px;
}
input[type="submit"]:hover {
```

```

background-color: #45a049;
}
.error-message {
color: red;
margin-top: 5px;
}
</style>
</head>
<body>
<div class="container">
<h2>Register</h2>
<form action="{ % url 'register' % }" method="POST">
{ % csrf_token % }
{ % if messages % }
{ % for message in messages % }
<p class="error-message">{{ message }}</p>
{ % endfor % }
{ % endif % }
<div class="form-group">
<label for="username">Username:</label>
<input type="text" id="username" name="username"
value="{{ request.POST.username }}" required>
</div>
<div class="form-group">
<label for="email">Email:</label>
<input type="email" id="email" name="email" required>
</div>
<div class="form-group">

```

```

<label for="password">Password:</label>
<input type="password" id="password" name="password" required>
</div>
<div class="form-group">
<label for="password-confirm">Confirm Password:</label>
<input type="password" id="password-confirm" name="password-confirm" required>
</div>
<input type="submit" value="Register">
</form><br><br>
You can now login here: <a href="{ % url 'login_view' % }">login here</a>
</div>
</body>
</html>

```

templates/app -> cart.html

```

<!DOCTYPE html>
<html>
<head>
<title>PythonGeeks Cake Ordering Portal</title>
<style>
body {
font-family: Arial, sans-serif;
background-color: #f1f1f1;
margin: 0;
padding: 0;
}
.container {
max-width: 800px;

```

```
margin: 0 auto;
padding: 20px;
background-color: #fff;

border: 1px solid #ccc;
border-radius: 5px;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
h2 {
text-align: center;
color: #333;
}
.cart-page {
text-align: center;
margin-bottom: 20px;
}
.cart-page h3 {
font-size: 18px;
margin-bottom: 10px;
}
.cart-page p {
font-size: 16px;
color: #777;
margin-bottom: 20px;
}
.cart-items {
display: flex;
flex-direction: column;
```



```
align-items: center;
}
.cart-item {
width: 70%;
margin-bottom: 20px;
```

```
padding: 10px;
background-color: #f9f9f9;
border: 1px solid #ccc;
border-radius: 4px;
}
```

```
.cart-item img {
width: 100%;
height: 150px;
object-fit: cover;
border-radius: 4px;
}
```

```
.cart-item h3 {
text-align: center;
margin-top: 10px;
}
```

```
.cart-item p {
text-align: center;
margin-top: 5px;
}
```

```
.cart-buttons {
display: flex;
```

```
justify-content: center;
margin-top: 20px;
}
.green-button {
background-color: #4CAF50;
```

```
border: none;
color: white;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
}
```

```
.yellow-button {
background-color: #FFD700;
border: none;
color: black;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
```

```
}  
.cart-buttons {  
display: flex;  
  
justify-content: space-between;  
margin-top: 10px;  
}  
.dropdown {  
position: relative;  
}  
.dropdown-button {  
padding: 10px 20px;  
border: none;  
border-radius: 5px;  
font-weight: bold;  
text-align: center;  
text-decoration: none;  
cursor: pointer;  
background-color: #f0f0f0;  
transition: background-color 0.3s, color 0.3s;  
}  
.dropdown-button:hover {  
background-color: #e0e0e0;  
}  
.dropdown-content {
```

```
display: none;
position: absolute;
background-color: white;
```

```
min-width: 160px;
box-shadow: 0px 8px 16px 0px rgba(0, 0, 0, 0.2);
z-index: 1;
}
```

```
.dropdown-content a {
color: black;
padding: 12px 16px;
text-decoration: none;
display: block;
}
```

```
.dropdown:hover .dropdown-content {
display: block;
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p><a href="{% url 'caking-ordering' %}">Close</a></p>
```

```
<center>
```

```
<h1>Cart Items</h1>
```

```
<p style="color: grey;">You have {{ cart_items | length }} items in Your list</p>
```

```
</center>
```

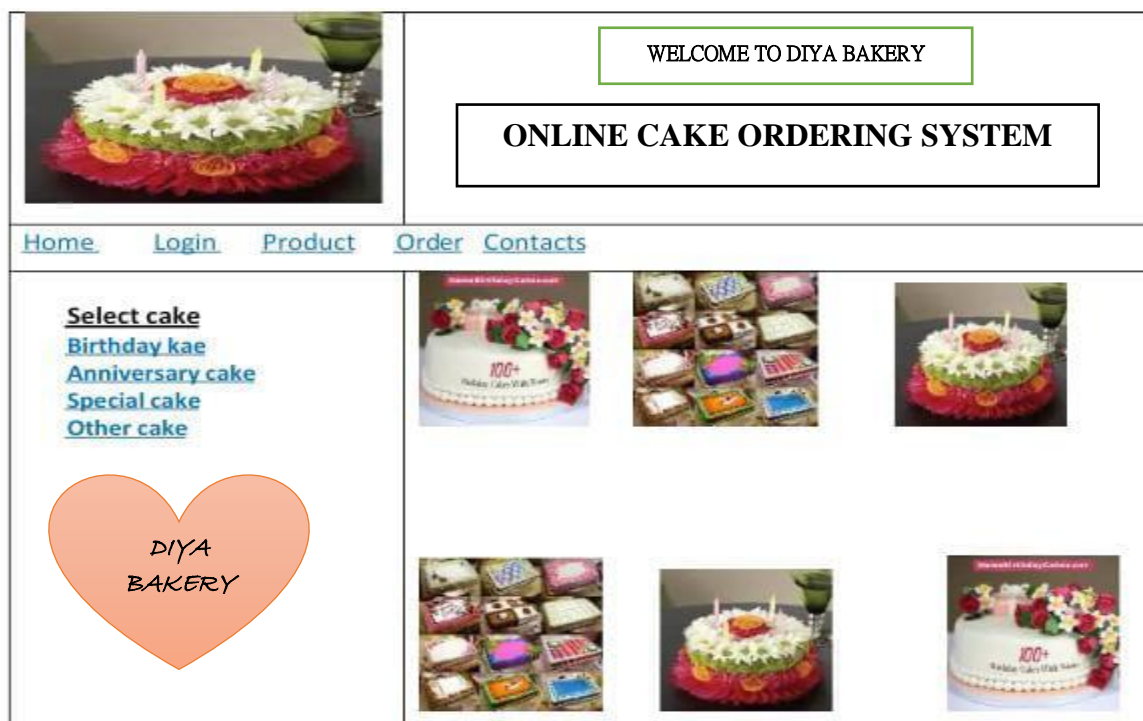
```

<div class="cart-items">
  {% for cart_item in cart_items %}
    <div class="cart-item">
      {% if request.user == cart_item.user %}
        <h3>{{ cart_item.cake.name }}</h3>
        <p>{{ cart_item.cake.description }}</p>
        <div class="cart-buttons">
          <select class="payment-dropdown">
            <option value="">-----Select Payment-----</option>
            <option value="">Online Payment</option>
            <option value="">Cash on Delivery</option>
          </select>
          <a class="yellow-button" href="{% url 'remove_from_cart' pk=cart_item.pk %}">Remove
            from Cart</a>
        </div>
      {% endif %}
    </div>
  {% empty %}
    <p>Your cart is empty.</p>
  {% endfor %}
</div>
</body>
</html>



```

10. 2 SAMPLE SCREENSHOTS


HOME PAGE



LOGIN PAGE

	<div style="border: 1px solid black; padding: 5px; display: inline-block;">ONLINE CAKE ORDERING SYSTEM</div>
Home Login Product Order Contacts	
<p>Select cake Birthday kae Anniversary cake Special cake Other cake</p> <div style="text-align: center;"><p>DIYA BAKERY</p></div>	<h2>Login</h2> <p>User id <input type="text"/></p> <p>Password <input type="password"/></p> <p>Login</p> <p>New user? register her</p>

REGISTRATION

	<div style="border: 1px solid black; padding: 5px; display: inline-block;">ONLINE CAKE ORDERING SYSTEM</div>																				
Home Login Product Order Contacts																					
<p>Select cake Birthday cake Anniversary cake Special cake Other cake</p> <div data-bbox="280 1310 526 1514" style="text-align: center; margin-top: 20px;"><p>DIYA BAKERY</p></div>	<h2 style="text-align: center;">Registration</h2> <table><tr><td>cid</td><td><input type="text"/></td></tr><tr><td>cname</td><td><input type="text"/></td></tr><tr><td>gender</td><td><input type="text"/></td></tr><tr><td>email</td><td><input type="text"/></td></tr><tr><td>contact_no</td><td><input type="text"/></td></tr><tr><td>city</td><td><input type="text"/></td></tr><tr><td>address</td><td><input type="text"/></td></tr><tr><td>username</td><td><input type="text"/></td></tr><tr><td>password</td><td><input type="password"/></td></tr><tr><td>dob</td><td><input type="text"/></td></tr></table> <div style="text-align: center; margin-top: 20px;">click Registr</div>	cid	<input type="text"/>	cname	<input type="text"/>	gender	<input type="text"/>	email	<input type="text"/>	contact_no	<input type="text"/>	city	<input type="text"/>	address	<input type="text"/>	username	<input type="text"/>	password	<input type="password"/>	dob	<input type="text"/>
cid	<input type="text"/>																				
cname	<input type="text"/>																				
gender	<input type="text"/>																				
email	<input type="text"/>																				
contact_no	<input type="text"/>																				
city	<input type="text"/>																				
address	<input type="text"/>																				
username	<input type="text"/>																				
password	<input type="password"/>																				
dob	<input type="text"/>																				

PRODUCT DETAILS

	<div style="border: 1px solid black; padding: 10px; display: inline-block;"> ONLINE CAKE ORDERING SYSTEM </div>	
Home Login Product Order Contacts		
<p>Select cake</p> <p>Birthday kae</p> <p>Anniversary cake</p> <p>Special cake</p> <p>Other cake</p> <p>Search</p>	  	
<p>Sort by price</p> <p>From <input type="text"/></p> <p>To <input type="text"/></p> <p>Search</p>	  	
<p>View cart</p> <p>Logout</p>		

CART

	<div style="border: 1px solid black; padding: 10px; display: inline-block;"> ONLINE CAKE ORDERING SYSTEM </div>						
Home Login Product Order Contacts							
	<table border="1" data-bbox="646 919 1271 1073"> <thead> <tr> <th>Pmane</th> <th>peast</th> </tr> </thead> <tbody> <tr> <td>Sinfull mouton cake</td> <td>4000</td> </tr> <tr> <td>2 tier chocolate cake</td> <td>3000</td> </tr> </tbody> </table> <p data-bbox="867 1157 1040 1188" style="text-align: center;">Total Rs – 7000</p> <div data-bbox="812 1205 1075 1268" style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; display: inline-block;"> Conform order </div> </div>	Pmane	peast	Sinfull mouton cake	4000	2 tier chocolate cake	3000
Pmane	peast						
Sinfull mouton cake	4000						
2 tier chocolate cake	3000						

CHAPTER 11

REFERENCES

- [1]. <https://github.com/shabu/Online-cake order-System Reporting-System>
- [2]. <https://cakeshopprojects.com/online-cakeorder-system-reporting-systemphp/#:~:text=It%20an%20%20online%20cakeorder,and%20login%20using%20registerd%20details.&text=The%20system%20allows%20admin%20t0,email%20it20%%20intende d%20customer.>
- [3]. https://www.academia.edu/43044164/ONLINE_CAKE_ORDERING_SYSTEM
- [4]. <https://pdfdemo.com/online-cakeordering-sysytem-pdf-free.html>
- [5]. <https://m.facebook.com/finalyearprojects.info/photos/a.156789016585/34421798940098/?type=3>

----- THE END -----

CHAPTER 11

REFERENCES

- [1]. <https://github.com/shihabhub/Online-Diagnostic-Lab> Reporting-System
- [2]. <https://nevonprojects.com/online-diagnosticlab-reporting-systemphp/#:~:text=It%20is%20an%20online%20diagnostic, and%20login%20using%20registered%20details. &text=The%20system%20allows%20ad min%20to, email%20it%20to%20intended%20 patient.>
- [3]. [https://www.academia.edu/43044163/ONLINE _DIAGNOSTIC_LAB_REPORTING_SYSTEM](https://www.academia.edu/43044163/ONLINE_DIAGNOSTIC_LAB_REPORTING_SYSTEM)
- [4]. <https://pdfcoffee.com/online-diagnostic-labreporting-system-pdf-free.html>
- [5]. <https://m.facebook.com/finalyearproject.info/photos/a.156786468016585/344217985940098/? type=3>