ONLINE PIZZA ORDERING SYSTEM

A PROJECT REPORT
Submitted By

VIDHI SHARMA
(University Roll No.2100290140144)
SATYAM GUBRELE
(University Roll No.2100290140118)
RASHI RUHELA
(University Roll No.2100290140112)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Dr. Vipin Kumar
Associate Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(JUNE 2023)

DECLARATION

I hereby declare that the work presented in this report entitled “Online Pizza Ordering System”, was carried out by us. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.
I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name: Vidhi Sharma
Roll No: 2100290140144
Branch: Master of Computer Applications

(Candidate Signature)

Name: Satyam Gubrele
Roll No: 2100290140118
Branch: Master of Computer Applications

(Candidate Signature)

Name: Rashi Ruhela
Roll No.: 2100290140112
Branch: Master of Computer Applications

(Candidate Signature)

CERTIFICATE

Certified that **Vidhi Sharma, Satyam Gubrele, Rashi Ruhela** have carried out the project work having "**ONLINE PIZZA ORDERING SYSTEM**" for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.
Date:

Vidhi Sharma (2100290140144)

Satyam Gubrele(2100290140118)

Rashi Ruhela (2100290140112)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Dr. Vipin Kumar

(Associate Professor)

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner **Signature of External Examiner**

Dr. Arun K. Tripathi

Head, Department of Computer Applications

KIET Group of Institutions, Ghaziabad

ABSTRACT

The abstract for a Pizza Ordering Website describes the key features and benefits of the platform. It highlights its user-friendly interface, customizable menu options, and online ordering system. The platform provides customers with seamless ordering experience, allowing them to easily track and customize their orders, track their orders in real-time, and receive updates and notifications via email or text message. This website provides an efficient and reliable system.

The objective and scope of my Project Pizza Ordering System is to record the details various activities of user. It will simplify the task and reduce the paper work.

The Pizza Ordering System is an Online platform that allows customers to order pizza of their preferred choice. This system enables users to choose from a variety of pizza types, the size of the pizza, toppings and extra items. The customers can pay online and offline in both ways.

This project consists of a total of 6 modules which are User Module, Sign Up/Login, Add to Cart, Build your Pizza, Admin Module, Payment Module.

ACKNOWLEDGEMENT

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Dr. Vipin Kumar** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions. Words are not enough to express my gratitude to Dr. Arun K Tripathi Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions. Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Vidhi Sharma (2100290140144)

Satyam Gubrele (2100290140118)

Rashi Ruhela (2100290140112)

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.PROJECT DESCRIPTION:

The pizza ordering system has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and in some cases reduce the hardships faced by the existing system. Moreover, this system is designed for the particular need of the company to carry out operations in smooth and effective manner.

The application is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for the user to use this system. Thus, it proves to be user-friendly. Pizza ordering system can lead to error free, secure, reliable and fast management system. It can assist the user to concentrate on their activities rather than the record keeping. Thus, it will help the organization in better utilization of resources.

Every organization, whether big or small, has challenges to overcome and managing the information of online order, pizza, payment. Every pizza ordering system has different pizza needs therefore we design exclusive pizza ordering and management systems that are adapted to your managerial requirements. This system will ultimately allow you to better manage resources

1.1. EXISTING SYSTEM:

There are many different web based ordering systems. Those systems do not offer all the functionality that is needed for a pizza. Ordering systems usually allow people to add products and separate them in different categories and subcategories. Pizza ordering system sells pizzas, and most of them also offer clients to customize their own pizzas by picking their own ingredients.

1.2. PROJECT SCOPE:

Our project deals with ordering pizzas online. It provides an interactive interface where users can pick or choose the pizza of their choices.

1.3. MODULES:

There are total six modules in this project which are as follows :

1.3.1. User Module :

User module provide accessibility to user so they can manage their profile and use other modules. He /She can signup or login to the website.

1.3.2. Admin Module :

Custom pizzas are made by selecting several ingredients, the list of available ingredients and their prices are administrated by an employee, for instance the manager. Administration includes added, editing and deleting ingredients. Besides the administration of ingredients, the non-pizza side orders must also be administrated. Administrators also need to have logs of previous orders.

1.3.3. Build your Pizza :

The customer will get an option to select the pizza crust's size which will be small, medium and large .Then the customer has to choose the sauce to be used on his pizza crust. These can be marinara cheese, ranch and others. At last the system shows the customers different types of toppings.

1.3.4. Login/Sign-Up:

The user who is already registered can login to system by using their login id and password. The user who is not registered can create his or her account, option of sign-up is available.

1.3.5. Add to Cart:

This feature is useful in a situation where you have to order more than one pizza. Suppose you have selected a Margherita pizza and now you want to select another pizza as well, then you just have to add that pizza to the cart using the Add to cart option. Items added to the cart will be saved so that you can choose other pizzas as well.

1.3.6. Payment Module:

Payment module provide an interactive interface and integration to website so user can pay online and all the access of the transaction control by admin.

1.4. Hardware and Software Requirements:

There are some hardware software requirements below:

1.4.1. Software:

- Operating System (windows)
- Code Editor: VS Code
- Front End: HTML, CSS, JavaScript, React Js , MongoDB, NodeJS

1.4.2. Hardware:

- Intel i3 or Above
- 4GB RAM or Above
- Hard-Drive Capacity 64GB or Above

CHAPTER 2

FEASIBILITY STUDY

2.1 FEASIBILITY STUDY:

This project will be developed on computer, so first check whether the technology is technically available or not. Now a day's computer interaction with any job becomes common for any kind of job or work. And because of increasing usage of Computer, Computer is also available with a variety of hardware. Users can fulfil any type of hardware requirement.

Preliminary investigation of a system examines the feasibility of a system that is useful to an organization. It is the first phase of system development.

The main objective of this phase is to identify the current deficiencies in the user's environment and to determine which existing problem are going to be solve in proposed system and also which new function needs to be added in proposed system.

An important outcome of such preliminary investigation is to determine whether the system will meet all needed requirements.

Thus, three tests are carried out on the system namely operation, technical and economical.

2.2 OPERATIONAL FEASIBILITY :

Any project is beneficial if and only satisfies the organization's requirement. For any new system setup, it only meets to be communicated and work the other supporting system.

The new system meets all existing operations since it provides right information at a right time to the right user. A Leigh man can easily operate with the system.

2.3 TECHNICAL FEASIBILITY :

Technical Feasibility examines whether the technology needed is available and if it is available then it feasible to carry out all project activities. The technical needs of a system include:

- The facility to produce outputs in a given time.
- Ability to process large number of transaction at a particular speed.
- Giving response to users under certain conditions.

The technology needed for our system is mainly:

- Latest version of browsers.
- Any operating system.

These technologies are available which helps to carry out the system efficiently.

2.4 ECONOMICAL FEASIBLITY :

Economical feasibility of a system examines whether the finance is available for implementing the new system and whether the money spent is recoverable the satisfaction. The cost involves is in designing and developing a good investment for the organization. Thus, hardware requirements used for proposed system are very standard. Moreover, by making use of proposed system to carry out the work speedily will increase and also saves the valuable time of an organization.

In the proposed system the finance is highly required for the installation of the software's which can also be recovered by implementing a better system.

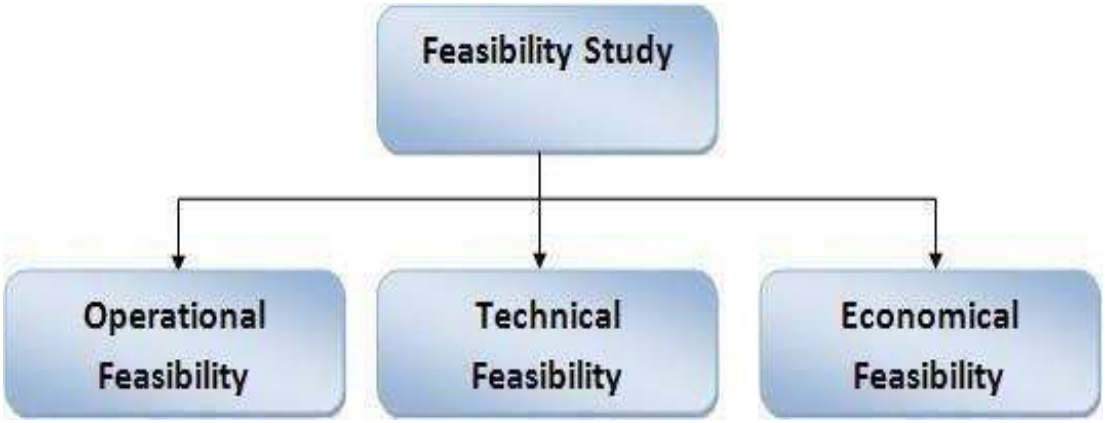

Figure 2.1 : Feasibility study

CHAPTER 3

DESIGN

3.1. WORKFLOW DIAGRAM FOR WEB APPLICATION :

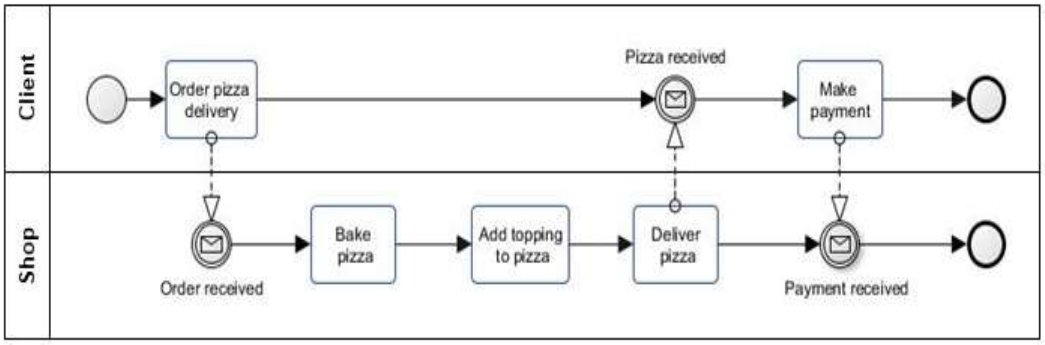

Figure 3.1 Work Flow Diagram

3.2. ER-Model / Diagrams :

The entity-relationship (ER) data model allows us to describe the data involved in a real-world enterprise term of objects and their relationships and is widely used to develop an initial database.

The ER model is important primarily for its role in database design. It provides useful concepts that allow us to move from an informal description of what users want from their database to a more detailed and precise, description that can be implemented in a DBMS. We note that many variations of ER diagrams are in use, and no widely accepted standards prevail.

The database design process can be divided into six steps. The ER model is most relevant to the first three steps:

3.3. Requirements Analysis:

The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance.

3.3.1. Conceptual Database Design:

The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high-level data model, and is discussed in the rest of this chapter.

3.3.2. Logical Database Design:

We must choose a DBMS to implement our database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS. We will only consider relational DBMS, and therefore, the task in the logical design step is to convert an ER schema into a relational database schema. The result is a conceptual schema, sometimes called the logical schema, in the relational data model.

3.3.3. Beyond the ER model:

ER model is sometimes regarded as a complete approach to designing a logical database schema. This is incorrect because the ER diagram is just an approximate description of the data, constructed through a very subjective evaluation of the information collected during requirements analysis.

3.3.4. Schema Refinement:

The fourth step in database design is to analyse the collection of relations in our relational database schema to identify potential problems, and to refine it.

3.3.5. Physical Database Design:

In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired.

3.3.6. Security Design :

In this step, we identify different user groups and different roles played by various users (e.g., the development for a product, the customer support representatives, and the product manager).

3.4. Entities, Attributes and Entity sets:

An entity is an object in the real world that is distinguishable from other objects manager of the toy department, the home address of the manager of the toy department. It is often useful to identify a collection of similar entities. Such a collection is called an

entity set. Examples include the following: the Green Dragonwort toy, the toy department, and the performance criteria.

An entity is described by set of attributes. All entities in a given entity set have the same attribute; this is essentially what u has seen by similar. For each attribute associated with an entity set, we must identify a domain of possible values. A key is a minimal set of attributes whose values uniquely identify an entity in the set. There could be more than one candidate key.

3.4.1. ER Diagram :

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system.

ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

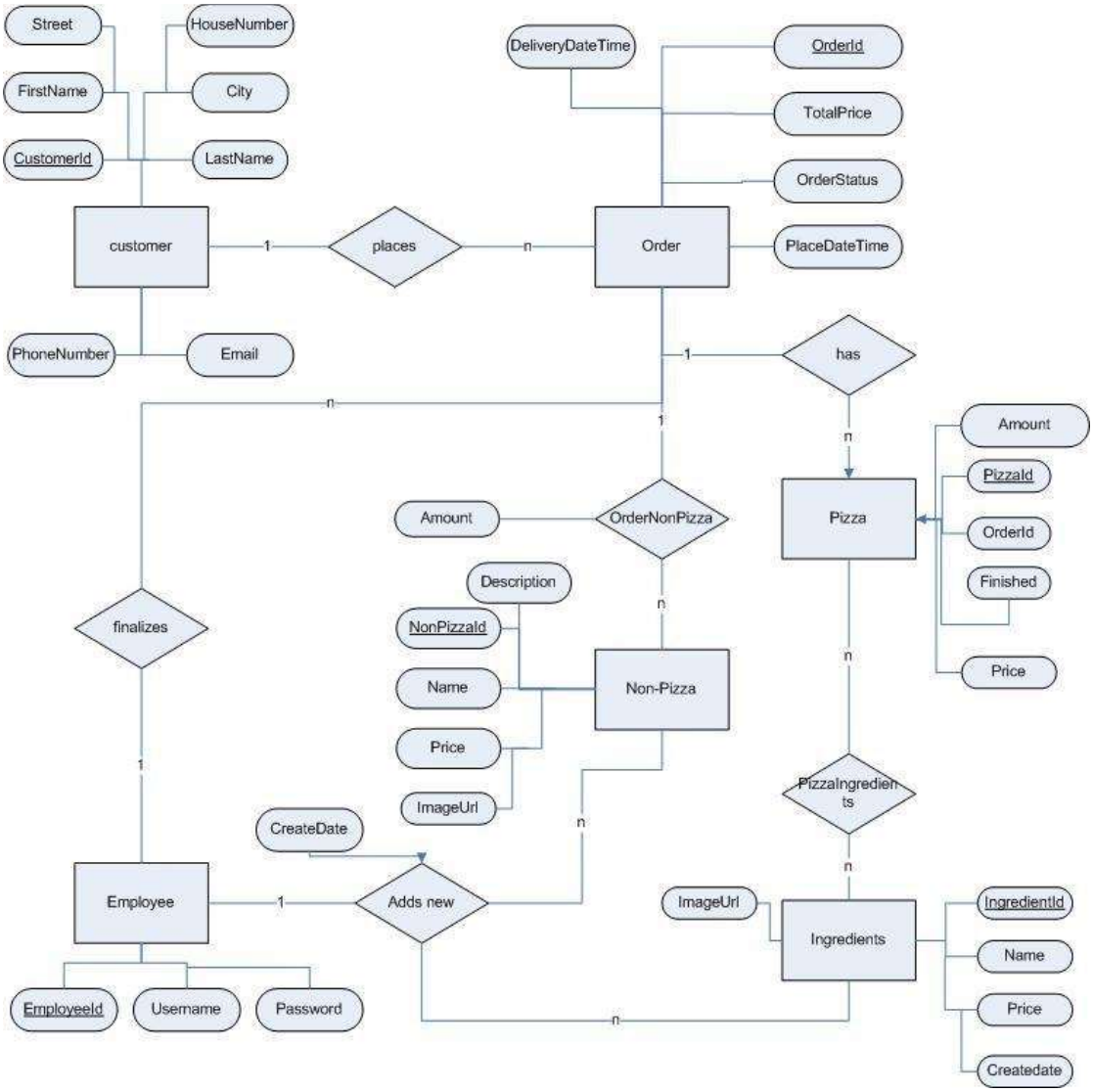

Figure 3.2 : ER Diagram

Explanation: In ER diagram , Customer can place the order with his order number. He /She has to signup first then he can login. Order has its id, date of placing the order, total price order status etc. He has to choose ingredient as admin make new pizza as ingredient has id, Name ,price, create date etc.There admin can put the image on the new pizza or topping. There are some non pizza item as well so He /She can order those items.

3.4.2. Data Flow Diagram :

Data flow diagrams (DFDs) use a number of symbols to represents systems. Most data flow modelling methods use four kinds of symbols to represent four kinds of system components: Processes, data stores, data flows and external entities (source or destination of data). The symbols that are used to represent the DFD are as follows: -

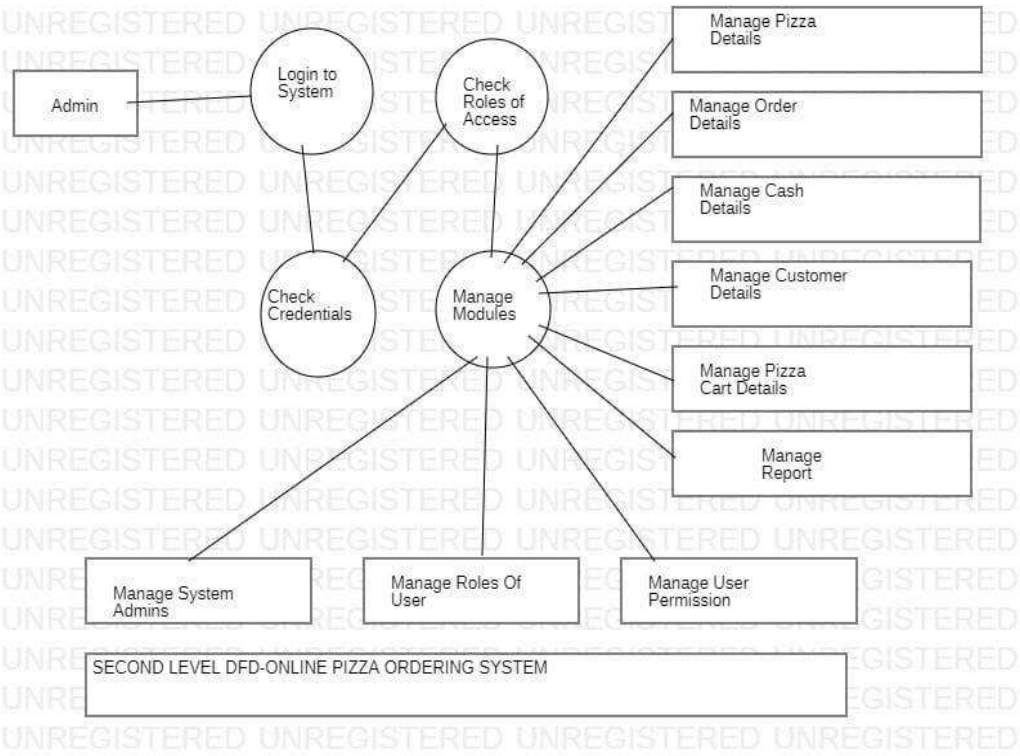

Figure 3.3 Data Flow Diagram

Explanation: In DFD diagram , Admin has different login id so website can identify the admin. In the diagram , Admin can check the roles in the restaurant and also manage his module like Building pizza, manage cart etc. This is second level DFD which shows roles of user and user as well.

3.5. UML Diagrams

A diagram is a graphical representation of a set of elements. The various diagrams in UML are as follows:

3.5.1 Class Diagram

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams address the static design view of a system. Class diagrams that include active classes address the static process view of a system. A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements on or more interfaces.

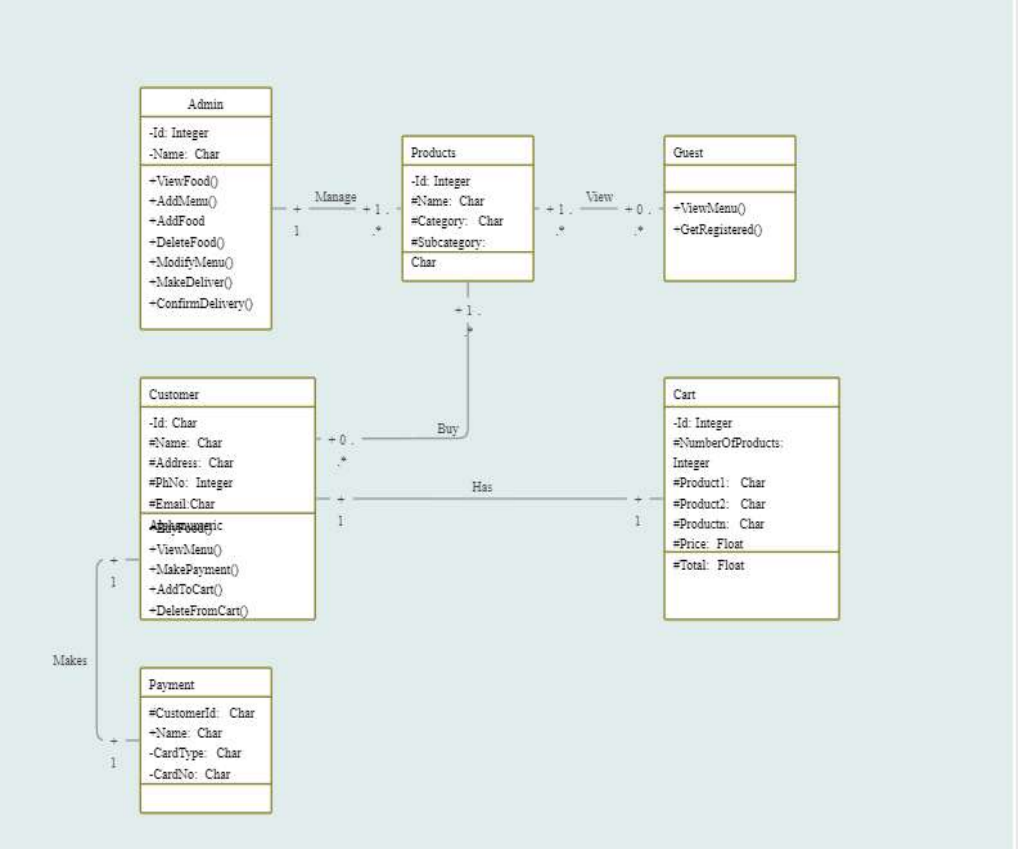

Figure 3.4 Class Diagram

Explanation: In Class diagram , There are some class which have there member function and datatype. In above Diagram Admin, cart, Payment, Customer, Guest and product have different attributes and classes. Customer hav View menu(),Make payment(), addTocart() function where guest has different functions like view menu etc. Other classes have different datatypes.

3.5.2. Use case Diagram

A use case diagram shows a set of use cases and Actors (a special kind of class) and their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modelling the behaviours of a system.

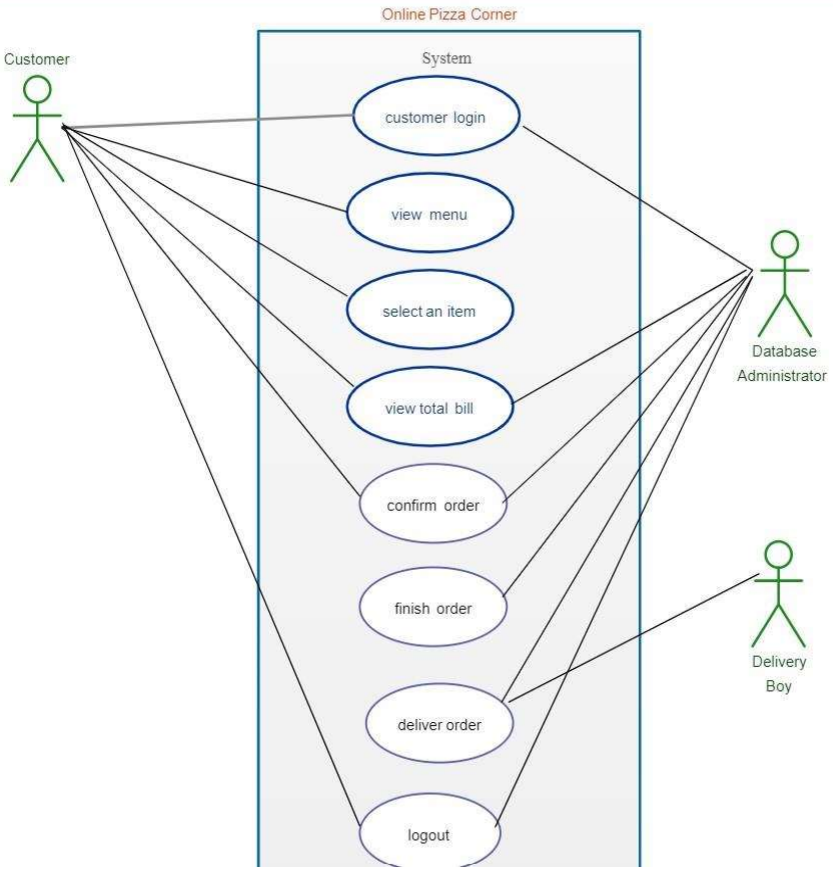

Figure 3.5 :Use case diagram

Explanation: In Use Case diagram , Customer can login or signup ,view menu, select item, confirm order ,finish order, deliver order and logout.

3.5.3. Sequence Diagram :

The sequence diagram is an interaction diagram that emphasizes the time ordering of messages for modelling a real time system. Graphically, a sequence diagram is a table that shows objects arranged along the X-axis and messages, ordered in increasing time, along the Y-axis. Sequence Diagram consists of objects, links, lifeline, focus of control, and messages.

- It has two features they are:
- This is the object life time
- There is the focus of control

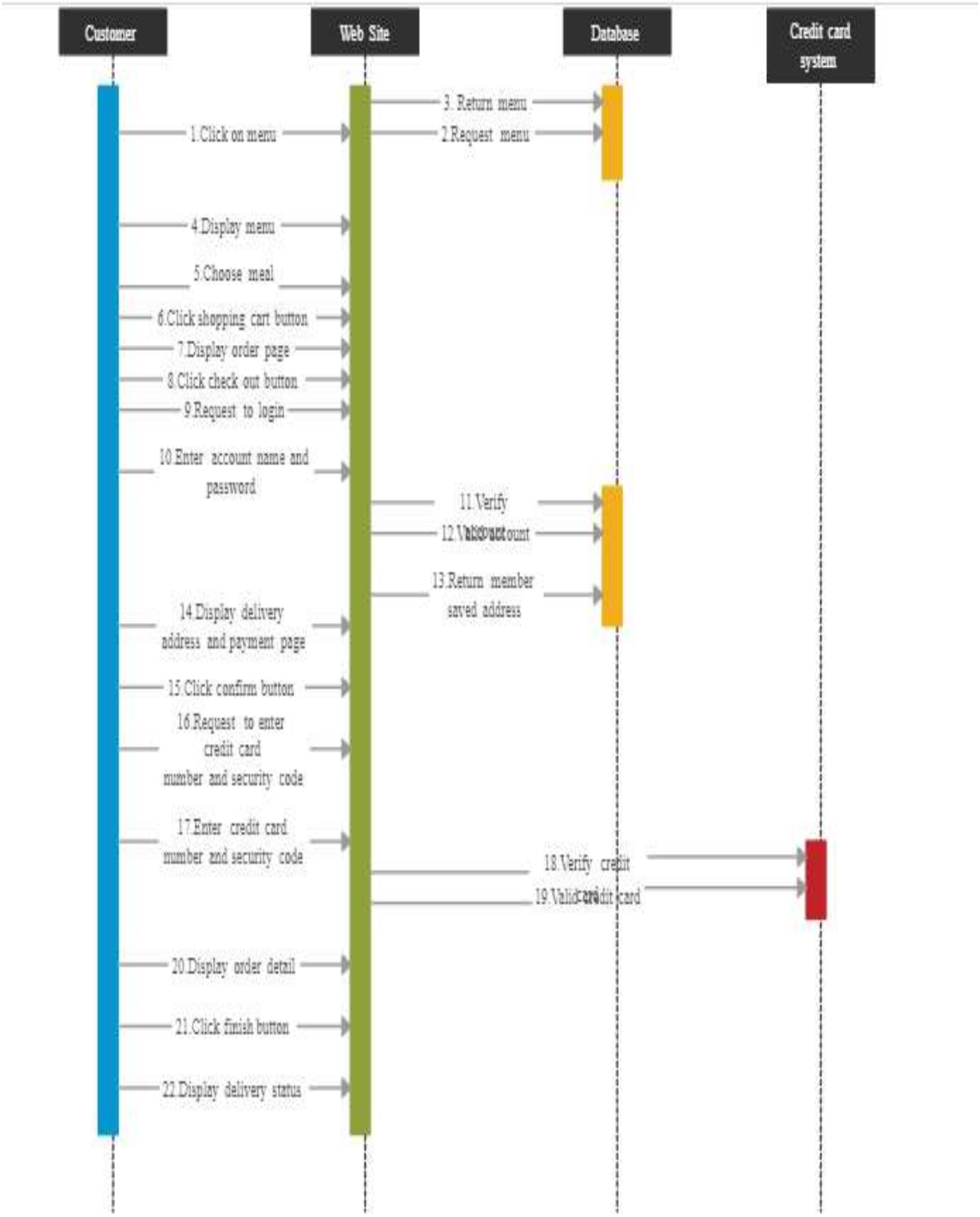

Figure 3.6 :sequence diagram

3.5.4. Activity Diagram:

An activity diagram is a special type of state chart diagram. It usually depicts the flow of events within an object. An activity diagram addresses the dynamic view of a system. They are especially important in modelling the function of a system and emphasize the flow of control among objects.

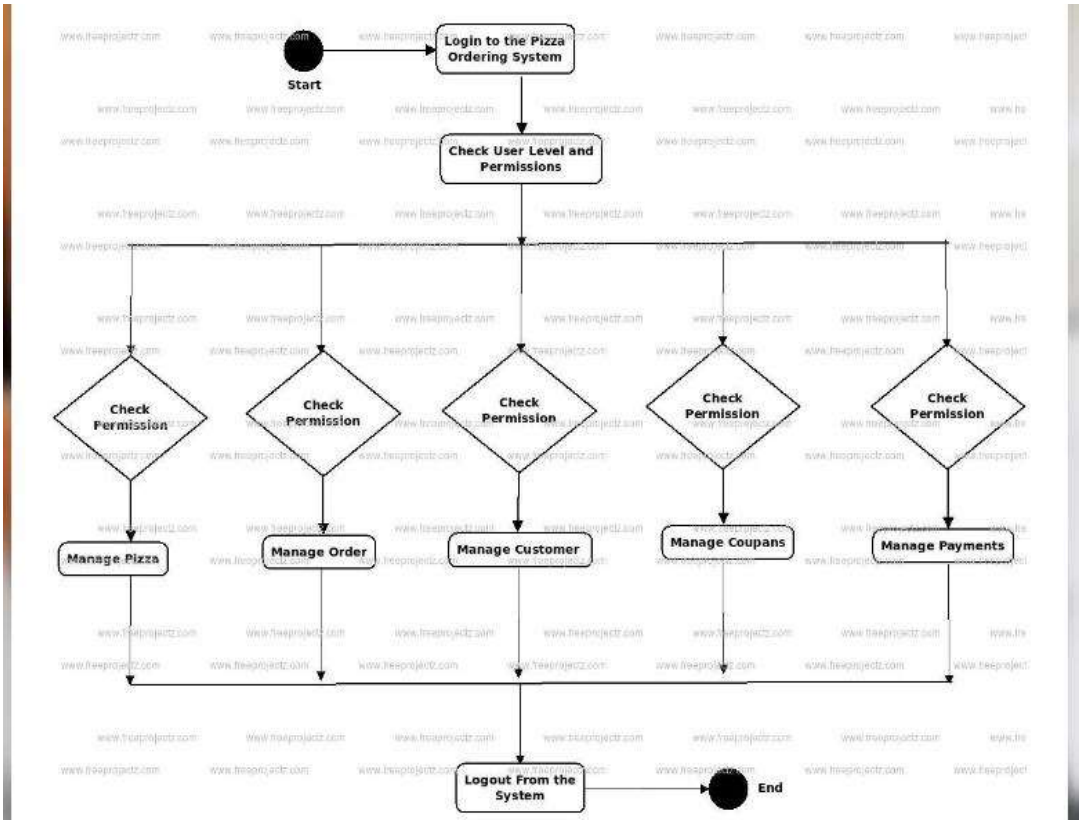

Figure 3.7 : Activity digram

3.5.5. Collaboration Diagram

A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Collaboration diagram address the dynamic view of a system.

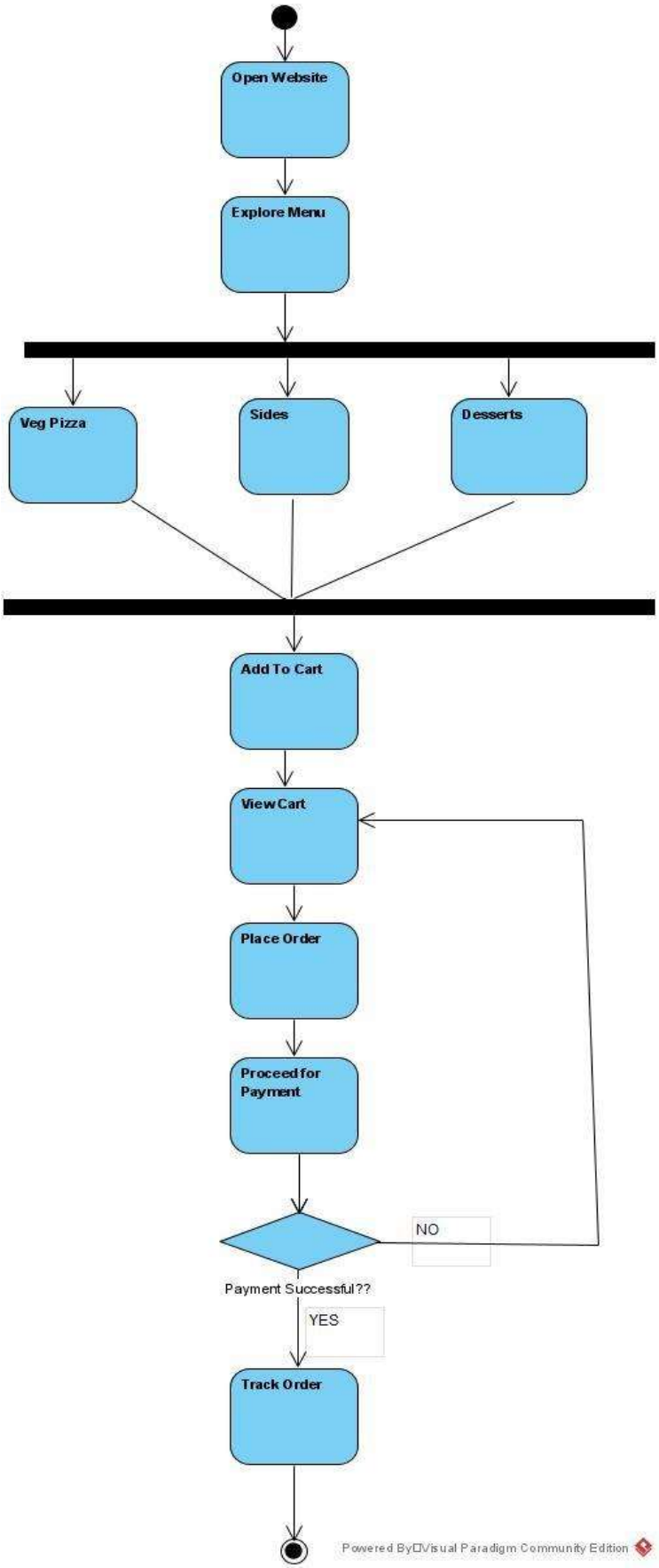

Figure 3.8 : Collaboration diagram

3.5.6. Unified Modelling Language:

UML is a method for describing the system architecture in details using the blueprint. UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software .

3.5.7.UML Specifying:

Specifying means building models that are precise, unambiguous and complete. In particular, the UML address the specification of all the important analysis, design and implementation decisions that must be made in developing and displaying a software intensive system.

3.5.8. UML Visualization:

The UML includes both graphical and textual representation. It makes easy to visualize the system and for better understanding.

3.5.9. UML Constructing:

UML models can be directly connected to a variety of programming languages and it is sufficiently expressive and free from any ambiguity to permit the direct execution of models.

3.5.10.UML Documentation:

UML provides variety of documents in addition raw executable codes. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a se t of diagrams, which is as follows.

3.5.11. User Model View:

- This view represents the system from the user's perspective.
- The analysis representation describes the usage scenario from the end-user's perspective.

3.5.12. Goal of UML:

- The primary goals in the design of the UML were:
- Provide users with a ready-to-use, expressive visual modelling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanism to extend the core concepts.
- Be independent of particular programming language and development processes.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frame works, patterns and components.
- Integrate best practices

➢ Uses of UML:

The UML is intended primarily for software intensive systems. It has been used effectively for such domains as :

- Enterprise Information System
- Telecommunications
- Transportation
- Defense /Aerospace
- Retails
- Medical Electronics
- Scientific Fields
- Distributed Web

➢ Rules of UML:

The UML has semantic rules for

1. NAMES: It will call things, relationships and diagrams.
2. SCOPE: The content that gives specific meaning to a name.
3. VISIBILITY: How those names can be seen and used by others.
4. INTEGRITY: How things properly and consistently relate to another.
5. EXECUTION: What it means is to run or simulate a dynamic model.

➢ Building blocks of UML:

The vocabulary of the UML encompasses 3 kinds of building blocks

1. Things

2. Relationships
3. Diagrams

➢ Things:

- Things are the data abstractions that are first class citizens in a model. Things are of four types
- Structural Things
- Behavioural Things
- Grouping Thing
- A notational Things

➢ Relationships:

- Relationships tie the things together. Relationships in the UML are
- Dependency
- Association
- Generalization
- Specialization

3.6. Modules :

The project entitled with "Pizza Ordering System" is divided into numerous modules. The detail description about the whole modules will be explained in below:

3.6.1. User Module:

User module provide accessibility to user so they can manage their profile and use other modules.

3.6.2. Login/Sign-Up:

The user who is already registered can login to system by using their login id and password.

The user who is not registered can create his or her account , option of sign-up is available.

3.6.3. Add To Cart:

This feature is useful in a situation where you have to order more than one pizza. Suppose you have selected a Margherita pizza and now you want to select another pizza as well, then you just have to add that pizza to the cart using the Add to cart option.

Items added to the cart will be saved so that you can choose other pizzas as well.

3.6.4. Build Your Pizza:

- The customer will get an option to select the pizza crust's size which will be small, medium and large.
- Then the customer has to choose the sauce to be used on his pizza crust. These can be marinara cheese, ranch and others.
- At last the system shows the customers different types of toppings.

3.6.5. Administration:

- Admin can add/edit/delete different ingredients into the system.
- He can add/edit/delete new pizzas, their images, prices, and other details.
- He can add/edit/delete users in this system. It's the admin who adds new employees' accounts to the system.
- Admin has rights to add/edit/delete orders for customers.

3.6.6. Payment Module:

Payment module provide an interactive interface and integration to website so user can pay online and all the access of the transaction control by admin.

CHAPTER 4

UI DESIGNS

4.1.From The User Side:

This is the user side when user login to websites.

4.1.1 Home Page:

The First Interface of the Pizza Ordering Website.

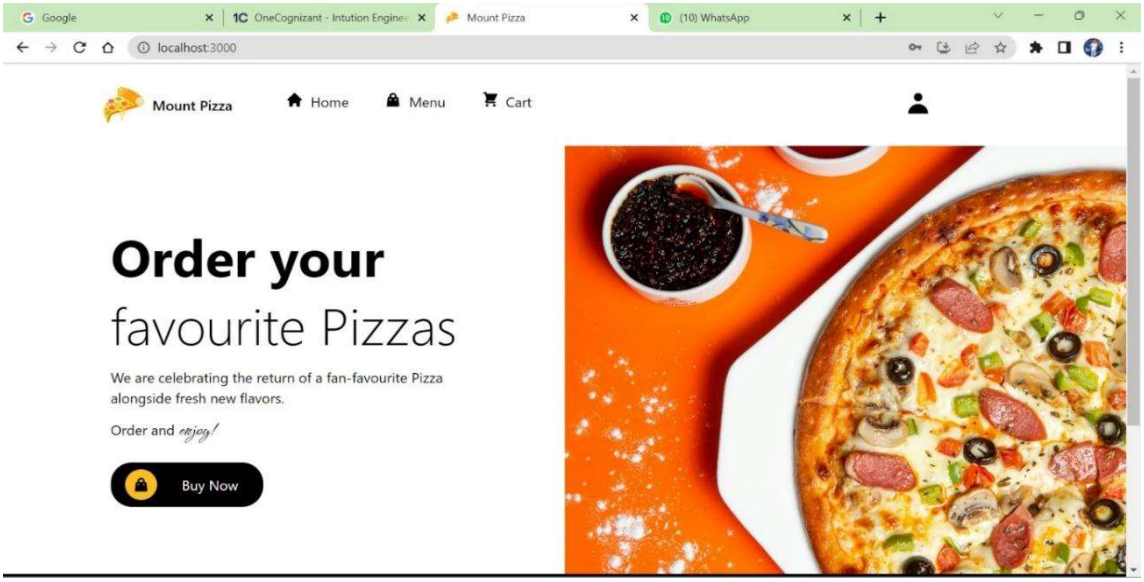

Figure 4.1 :Home page

4.1.2. Login/Sign-Up Page:

The next page of the web application is the login or sign-up page in we have to enter the credentials or create an account (if new user) and then press next to move on to the next page.

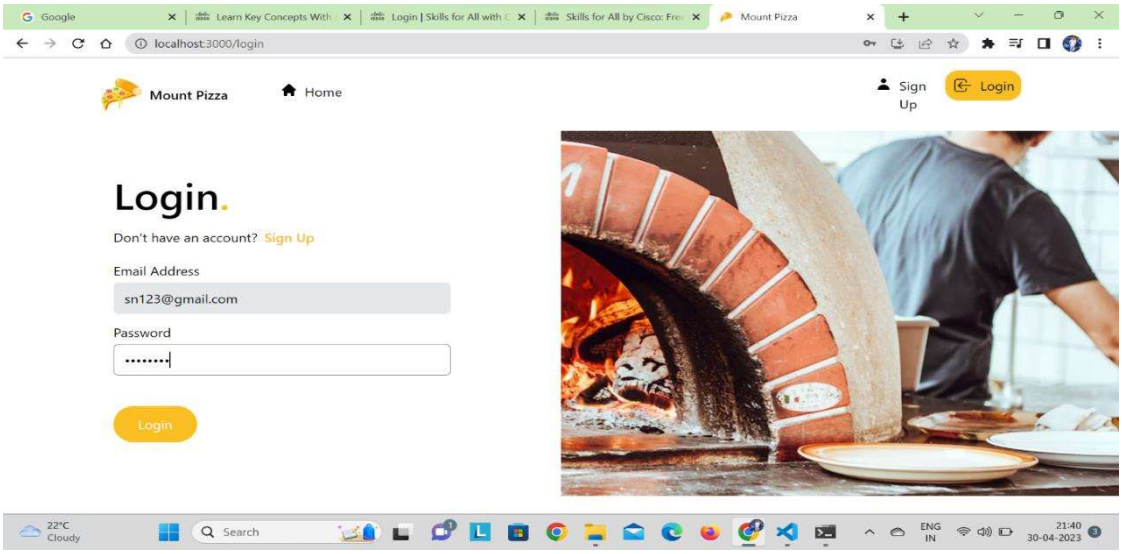

Figure 4.2 : Login page

4.1.3. Order Your Pizza:

The User can choose and order pizza of his/her choice.

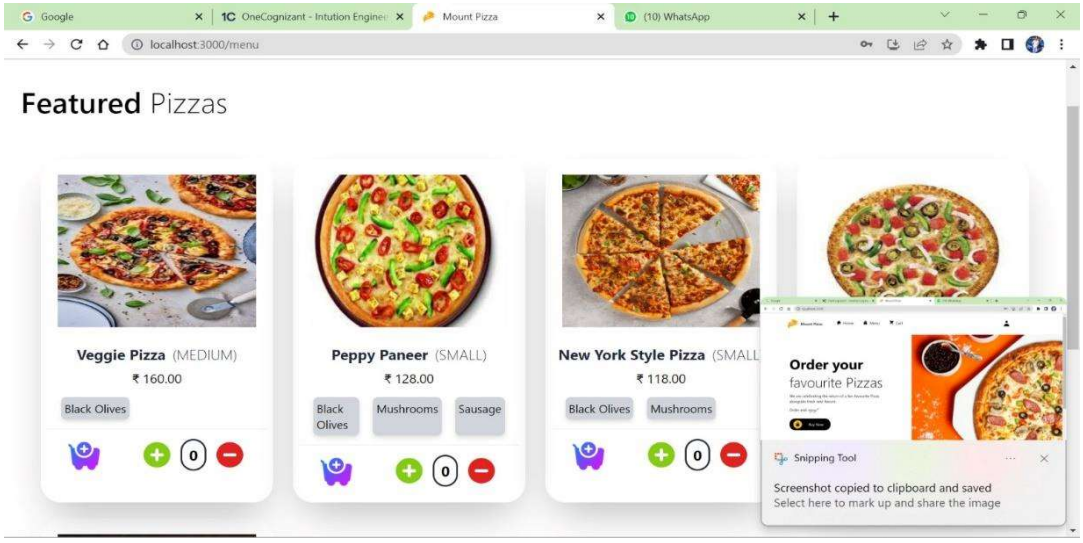

Figure 4.3 : Menu page

4.1.4. Add to Cart:

The user can add pizzas to cart in case he/she needs to order more than one pizza or may be later.

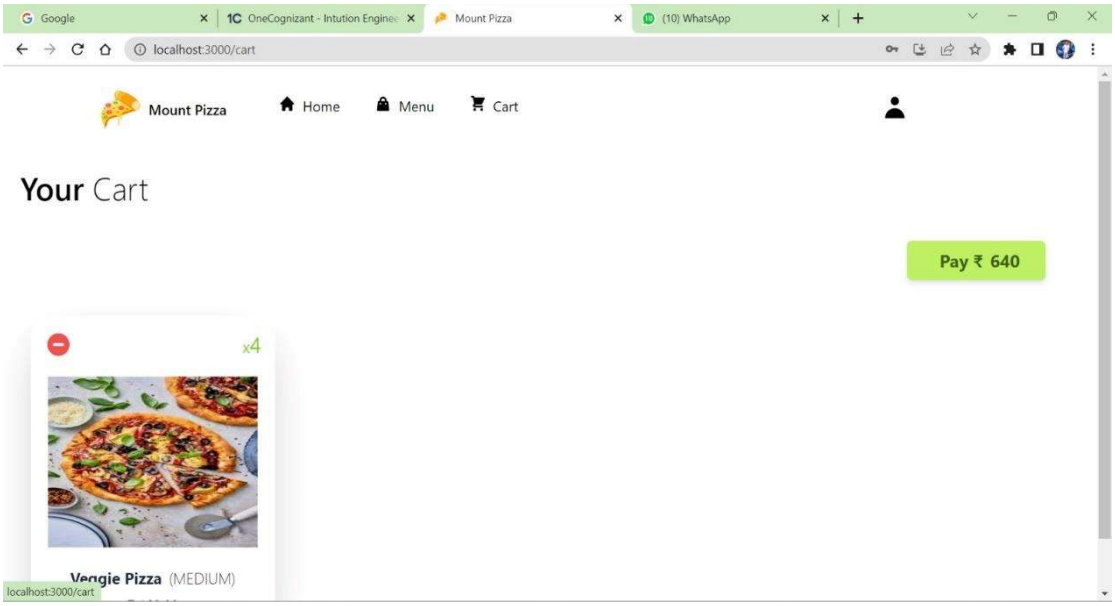

Figure 4.4 : Cart page

4.2.From the Admin Side: This is admin side of design as its functionalities shown below :

4.2.1.Home Page:

Home page is a first page of admin side where he can see the options like menu , cart and user profile.

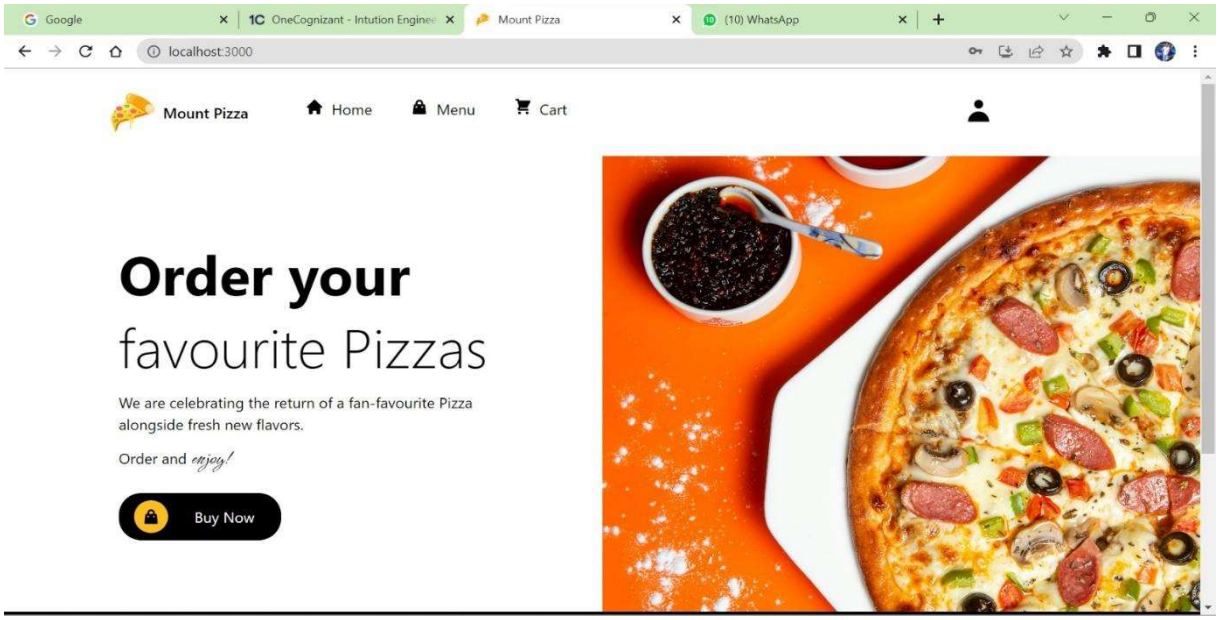

Figure 4.5 : Home page

4.2.2. Login Page:

This is a login page where user or admin enter their credentials.

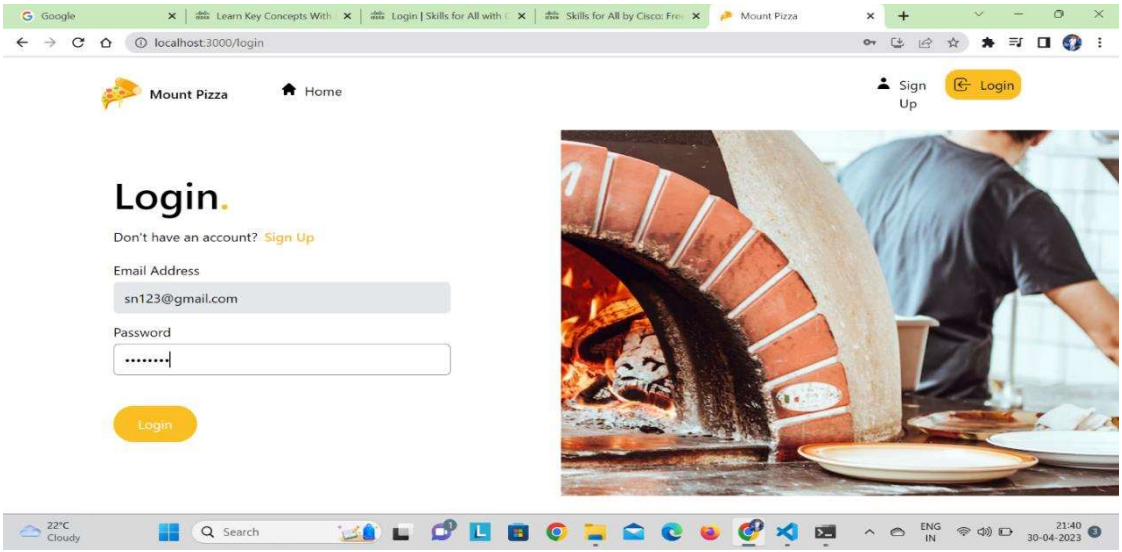

Figure 4.6 Login Page

4.2.3. Manage Users: The Admin has the right to manage and see the users who are accessing his/her web Application.

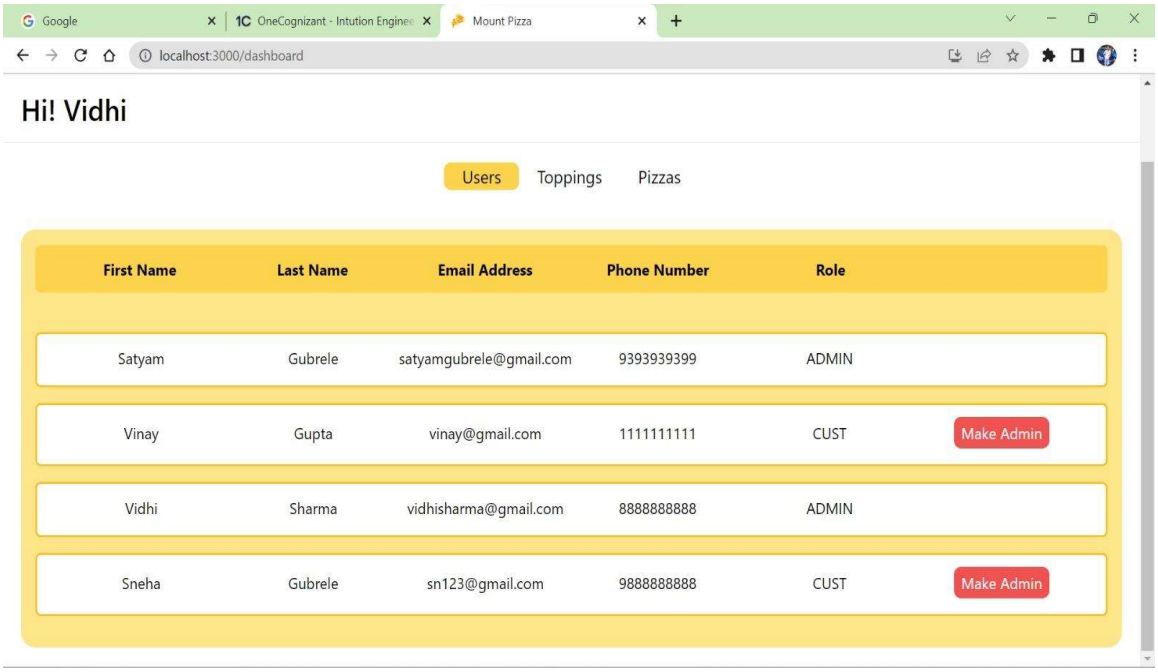

Figure 4.7 : Admin module

4.2.4. Build Your Pizza:

- The Admin can build pizza according to his/her choice.
- He/she can add toppings and can build pizza of his/her own choice.

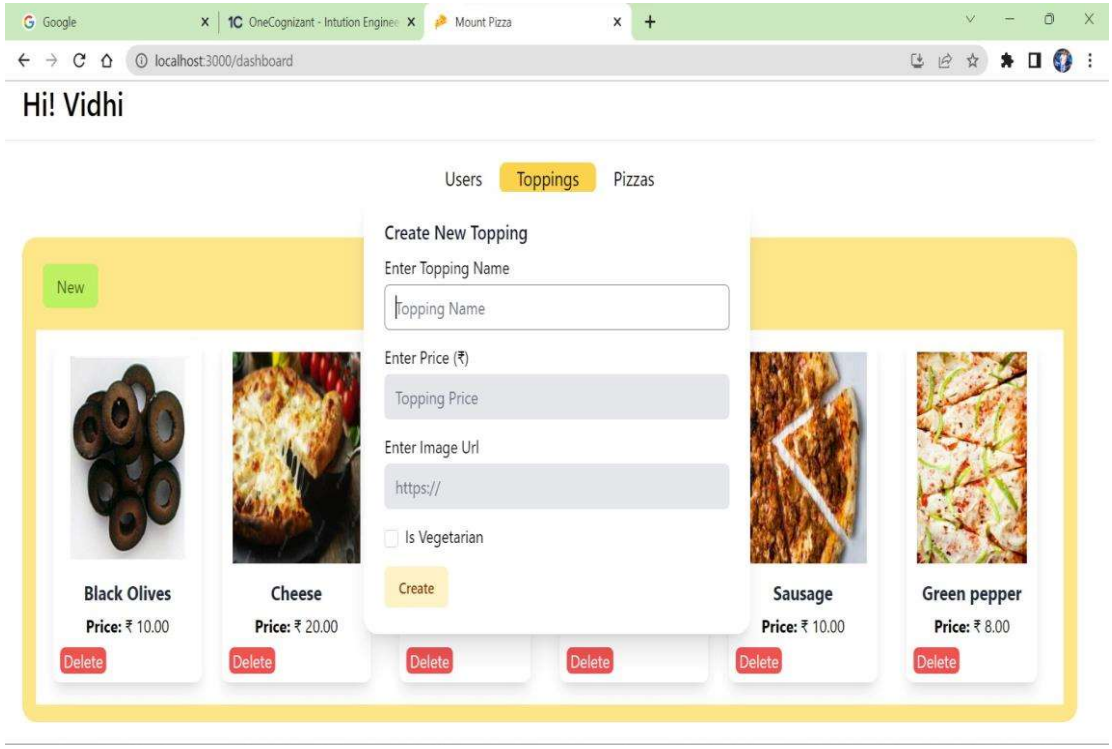

Figure 4.8

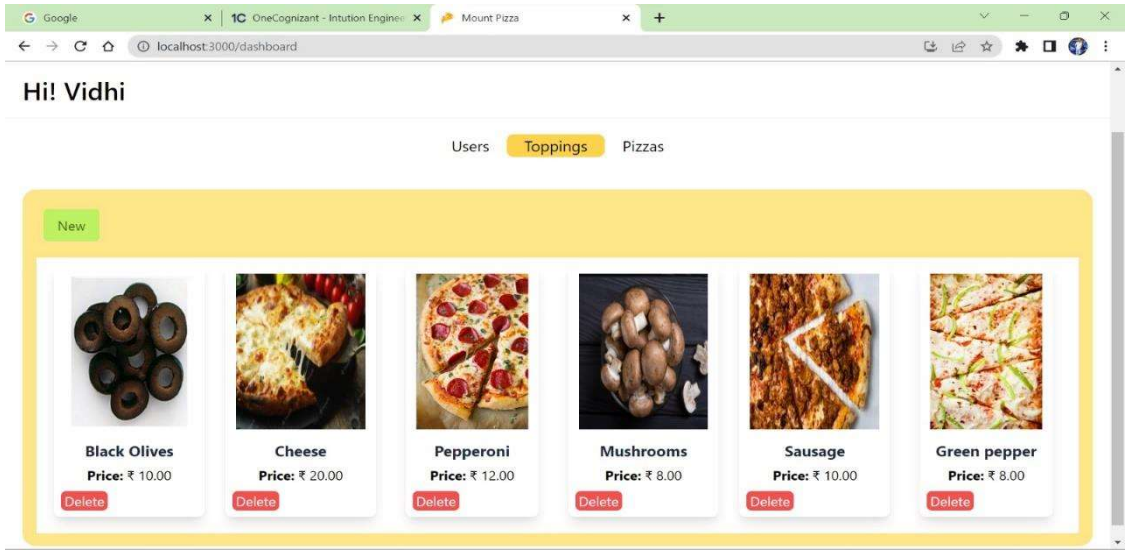

Figure 4.9

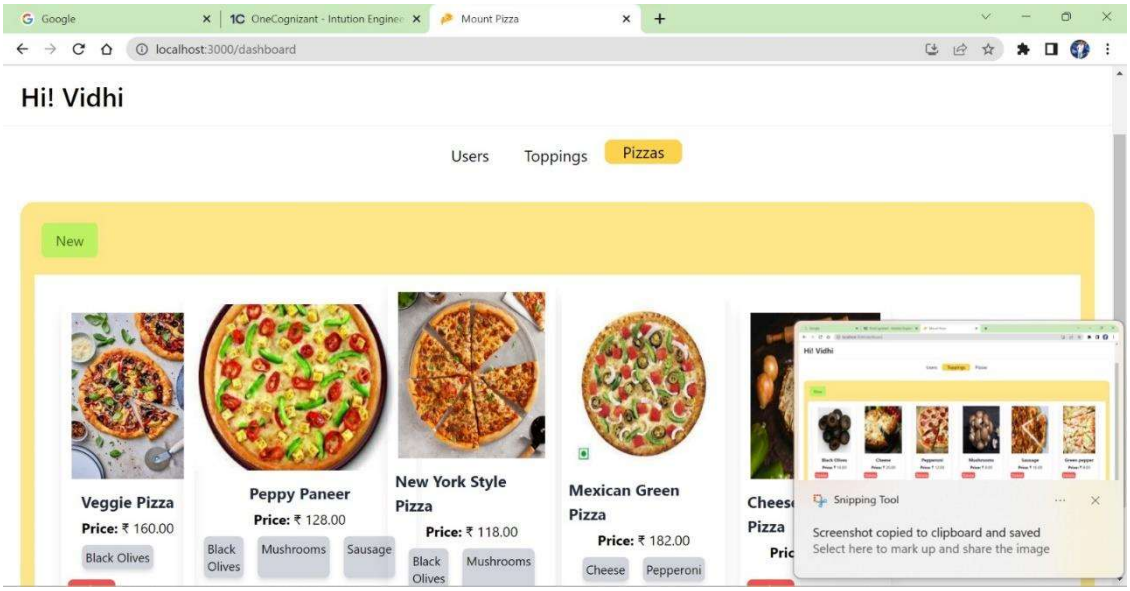
Google
OneCognizant - Intution Enginee
Mount Pizza
localhost:3000/dashboard
Hi! Vidhi
Users
Toppings
Pizzas
New
Veggie Pizza
Price: ₹ 160.00
Black Olives
Peppy Paneer
Price: ₹ 128.00
Black Olives
Mushrooms
Sausage
New York Style Pizza
Price: ₹ 118.00
Black Olives
Mushrooms
Mexican Green Pizza
Price: ₹ 182.00
Cheese
Pepperoni
Snipping Tool
Screenshot copied to clipboard and saved
Select here to mark up and share the image

Figure 4.10

CHAPTER 5

CODING

MODULEWISE:

In the section below, module wise coding is given:

1.1.Index.Js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { AuthContextProvider } from './store/auth-context';



ReactDOM.render(
  <React.StrictMode>
    <AuthContextProvider>
      <App />
    </AuthContextProvider>
  </React.StrictMode>,
  document.getElementById('root')
);
```

1.2.Index.css :

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

1.3.App.js:

```
import {
  BrowserRouter as Router,
  Routes,
  Route,
} from 'react-router-dom';
import { useContext } from 'react';
import AuthContext from './store/auth-context';

// importing components
import Navbar from './components/Navbar/Navbar';

// importing routes
import Home from './Routes/Home/Home';
import Menu from './Routes/Menu/Menu';
import Signup from './Routes/Signup/Signup';
import Login from './Routes/Login/Login';
import Cart from './Routes/Cart/Cart';
import NotFound from './Routes/404/404';
import { Dashboard } from './Routes/Dashboard/Dashboard';
```

```
function App() {
  const authCtx = useContext(AuthContext);
  return (
    <div className='h-screen select-none'>
      <Router>
        <Navbar />
        <Routes>
          <Route exact path='/' element={<Home />} />
          {
            authCtx.user.role === 'CUST'
            &&
            authCtx.isLoggedIn
            &&
            <Route exact path='/menu' element={<Menu />} />
          }
          {
            authCtx.user.role === 'CUST'
            &&
            authCtx.isLoggedIn
            &&
            <Route exact path='/cart' element={<Cart />} />
          }
          {
            !authCtx.isLoggedIn && <Route exact path='/login'
element={<Login />} />
          }
          {
            !authCtx.isLoggedIn && <Route exact path='/signup'
element={<Signup />} />
          }
          {
            authCtx.user.role === 'ADMIN'
            &&
            authCtx.isLoggedIn
            &&
            <Route exact path='/dashboard' element={ <Dashboard /> } />
          }
          <Route path='*' element={<NotFound />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```

2. Modules: There are some modules in project.

2.1.Dashboard :

2.1.1:Board.js:

```
import { useEffect, useState, Fragment } from "react";
import { Dialog, Transition } from '@headlessui/react'
import { getAllUsers, titleCase, makeAdmin, getAllToppings,
deleteTopping, createTopping } from '../../../utils/helper';


import { UsersBoard } from "../UsersBoard/UsersBoard";
import { ToppingBoard } from '../ToppingsBoard/ToppingsBoard';
import { PizzasBoard } from "../PizzasBoard/PizzaBoard";


export function Board(props) {
    return (
        <div className="board bg-amber-200 rounded-2xl p-4">
            {
                props.selectedBoard === 'users'
                &&
                <UsersBoard token={props.token} />
            }

            {
                props.selectedBoard === 'toppings'
                &&
                <ToppingBoard token={props.token} />
            }

            {
                props.selectedBoard === 'pizzas'
                &&
                <PizzasBoard token={props.token} />
            }
        </div>
    );
}
```

2.1.2: MenuNavigation:

```
import { useEffect, useState, Fragment } from "react";
import { Dialog, Transition } from '@headlessui/react'
```

```
    import { getAllUsers, titleCase, makeAdmin, getAllToppings,
deleteTopping, createTopping } from '../../../utils/helper';


    import { UsersBoard } from "../UsersBoard/UsersBoard";
    import { ToppingBoard } from '../ToppingsBoard/ToppingsBoard';
    import { PizzasBoard } from "../PizzasBoard/PizzaBoard";


    export function Board(props) {
        return (
            <div className="board bg-amber-200 rounded-2xl p-4">
                {
                    props.selectedBoard === 'users'
                    &&
                    <UsersBoard token={props.token} />
                }

                {
                    props.selectedBoard === 'toppings'
                    &&
                    <ToppingBoard token={props.token} />
                }

                {
                    props.selectedBoard === 'pizzas'
                    &&
                    <PizzasBoard token={props.token} />
                }
            </div>
        );
    }
```

2.1.3 PizzaBoard:

```
import { useEffect, useState, Fragment } from "react";
import { Dialog, Transition } from '@headlessui/react'
import { getAllPizzas, createPizza, deletePizza, getAllToppings,
formatToppingNameToId } from '../../../utils/helper';
import Select from 'react-select';
import { correctToppingFormat } from "../../../utils/helper";
import { ToppingCard } from "../../PizzaCard/ToppingCard/ToppingCard";


function PizzaCard(props) {
    const handleDelete = async () => {
        const res = await deletePizza(props.token, props.pizza._id);
        if (res.error) {
            alert(res.message);
```

```
        } else {
            alert('Pizza deleted');
            props.setDeleted(true);
        }
    }

    return (
        <div className="pizza shadow-lg rounded-lg md:mx-3 mb-3 p-2">
            <div className="pizza__image p-1 mb-2">
                <img className="h-48 w-full" src={props.pizza.imageUrl}
alt={props.pizza.name} />
            </div>

            <div className="topping__name flex justify-center items-
center font-bold text-lg text-gray-800 mb-1">
                {props.pizza.name}
            </div>

            <div className="topping__price flex justify-center items-
center text-base mb-2">
                <span className="mr-1 font-bold">Price: </span>{`₹
${props.pizza.price}.00`}
            </div>

            <div className="pizza__toppings mb-3 px-2 flex grid-cols-3">
                {
                    props.pizza.toppings.map(topping => {
                        return <ToppingCard key={topping._id}
topping={topping} />
                    })
                }
            </div>

            <div className="topping__delete flex justify-right items-
center">
                <button onClick={handleDelete} className="bg-red-500
text-white px-1 rounded-md">
                    Delete
                </button>
            </div>
        </div>
    );
}

function ToppingsSelect(props) {
    const [toppingsName, setToppingsName] = useState([]);
    const [toppingOptions, setToppingOptions] = useState([]);
    const [toppingData, setToppingData] = useState([]);
```

```
    useEffect(() => {
        // fetch toppings data
        const fetchToppings = async (token) => {
            const res = await getAllToppings(token);
            if (res.error) {
                alert(res.message);
            } else {
                setToppingOptions(correctToppingFormat(res.data));
                setToppingData(res.data);
            }
        }
        fetchToppings(props.token);
    }, [])

    useEffect(() => {
        const ids = formatToppingNameToId(toppingsName, toppingData)
        props.setToppings(ids);
    }, [toppingsName])

    return (
        <>
            <Select
                onChange={e => setToppingsName(e)}
                isMulti
                name='toppings'
                options={toppingOptions}
                required
            />
        </>
    );
}

function CrustSelect(props) {
    const crustOptions = [
        { value: 'SMALL', label: 'Small' },
        { value: 'MEDIUM', label: 'Medium' },
        { value: 'LARGE', label: 'Large' },
    ]
    return (
        <>
            <Select
                onChange={e => props.setCrustType(e.value)}
                name='crust'
                options={crustOptions}
                required
            />
        </>
    );
}
```

```
function CreatePizzaButton(props) {
    let [isOpen, setIsOpen] = useState(false);
    const [name, setName] = useState(null);
    const [imageUrl, setImageUrl] = useState(null);
    const [crustType, setCrustType] = useState(null);
    const [toppings, setToppings] = useState([]);

    const handleSubmit = async (e) => {
        e.preventDefault();
        const res = await createPizza(props.token, {
            name,
            imageUrl,
            crustType,
            toppings
        })

        if (res.error) {
            alert(res.message);
        } else {
            alert('Pizza is created!');
            props.setNew(true);
            closeModal();
        }
    }

    function closeModal() {
        setIsOpen(false)
    }

    function openModal() {
        setIsOpen(true)
    }

    return (
        <>
            <div>
                <button
                    type="button"
                    onClick={openModal}
                    className="px-4 py-2 hover:-translate-y-1 transition
bg-lime-300 hover:bg-lime-500 text-lime-800 rounded-md"
                >
                    New
                </button>
            </div>

            <Transition appear show={isOpen} as={Fragment}>
                <Dialog
```

```
                    as="div"
                    className="fixed inset-0 z-10 overflow-y-auto"
                    onClose={closeModal}
                >
                    <div className="min-h-screen px-4 text-center">
                        <Transition.Child
                            as={Fragment}
                            enter="ease-out duration-300"
                            enterFrom="opacity-0"
                            enterTo="opacity-100"
                            leave="ease-in duration-200"
                            leaveFrom="opacity-100"
                            leaveTo="opacity-0"
                        >
                            <Dialog.Overlay className="fixed inset-0" />
                        </Transition.Child>

                        <span
                            className="inline-block h-screen align-
middle"
                            aria-hidden="true"
                        >
                            &#8203;
                        </span>
                        <Transition.Child
                            as={Fragment}
                            enter="ease-out duration-300"
                            enterFrom="opacity-0 scale-95"
                            enterTo="opacity-100 scale-100"
                            leave="ease-in duration-200"
                            leaveFrom="opacity-100 scale-100"
                            leaveTo="opacity-0 scale-95"
                        >
                            <div className="inline-block w-full max-w-md
p-6 my-8 text-left align-middle transition-all transform shadow-xl
rounded-2xl bg-white">
                                <Dialog.Title
                                    as="h3"
                                    className="text-lg font-medium
leading-6 text-gray-900"
                                >
                                    Create New Pizza
                                </Dialog.Title>

                                <div className="mt-2">
                                    <form onSubmit={handleSubmit}>
                                        <div className="mb-3">
                                            <label className="block">
```

```
                                                <span className="text-
grey-700">Pizza Name</span>
                                                <input type='text'
onChange={e => setName(e.target.value)} className="mt-1 block w-full
rounded-md bg-gray-200 border-transparent focus:border-gray-500 focus:bg-
white focus:ring-0" placeholder='Enter Pizza Name' required />
                                            </label>
                                        </div>

                                        <div className="mb-3">
                                            <label className="block">
                                                <span className="text-
grey-700">Enter Image Url</span>
                                                <input type='text'
onChange={e => setImageUrl(e.target.value)} className="mt-1 block w-full
rounded-md bg-gray-200 border-transparent focus:border-gray-500 focus:bg-
white focus:ring-0" placeholder='https://' required />
                                            </label>
                                        </div>


                                        <div className="mb-3">
                                            <label className="block">
                                                <span className="text-
grey-700">Pizza Crust Type</span>
                                                <CrustSelect
setCrustType={setCrustType} />
                                            </label>
                                        </div>

                                        {/* TODO react multiselect */}
                                        <div className="mb-3">
                                            <label className="block">
                                                <span className="text-
grey-700">Choose Toppings</span>
                                                <ToppingsSelect
token={props.token} setToppings={setToppings} />
                                            </label>
                                        </div>


                                        <div className="mt-4">
                                            <button
                                                type="submit"
                                                className="inline-flex
justify-center px-4 py-2 text-sm font-medium text-amber-900 bg-amber-100
border border-transparent rounded-md hover:bg-amber-200 focus:outline-
none focus-visible:ring-2 focus-visible:ring-offset-2 focus-visible:ring-
amber-500"
```

```
                                        >
                                            Create
                                        </button>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </Transition.Child>
                </div>
            </Dialog>
        </Transition>
    </>
  );
}

function NoPizza() {
  return (
    <>
      <div className="noPizza grid grid-cols-1 justify-items-center
items-center bg-white rounded-lg">
        <div className="mb-5">
          <img className="w-52 h-52" src='/topping.png' alt='no
pizza' />
        </div>

        <div className="text-4xl font-bold mb-10">
          No Pizza Found!
        </div>
      </div>
    </>
  )
}

export function PizzasBoard(props) {
  const [pizzaList, setPizzaList] = useState([]);
  const [isNewCreated, setIsNewCreated] = useState(false);
  const [isDeleted, setIsDeleted] = useState(false);


  useEffect(() => {
    const fetchPizzas = async (token) => {
      const res = await getAllPizzas(token);
      if (res.error) {
        console.log(res.message)
      } else {
        setPizzaList(res.data);
      }
    }
```

```
        fetchPizzas(props.token);
        setIsNewCreated(false);
        setIsDeleted(false);
    }, [isNewCreated, isDeleted])


    return (
        <div className="pizza">
            <div className="pizza__options flex p-2 rounded-md mb-3">
                <CreatePizzaButton setNew={setIsNewCreated}
token={props.token} />
            </div>
            {
                pizzaList.length !== 0
                    ?
                    <div className="pizza__board p-2 bg-white">
                        <div className="grid grid-cols-1 justify-items-
center items-center md:grid-cols-6">
                            {
                                pizzaList.map(pizza => {
                                    return <PizzaCard
setDeleted={setIsDeleted} setNew={setIsNewCreated} key={pizza._id}
pizza={pizza} token={props.token} />
                                })

                            }
                        </div>
                    </div>
                    :
                    <NoPizza />
            }

        </div>
    );
}
```

2.1.3 ToppingBoard:

```
import { useEffect, useState, Fragment } from "react";
import { Dialog, Transition } from '@headlessui/react'
import { getAllToppings, createTopping, deleteTopping } from
'../../../utils/helper';


function ToppingCard(props) {
    const handleDelete = async () => {
        const res = await deleteTopping(props.token, props.topping._id);
        if (res.error) {
```

```
            alert(res.message);
        } else {
            alert('Topping deleted');
            props.setDeleted(true);
        }
    }

    return (
        <div className="topping shadow-lg rounded-lg md:mx-3 mb-3 p-2">
            <div className="topppping__image p-1 mb-2">
                <img className="h-48 w-full" src={props.topping.imageUrl}
alt={props.topping.name} />
            </div>

            <div className="topping__name flex justify-center items-
center font-bold text-lg text-gray-800 mb-1">
                {props.topping.name}
            </div>

            <div className="topping__price flex justify-center items-
center text-base mb-2">
                <span className="mr-1 font-bold">Price: </span>{`₹
${props.topping.price}.00`}
            </div>

            <div className="topping__delete flex justify-right items-
center">
                <button onClick={handleDelete} className="bg-red-500
text-white px-1 rounded-md">
                    Delete
                </button>
            </div>
        </div>
    );
}

function CreateToppingButton(props) {
    let [isOpen, setIsOpen] = useState(false);
    const [name, setName] = useState(null);
    const [price, setPrice] = useState(0);
    const [imageUrl, setImageUrl] = useState(null);
    const [category, setCategory] = useState('VEG');

    const handleSubmit = async (e) => {
        e.preventDefault();
        const res = await createTopping(props.token, {
            name,
            price,
            imageUrl,
```

```
            category,
        })

        if (res.error) {
            alert(res.message);
        } else {
            alert('Topping is created!');
            closeModal();
            props.setNew(true);
        }
    }

    function closeModal() {
        setIsOpen(false)
    }

    function openModal() {
        setIsOpen(true)
    }

    return (
        <>
            <div>
                <button
                    type="button"
                    onClick={openModal}
                    className="px-4 py-2 hover:-translate-y-1 transition
bg-lime-300 hover:bg-lime-500 text-lime-800 rounded-md"
                >
                    New
                </button>
            </div>

            <Transition appear show={isOpen} as={Fragment}>
                <Dialog
                    as="div"
                    className="fixed inset-0 z-10 overflow-y-auto"
                    onClose={closeModal}
                >
                    <div className="min-h-screen px-4 text-center">
                        <Transition.Child
                            as={Fragment}
                            enter="ease-out duration-300"
                            enterFrom="opacity-0"
                            enterTo="opacity-100"
                            leave="ease-in duration-200"
                            leaveFrom="opacity-100"
                            leaveTo="opacity-0"
                        >
```

```
                        <Dialog.Overlay className="fixed inset-0" />
                    </Transition.Child>

                    <span
                        className="inline-block h-screen align-
middle"
                        aria-hidden="true"
                    >
                        &#8203;
                    </span>
                    <Transition.Child
                        as={Fragment}
                        enter="ease-out duration-300"
                        enterFrom="opacity-0 scale-95"
                        enterTo="opacity-100 scale-100"
                        leave="ease-in duration-200"
                        leaveFrom="opacity-100 scale-100"
                        leaveTo="opacity-0 scale-95"
                    >
                        <div className="inline-block w-full max-w-md
p-6 my-8 overflow-hidden text-left align-middle transition-all transform
shadow-xl rounded-2xl bg-white">
                            <Dialog.Title
                                as="h3"
                                className="text-lg font-medium
leading-6 text-gray-900"
                            >
                                Create New Topping
                            </Dialog.Title>

                            <div className="mt-2">
                                <form onSubmit={handleSubmit}>
                                    <div className="mb-3">
                                        <label className="block">
                                            <span className="text-
grey-700">Enter Topping Name</span>
                                            <input type='text'
onChange={e => setName(e.target.value)} className="mt-1 block w-full
rounded-md bg-gray-200 border-transparent focus:border-gray-500 focus:bg-
white focus:ring-0" placeholder='Topping Name' required />
                                        </label>
                                    </div>

                                    <div className="mb-3">
                                        <label className="block">
                                            <span className="text-
grey-700">Enter Price (₹)</span>
                                            <input type='text'
onChange={e => setPrice(e.target.value)} className="mt-1 block w-full
```

```
rounded-md bg-gray-200 border-transparent focus:border-gray-500 focus:bg-
white focus:ring-0" placeholder='Topping Price' required />
                                        </label>
                                    </div>

                                    <div className="mb-3">
                                        <label className="block">
                                            <span className="text-
grey-700">Enter Image Url</span>
                                            <input type='text'
onChange={e => setImageUrl(e.target.value)} className="mt-1 block w-full
rounded-md bg-gray-200 border-transparent focus:border-gray-500 focus:bg-
white focus:ring-0" placeholder='https://' required />
                                        </label>
                                    </div>

                                    <div className="mb-3">
                                        <input type="checkbox"
className="rounded border-gray-300 text-indigo-600 shadow-sm
focus:border-indigo-300 focus:ring focus:ring-offset-0 focus:ring-indigo-
200 focus:ring-opacity-50" />
                                        <span className="ml-2">Is
Vegetarian</span>
                                    </div>

                                    <div className="mt-4">
                                        <button
                                            type="submit"
                                            className="inline-flex
justify-center px-4 py-2 text-sm font-medium text-amber-900 bg-amber-100
border border-transparent rounded-md hover:bg-amber-200 focus:outline-
none focus-visible:ring-2 focus-visible:ring-offset-2 focus-visible:ring-
amber-500"
                                        >
                                            Create
                                        </button>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </Transition.Child>
                </div>
            </Dialog>
        </Transition>
    </>
  );
}

function NoTopping() {
```

```
    return (
        <>
            <div className="noTopping grid grid-cols-1 justify-items-
center items-center bg-white rounded-lg">
                <div className="mb-5">
                    <img className="w-52 h-52" src='/topping.png' alt='no
topping' />
                </div>

                <div className="text-4xl font-bold mb-10">
                    No Toppings Found!
                </div>
            </div>
        </>
    )
}

export function ToppingBoard(props) {
    const [toppingList, setToppingList] = useState([]);
    const [isNewCreated, setIsNewCreated] = useState(false);
    const [isDeleted, setIsDeleted] = useState(false)

    useEffect(() => {
        const fetchToppings = async (token) => {
            const res = await getAllToppings(token);
            if (res.error) {
                console.log(res.message)
            } else {
                setToppingList(res.data);
            }
        }

        fetchToppings(props.token);
        setIsNewCreated(false);
        setIsDeleted(false);
    }, [isNewCreated, isDeleted])

    return (
        <div className="toppings">
            <div className="toppings__options flex p-2 rounded-md mb-3">
                <CreateToppingButton setNew={setIsNewCreated}
token={props.token} />
            </div>


            {
                toppingList.length !== 0
                    ?
                    <div className="toppings__board p-2 bg-white">
```

```
                        <div className="grid grid-cols-1 justify-items-
center items-center md:grid-cols-6">
                            {
                                toppingList.map(topping => {
                                    return <ToppingCard
setDeleted={setIsDeleted} setNew={setIsNewCreated} key={topping._id}
topping={topping} token={props.token} />
                                })

                            }
                        </div>
                    </div>
                    :
                    <NoTopping />
            }

        </div>
    );
}
```

2.1.3 UserBoard :

```
import { useEffect, useState } from "react";
import { getAllUsers, titleCase, makeAdmin } from
'../../../utils/helper';


function RoleChangeButton(props) {
    const handleRoleChange = async () => {
        const res = await makeAdmin(props.token, props.id);
        if (res.error) {
            alert(res.message);
        } else {
            alert('Role changed');
            props.setIsRoleChanged(true);
        }
    }

    return (
        <>
            <button className={props.currentRole === 'ADMIN' ? 'hidden' :
''} onClick={handleRoleChange}>
                <div className="bg-red-500 px-2 py-1 rounded-lg text-
white">
                    Make Admin
                </div>
            </button>
```

```
        </>
    );
}

export function UsersBoard(props) {
    const [users, setUsers] = useState([]);
    const [isRoleChanged, setIsRoleChanged] = useState(false);

    useEffect(() => {
        async function fetchUsers() {
            const res = await getAllUsers(props.token);
            if (res.error) {
                alert(res.message);
            } else {
                setUsers(res.data);
            }
        }

        fetchUsers();
        setIsRoleChanged(false);
    }, [isRoleChanged])

    return (
        <>
            <div className="bg-amber-300 flex-grow grid grid-cols-1
md:grid-cols-6 py-3 px-5 mb-10 font-bold text-base md:items-center
md:justify-items-center rounded-md">
                <div>
                    First Name
                </div>

                <div>
                    Last Name
                </div>

                <div>
                    Email Address
                </div>

                <div>
                    Phone Number
                </div>

                <div>
                    Role
                </div>
            </div>
            {
```

```
                users.map(user => {
                    return <div key={user._id} className="bg-white
border-2 border-amber-400 flex-grow grid grid-cols-1 md:grid-cols-6 py-3
px-5 mb-4 md:items-center md:justify-items-center rounded-md">
                        <div className="mb-1">
                            {titleCase(user.firstName)}
                        </div>

                        <div className="mb-1">
                            {titleCase(user.lastName)}
                        </div>

                        <div className="mb-1">
                            {user.email}
                        </div>

                        <div className="mb-1">
                            {user.phoneNumber}
                        </div>

                        <div className="mb-1">
                            {user.role}
                        </div>

                        <div className="mb-1">
                            <RoleChangeButton id={user._id}
token={props.token} currentRole={user.role}
setIsRoleChanged={setIsRoleChanged} />
                        </div>
                    </div>
                })
            }
        </>

    );
}
```

2.2.1. BuyNowButton:

```
import { useContext } from "react";
import { useNavigate } from "react-router-dom";
import AuthContext from "../../store/auth-context";

export function BuyNowButton() {
    const authCtx = useContext(AuthContext);
    const isLoggedIn = authCtx.isLoggedIn;
```

```
    const navigator = useNavigate();

    const handleClick = () => {
        // if logged in redirect to menu page
        if(isLoggedIn)
            navigator('/menu');
        // else redirect to login page
        else
            navigator('/login');
    }

    return (
        <>
            <button onClick = {handleClick} className="flex p-2 rounded-
3xl items-center justify-items-center bg-black">
                <div className="mx-2 bg-amber-400 p-2 rounded-full">
                    <svg xmlns="http://www.w3.org/2000/svg" className="h-
5 w-5" viewBox="0 0 20 20" fill="currentColor">
                        <path fillRule="evenodd" d="M10 2a4 4 0 00-4
4v1H5a1 1 0 00-.994.89l-1 9A1 1 0 004 18h12a1 1 0 00.994-1.11l-1-9A1 1 0
0015 7h-1V6a4 4 0 00-4-4zm2 5V6a2 2 0 10-4 0v1h4zm-6 3a1 1 0 112 0 1 1 0
01-2 0zm7-1a1 1 0 100 2 1 1 0 000-2z" clipRule="evenodd" />
                    </svg>
                </div>
                <div className="mx-5 text-white">
                    Buy Now
                </div>
            </button>
        </>
    );
}
```

2.2.2.Homepage.js:

```
import './Homepage.css';
import { BuyNowButton } from './BuyNowButton';

import { Footer } from '../Footer/Footer';
import { useContext } from 'react';
import AuthContext from '../../store/auth-context';

function HomePageContent() {
    const authCtx = useContext(AuthContext);

    return (
        <div className="homepage__content p-5">
            <div className="homepage__content__title">
                <div className="text-6xl font-bold my-5">
                    Order your
```

```
                </div>
                <div className="text-6xl font-light my-5">
                    favourite Pizzas
                </div>
            </div>

            <div className="homepage__content__subtitle text-md my-5">
                <div className="my-2">
                    We are celebrating the return of a fan-favourite
Pizza <br />
                    alongside fresh new flavors.
                </div>

                <div className="my-2">
                    Order and <span className='font-
hurricane'>enjoy!</span>
                </div>
            </div>

            {
                authCtx.user.role !== 'ADMIN'

                &&

                <div className='buyNowButton'>
                    <BuyNowButton />
                </div>
            }

        </div>
    )
}

function HomeIllustration() {
    return (
        <div className="bg-image hidden md:block h-full w-full"></div>
    );
}

export function HomePage() {
    return (
        <>
            <div className="
            homepage
            h-5/6
            grid
            grid-cols-1
            md:items-center
            md:justify-items-center
```

```
            md:grid-cols-2
            bg-[url('https://images.pexels.com/photos/315755/pexels-
photo-315755.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1')]
            md:bg-none
            ">
                <HomePageContent />
                <HomeIllustration />
            </div>

            <Footer />
        </>
    );
}
```

2.2.3.Homepage.css:

```
@import
url('https://fonts.googleapis.com/css2?family=Hurricane&display=swap');

.font-hurricane{
    font-family: 'Hurricane', cursive;
    font-size: 24px;
}

.bg-image{
    background-image: url('/public/home.jpg');
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
}
```

2.3. Navbar:

2.3.1. AuthLink:

```
import { useContext, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { Popover } from "@headlessui/react";
import { logoutUser, titleCase } from '../../../utils/helper';
import AuthContext from "../../../store/auth-context";

function ProfileButton() {
    const authCtx = useContext(AuthContext);
    const navigator = useNavigate();

    const handleLogout = async () => {
```

```
        const token = authCtx.token;
        const res = await logoutUser(token);
        if (res.error) {
            alert(res.message);
        }
        else {
            authCtx.logout();
            navigator('/', { replace: true });  // redirect user after
logout
        }
    }

    return (
        <>
            <div className="avatar hidden md:block">
                <Popover className="relative">
                    <Popover.Button>
                        <div className="avatar__icon">
                            <svg xmlns="http://www.w3.org/2000/svg"
className="h-8 w-8" viewBox="0 0 20 20" fill="currentColor">
                                <path fillRule="evenodd" d="M10 9a3 3 0
100-6 3 3 0 000 6zm-7 9a7 7 0 1114 0H3z" clipRule="evenodd" />
                            </svg>
                        </div>
                    </Popover.Button>

                    <Popover.Panel className="absolute z-10">
                        <div className="grid grid-cols-1 bg-white shadow-
2xl border-2 w-48 rounded-lg p-2">
                            <div className="my-2 flex justify-center
items-center font-medium">
                                {`${titleCase(authCtx.user.name)}`}
                            </div>
                            <hr className="bg-black" />

                            <div className="mt-2 flex items-center
justify-center ">
                                <button onClick={handleLogout}
className='text-red-500'>
                                    <svg
xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor" strokeWidth={2}>
                                        <path strokeLinecap="round"
strokeLinejoin="round" d="M17 16l4-4m0 0l-4-4m4 4H7m6 4v1a3 3 0 01-3
3H6a3 3 0 01-3-3V7a3 3 0 013-3h4a3 3 0 013 3v1" />
                                    </svg>
                                </button>
                            </div>
```

```
                    </div>
                </Popover.Panel>
            </Popover>
        </div>

        <div>
            <div className="mt-2 flex items-center justify-center
md:hidden">
                <button onClick={handleLogout} className='text-red-
500'>
                    <svg xmlns="http://www.w3.org/2000/svg"
className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor"
strokeWidth={2}>
                        <path strokeLinecap="round"
strokeLinejoin="round" d="M17 16l4-4m0 0l-4-4m4 4H7m6 4v1a3 3 0 01-3
3H6a3 3 0 01-3-3V7a3 3 0 013-3h4a3 3 0 013 3v1" />
                    </svg>
                </button>
            </div>
        </div>
    </>
  );
}

export function AuthLink(props) {
    return (
        <>
            {
                !props.isLoggedIn
                &&
                <Link to='/signup'>
                    <div className="flex p-2">
                        <div className="mr-2">
                            <svg xmlns="http://www.w3.org/2000/svg"
className="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
                                <path fillRule="evenodd" d="M10 9a3 3 0
100-6 3 3 0 000 6zm-7 9a7 7 0 1114 0H3z" clipRule="evenodd" />
                            </svg>
                        </div>
                        Sign Up
                    </div>
                </Link>

            }

            {
                !props.isLoggedIn
                &&
                <Link to='/login'>
```

```
                    <div className="flex p-2 bg-amber-400 hover:bg-amber-
300 transition-all rounded-2xl">
                        <div className="mr-2">
                            <svg xmlns="http://www.w3.org/2000/svg"
className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor"
strokeWidth={1}>
                                <path strokeLinecap="round"
strokeLinejoin="round" d="M11 16l-4-4m0 0l4-4m-4 4h14m-5 4v1a3 3 0 01-3
3H6a3 3 0 01-3-3V7a3 3 0 013-3h7a3 3 0 013 3v1" />
                            </svg>
                        </div>
                        Login
                    </div>
                </Link>
            }
            {
                props.isLoggedIn
                &&
                <ProfileButton />
            }
        </>
    )
}
```

2.3.2.Banner.js:

```
export function Banner(){
    return(
        <div className="banner flex justify-items-center items-center">
            <img src='/logo512.png' height={48} width={48} alt='banner
pizza' />
            <div className="ml-2 font-medium text-base">
                Mount Pizza
            </div>
        </div>

    );
}
```

2.4. MenuLink:

```
import { Link } from "react-router-dom";


export function MenuLink(props) {
    return (
        <div className="flex flex-col md:flex-row">
            <Link to='/'>
                <div className='mx-5 flex mb-2'>
```

```
                    <div className="mr-2">
                        <svg xmlns="http://www.w3.org/2000/svg"
className="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
                            <path d="M10.707 2.293a1 1 0 00-1.414 0l-7
7a1 1 0 001.414 1.414L4 10.414V17a1 1 0 001 1h2a1 1 0 001-1v-2a1 1 0 011-
1h2a1 1 0 011 1v2a1 1 0 001 1h2a1 1 0 001-1v-6.586l.293.293a1 1 0
001.414-1.414l-7-7z" />
                        </svg>
                    </div>

                    <div>
                        Home
                    </div>
                </div>
            </Link>
            {
                props.role === 'ADMIN'
                &&
                props.isLoggedIn
                &&
                <Link to='/dashboard'>
                    <div className='mx-5 flex mb-2'>
                        <div className="mr-2">
                            <svg xmlns="http://www.w3.org/2000/svg"
className="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
                                <path d="M5 4a1 1 0 00-2 0v7.268a2 2 0
000 3.464V16a1 1 0 102 0v-1.268a2 2 0 000-3.464V4zM11 4a1 1 0 10-2
0v1.268a2 2 0 000 3.464V16a1 1 0 102 0V8.732a2 2 0 000-3.464V4zM16 3a1 1
0 011 1v7.268a2 2 0 010 3.464V16a1 1 0 11-2 0v-1.268a2 2 0 010-3.464V4a1
1 0 011-1z" />
                            </svg>
                        </div>

                        <div>
                            Dashboard
                        </div>
                    </div>
                </Link>
            }
            {
                props.role === 'CUST'
                &&
                props.isLoggedIn
                &&
                <Link to='/menu'>
                    <div className='mx-5 flex mb-2'>
                        <div className="mr-2">
                            <svg xmlns="http://www.w3.org/2000/svg"
className="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
```

```
                                <path fillRule="evenodd" d="M10 2a4 4 0
00-4 4v1H5a1 1 0 00-.994.89l-1 9A1 1 0 004 18h12a1 1 0 00.994-1.11l-1-9A1
1 0 0015 7h-1V6a4 4 0 00-4-4zm2 5V6a2 2 0 10-4 0v1h4zm-6 3a1 1 0 112 0 1
1 0 01-2 0zm7-1a1 1 0 100 2 1 1 0 000-2z" clipRule="evenodd" />
                            </svg>
                        </div>

                        <div>
                            Menu
                        </div>
                    </div>
                </Link>

            }
            {
                props.role === 'CUST'
                &&
                props.isLoggedIn
                &&
                <Link to='/cart'>
                    <div className='mx-5 flex mb-2'>
                        <div className="mr-2">
                            <svg xmlns="http://www.w3.org/2000/svg"
classNam="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
                                <path d="M3 1a1 1 0 000 2h1.22l.305
1.222a.997.997 0 00.01.042l1.358 5.43-.893.892C3.74 11.846 4.632 14 6.414
14H15a1 1 0 000-2H6.414l1-1H14a1 1 0 00.894-.553l3-6A1 1 0 0017 3H6.28l-
.31-1.243A1 1 0 005 1H3zM16 16.5a1.5 1.5 0 11-3 0 1.5 1.5 0 013 0zM6.5
18a1.5 1.5 0 100-3 1.5 1.5 0 000 3z" />
                            </svg>
                        </div>

                        <div>
                            Cart
                        </div>
                    </div>
                </Link>

            }
        </div>
    );
}
```

2.5. PizzaCard:

2.5.1. AddToCardButton:

```
import { useState, useEffect } from 'react';
import { addItemToCart } from '../../../utils/helper';
```

```
// add to cart button
export function AddToCartButton(props) {
    const [quantity, setQuantity] = useState(0);
    const [isAddedToCart, setIsAddedToCart] = useState(false);

    useEffect(() => {
        if (isAddedToCart === true) {
            setQuantity(0);
            setIsAddedToCart(false);
        }
    }, [isAddedToCart])


    const handleAddToCart = async () => {
        if(quantity === 0){
            alert('Please select quantity');
            return;
        }
        const res = await addItemToCart(props.token, props.itemId,
quantity);

        if (res.error) {
            alert(res.message);
        } else {
            alert('Item added in cart');
            setIsAddedToCart(true);
        }
    }

    const increaseQuantity = () => {
        let newQuantity = quantity + 1;
        if (newQuantity > 10)
            newQuantity = 10;

        setQuantity(newQuantity);
    }

    const decreaseQuantity = () => {
        let newQuantity = quantity - 1;
        if (newQuantity < 0)
            newQuantity = 0;

        setQuantity(newQuantity);
    }

    return (
        <div className="flex justify-around my-3">
            <button onClick={handleAddToCart}>
```

```
                <div>
                    <img className="h-10 w-10 hover:-translate-y-1
transition" src='/addToCart.png' alt='add to cart' />
                </div>
            </button>

            <div className="quantityButton flex">
                <button onClick={increaseQuantity}>
                    <div className="quantityButton__increase hover:-
translate-y-1 transition text-lime-500">
                        <svg xmlns="http://www.w3.org/2000/svg"
className="h-10 w-10" viewBox="0 0 20 20" fill="currentColor">
                            <path fillRule="evenodd" d="M10 18a8 8 0 100-
16 8 8 0 000 16zm1-11a1 1 0 10-2 0v2H7a1 1 0 100 2h2v2a1 1 0 102 0v-2h2a1
1 0 100-2h-2V7z" clipRule="evenodd" />
                        </svg>
                    </div>
                </button>
                <div className="quantityButton__quantity border-2 border-
gray-800 flex justify-center items-center rounded-full px-2 mx-2 font-
bold text-lg">
                    {quantity}
                </div>
                <button onClick={decreaseQuantity}>
                    <div className="quantityButton__decrease hover:-
translate-y-1 transition text-red-600">
                        <svg xmlns="http://www.w3.org/2000/svg"
className="h-10 w-10" viewBox="0 0 20 20" fill="currentColor">
                            <path fillRule="evenodd" d="M10 18a8 8 0 100-
16 8 8 0 000 16zM7 9a1 1 0 000 2h6a1 1 0 100-2H7z" clipRule="evenodd" />
                        </svg>
                    </div>
                </button>
            </div>
        </div>
    );
}
```

2.5.2. ToppingCard :

```
export function ToppingCard(props){
    return(
        <div className="mx-2 bg-gray-300 p-1 shadow-md text-sm rounded-
md">
        {props.topping.name}
        </div>
    );
}
```

2.5.3.PizzaCard :

```
// importing components
import { ToppingCard } from "./ToppingCard/ToppingCard";
import { AddToCartButton } from './AddToCardButton/AddToCardButton';

export function PizzaCard(props) {
    return (
        <div className="pizza shadow-2xl rounded-3xl md:mx-3 mb-5 p-2">
            <div className="pizza__image p-3 mb-2">
                <img className="h-48 w-full" src={props.item.imageUrl}
alt={props.item.name} />
            </div>

            <div className="pizza__name flex justify-center items-center
font-bold text-lg text-gray-800 mb-1">
                {props.item.name} <span className="ml-2 font-light">({
props.item.crustType })</span>
            </div>

            <div className="pizza__price flex justify-center items-center
text-base mb-3">
                <span className="mr-1">₹</span>{`${props.item.price}.00`}
            </div>

            <div className="pizza__toppings mb-3 px-2 flex grid-cols-3">
                {
                    props.item.toppings.map(topping => {
                        return <ToppingCard key={topping._id}
topping={topping} />
                    })
                }
            </div>

            <hr />
            <div className="pizza__add mb-1">
                <AddToCartButton itemId={props.item._id}
token={props.token} />
            </div>
        </div>
    );
}
```

2.6.Spinner:

```
export function Spinner() {
    return (
        <>
            <div className="animate-bounce text-red-600">
```

```
                <svg xmlns="http://www.w3.org/2000/svg" className="h-8 w-
8" viewBox="0 0 20 20" fill="currentColor">
                    <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8
0 0116 0zm-8-3a1 1 0 00-.867.5 1 1 0 11-1.731-1A3 3 0 0113 8a3.001 3.001
0 01-2 2.83V11a1 1 0 11-2 0v-1a1 1 0 011-1 1 1 0 100-2zm0 8a1 1 0 100-2 1
1 0 000 2z" clipRule="evenodd" />
                </svg>
            </div>
        </>
    )
}
```

3.Backend Code:

3.1. Item:

3.1.1:Pizzas.js:

```
// importing model
const Pizza = require('../../schemas/Pizza');
const Topping = require('../../schemas/Topping');

// importing errors handlers
const {
    sendError,
    sendSuccess,
} = require('../../utilities/helpers');

// importing status codes
const {
    OK,
    BAD_REQUEST,
    NOT_FOUND
} = require('../../utilities/statusCodes');


const CRUST = {
    SMALL: 100,
    MEDIUM: 150,
    LARGE: 200
}

// function to create pizza
const createPizza = async (req, res) => {
    const {
        name,
        toppings,
        imageUrl,
        crustType,
```

```
    } = req.body;

    // find if there is already a pizza present
    const pizza = await Pizza.findOne({ name: name });
    if (pizza)
        return sendError(res, 'Pizza is already present', BAD_REQUEST);

    // getting all toppings price
    const toppingsList = [];
    let price = 0;
    for (let i = 0; i < toppings.length; i++) {
        const topping = await Topping.findById(toppings[i]);
        price += topping.price;
        toppingsList.push(topping);
    }

    // adding crust price
    if (crustType === 'SMALL') {
        price += CRUST.SMALL;
    } else if (crustType === 'MEDIUM') {
        price += CRUST.MEDIUM;
    } else {
        price += CRUST.LARGE;
    }

    // creating new pizza
    const newPizza = new Pizza({
        name: name,
        toppings: toppings,
        style: "ADMIN",
        imageUrl: imageUrl,
        price: price,
        crustType: crustType,
    });

    // saving pizza
    await newPizza.save();

    return sendSuccess(res, newPizza);
};

// function to update pizza
const updatePizza = async (req, res) => {
    const pizzaId = req.params.id;

    // check if pizza exists
    let pizza = await Pizza.findById(pizzaId);
    if (!pizza)
        return sendError(res, 'Pizza is not found', NOT_FOUND);
```

```
    const {
        toppings,
        crustType
    } = req.body;

    let price = 0;
    // getting all toppings price
    if (toppings.length != 0) {
        const toppingsList = [];
        for (let i = 0; i < toppings.length; i++) {
            const topping = await Topping.findById(toppings[i]);
            price += topping.price;
            toppingsList.push(topping);
        }
    }


    // if crust is given
    if(pizza.crustType != null){
        if (crustType === 'SMALL') {
            price += CRUST.SMALL;
        } else if (crustType === 'MEDIUM') {
            price += CRUST.MEDIUM;
        } else {
            price += CRUST.LARGE;
        }
    }


    // update the pizza
    pizza = await Pizza.findByIdAndUpdate(pizzaId, {
        toppings: toppings,
        price: price,
    });

    return sendSuccess(res, pizza);
};

// function to delete pizza
const deletePizza = async (req, res) => {
    const pizzaId = req.params.id;

    // check if pizza is present
    let pizza = await Pizza.findById(pizzaId);
    if (!pizza)
        return sendError(res, 'Pizza is not present', NOT_FOUND);

    // delete pizza
```

```
    pizza = await Pizza.findByIdAndRemove(pizzaId);
    return sendSuccess(res, pizza);
};

// function to get pizza
const getPizza = async (req, res) => {
    const pizzaId = req.query.id;

    // check if pizza exits
    const pizza = await Pizza.findById(pizzaId);
    if (!pizza)
        return sendError(res, 'Pizza is not found', NOT_FOUND);

    return sendSuccess(res, pizza);
};

// function to getAll pizzas
const getAllPizzas = async (req, res) => {
    const pizzaList = await Pizza.find({}).populate('toppings');

    // check if pizza is present
    if (pizzaList.length === 0)
        return sendError(res, 'No pizza found', OK);

    return sendSuccess(res, pizzaList);
};

module.exports = {
    createPizza,
    updatePizza,
    deletePizza,
    getPizza,
    getAllPizzas,
};
```

3.1.2 :Toppings.js:

```
// importing model
const Topping = require('../../schemas/Topping');

// importing status codes
const {
    OK,
    BAD_REQUEST,
    NOT_FOUND,
} = require('../../utilities/statusCodes');

// importing error handles
```

```
const{
    sendError,
    sendSuccess
} = require('../../utilities/helpers');


// function to create new topping
const createTopping = async (req, res) => {
    const {
        name,
        price,
        imageUrl,
        category
    } = req.body;

    let topping = await Topping.findOne({name: name}).lean();

    // if topping is already present
    if(topping){
        return sendError(res, "Topping already present", BAD_REQUEST);
    }

    // if not present --> create new topping
    topping = new Topping({
        name: name,
        price: price,
        imageUrl: imageUrl,
        category: category,
    });

    await topping.save();
    return sendSuccess(res, topping);
}

// function to get topping by id
const getTopping = async (req, res) => {
    const toppingId = req.query.id;

    const topping = await Topping.findById(toppingId);

    if(!topping){
        return sendError(res, "Topping not found", NOT_FOUND);
    }

    return sendSuccess(res, topping);
}

// function to get all toppings
const getAllToppings = async (req, res) => {
```

```
    const toppingList = await Topping.find({}).lean();

    // if no topping is present
    if(toppingList.length == 0)
        return sendError(res, "Topping is not found", OK);

    return sendSuccess(res, toppingList);
}

// function to update topping
const updateTopping = async (req, res) => {
    const toppingId = req.params.id;
    let topping = await Topping.findById(toppingId).lean();

    // if topping is not present
    if(!topping){
        return sendError(res, 'Topping is not found', NOT_FOUND);
    }

    // updating values
    topping = await Topping.findByIdAndUpdate(toppingId, req.body);
    return sendSuccess(res, topping);

}

// function to delete topping
const deleteTopping = async (req, res) => {
    const toppingId  = req.params.id;
    let topping = await Topping.findById(toppingId).lean();

    // if topping is not found
    if(!topping)
        return sendError(res, 'Topping not found', NOT_FOUND);

    topping = await Topping.findByIdAndRemove(toppingId);
    return sendSuccess(res, topping);
}

module.exports = {
    createTopping,
    getTopping,
    getAllToppings,
    updateTopping,
    deleteTopping,
};
```

3.1.3.Authentication.js:

```
// importing schemas

const User = require('../schemas/User');
const { sendError, sendSuccess, setToken, checkToken } =
require('../utilities/helpers');
const {
    BAD_REQUEST
    , NOT_FOUND,
    FORBIDDEN
} = require('../utilities/statusCodes');


// cb: to login user
module.exports.login = async (req, res, next) => {
    const {
        email,
        password,
    } = req.body;

    const user = await User.findOne({ email: email });

    // check if user exists
    if (user) {
        // check is password is correct
        const validPassword = await user.isValidPwd(password,
user.password);
        if (!validPassword) {
            return sendError(res, 'Wrong email or password',
BAD_REQUEST);
        } else {
            // updating the last login time and date
            await User.findByIdAndUpdate(user._id, {
                lastLoginAt: new Date(Date.now()).toISOString(),
            })

            // generate new token if expired
            const newToken = await user.generateAuthToken(
                user.firstName,
                user.lastName,
                user.role,
                user.email,
            );

            console.log('User is logged in');
            return sendSuccess(res, newToken, newToken);
        }

    } else {
        return sendError(res, 'User not found', NOT_FOUND);
```

```
    }
}


// cb: logout controller
module.exports.logout = async (req, res) => {
    const token = req.header('x-auth-token')

    // delete token
    const tokenStatus = checkToken(token);

    // if token is present
    if (tokenStatus !== 'NOT FOUND') {
        setToken(token);
        console.log('User logged out');
        return sendSuccess(res, "");
    } else {
        return sendError(res, 'Already LoggedOut. Please Login.',
BAD_REQUEST);
    }
}
```

3.1.4.Cart.js:

```
// importing models
const CartSession = require('../schemas/cartSession');
const Cart = require('../schemas/Cart');

// importing error handlers
const {
    sendError,
    sendSuccess,
} = require('../utilities/helpers');

// importing status codes
const { SERVER_ERROR, NOT_FOUND, BAD_REQUEST } =
require('../utilities/statusCodes');
const User = require('../schemas/User');


// cb: add items to cart
module.exports.addItems = async (req, res) => {
    const { itemId, quantity } = req.query;

    // checking if customer exists or not
    const customer = await User.findOne({ email: req.user.email });
```

```
    if (!customer)
        return sendError(res, 'Customer not found', NOT_FOUND);

    // check if there is already a session running for customer
    const session = await CartSession.findOne({ customerId: customer._id
});

    // if session is present
    if (session) {
        // get the session id
        const sessionId = session._id;

        // if item is already present in cart
        const isAlreadyPresent = await Cart.findOne({
            sessionId: sessionId,
            item: itemId,
        });
        if (isAlreadyPresent) {
            return sendError(res, 'Already Present in cart',
BAD_REQUEST);
        }


        // create new cart item
        const newCartItem = new Cart({
            sessionId: sessionId,
            item: itemId,
            quantity: quantity,
        });

        // save item
        await newCartItem.save()
            .then(() => {
                console.log('Item in cart saved');
            })
            .catch(err => {
                return sendError(res, 'Item cannot be saved in cart',
SERVER_ERROR);
            })
    }
    // if session is not present
    else {
        // create new session for user
        const session = new CartSession({
            customerId: customer._id,
        });


        // saving newly created session
```

```
        await session.save()
            .then(async (session) => {
                // session id
                const sessionId = session._id;

                // creating new item for this session
                const newCartItem = new Cart({
                    sessionId: sessionId,
                    item: itemId,
                    quantity: quantity,
                });

                // saving item
                await newCartItem.save()
                    .then(() => {
                        console.log('Item is saved in cart');
                    })
                    .catch(err => {
                        return sendError(res, 'Item cannot be saved in
cart', SERVER_ERROR);
                    })
            })
            .catch(err => {
                return sendError(res, 'Cart session cannot be created',
SERVER_ERROR);
            })
    }

    return sendSuccess(res, { data: 'Item is added in cart' });
};

// cb: get items from cart
module.exports.getItems = async (req, res) => {
    const customer = await User.findOne({ email: req.user.email });
    if (!customer) {
        return sendError(res, 'Customer not found', NOT_FOUND);
    }

    const customerSession = await CartSession.findOne({ customerId:
customer._id });
    if (!customerSession) {
        return sendSuccess(res, []);
    }

    const cartItems = await Cart.find({ sessionId: customerSession._id
}).populate('item');

    return sendSuccess(res, cartItems);
}
```

```
// cb: clear all cart items
module.exports.clearAll = async (req, res) => {
    const customer = await User.findOne({ email: req.user.email });

    // get cart session id
    const cartSession = await CartSession.findOne({ customerId:
customer._id });
    if (!cartSession) {
        sendError('Cart Session not found');
    }

    // delete all cart items for this session
    const deleteCount = await Cart.deleteMany({ sessionId:
cartSession._id });
    if (deleteCount === 0) {
        return sendError(res, 'Cart is not cleared', SERVER_ERROR);
    }

    await CartSession.findByIdAndDelete(cartSession._id);

    return sendSuccess(res, 'Cart Cleared');
}

// cb: update items in cart

// cb: delete items from cart
module.exports.deleteItem = async (req, res) => {
    const { itemId } = req.query;
    const item = await Cart.findByIdAndRemove(itemId);
    if (!item) {
        return sendError(res, 'Item not found', NOT_FOUND);
    }
    return sendSuccess(res, 'Item is deleted');
}
```

3.1.5.Orders.js:

```
// importing modules
const Razorpay = require('razorpay');
const crypto = require('crypto');

// importing schemma
const Order = require('../schemas/Order');
const User = require('../schemas/User');

// importing constants
```

```
const {
    RZP_KEY_ID,
    RZP_KEY_SECRET
} = require('../configs/index');

// importing error handlers
const {
    sendError, sendSuccess
} = require('../utilities/helpers');

// importing status codes
const {
    BAD_REQUEST, NOT_FOUND, SERVER_ERROR
} = require('../utilities/statusCodes');


// function to create new order
const createOrder = async (req, res) => {
    const customer = await User.findOne({ email: req.user.email });
    const { orderPrice } = req.query;

    // check if customer exists
    if (!customer) {
        return sendError(res, 'Customer not found', NOT_FOUND);
    }

    // creating razorpay instance
    const instance = new Razorpay({
        key_id: RZP_KEY_ID,
        key_secret: RZP_KEY_SECRET,
    });

    const options = {
        amount: (orderPrice * 100).toString(),
        currency: 'INR',
        receipt: 'order_rcptid_1',
    }
    // creating an order
    instance.orders.create(options, function (err, order) {
        if (err) {
            return sendError(res, 'Order not placed', BAD_REQUEST);
        }

        return sendSuccess(res, order);
    })
}


// function to verify payment
```

```
const verifyPayment = async (req, res) => {
    const { razorpayOrderId, razorpayPaymentId, razorpaySignature } =
req.body;
    const customer = await User.findOne({email: req.user.email});

    // creating signature
    let body = razorpayOrderId + "|" + razorpayPaymentId;
    var expectedSignature = crypto.createHmac('sha256',
RZP_KEY_SECRET).update(body.toString()).digest('hex');

    // checking if signature is valid
    if (expectedSignature !== razorpaySignature) {
        return sendError(res, 'Signature not valid', BAD_REQUEST);
    }

    // saving order details in database
    const newOrder = new Order({
        customerId: customer._id,
        orderId: razorpayOrderId,
        paymentId: razorpayPaymentId,
        signatureId: razorpaySignature,
    });

    // adding order id into user db
    customer.orderHistory.push(newOrder._id);
    await customer.save(); // save

    // saving order in db
    await newOrder.save();

    return sendSuccess(res, { "signatureIsValid": "true" });
}

module.exports = {
    createOrder,
    verifyPayment
};
```

3.1.6 .User.js: `// importing model`

```
const user = require('../schemas/User');
const { findOneAndUpdate } = require('../schemas/User');
const User = require('../schemas/User');

// importing error handlers
```

```
const {
    sendError,
    setToken,
    sendSuccess,
    deleteToken,
} = require('../utilities/helpers');

// status codes
const {
    OK,
    BAD_REQUEST,
    NOT_FOUND,
} = require('../utilities/statusCodes');

// HASH LENGTH
const {
    USER_HASH_LENGTH
} = require('../configs/index');
const Token = require('../schemas/Token');

// POST: cb for creating user
const createUser = async (req, res) => {
    const {
        firstName,
        lastName,
        email,
        phoneNumber,
        password,
        role,
    } = req.body;

    // checking if this user is already in system
    const user = await User.findOne({
        email: email,
    });

    if (user)
        return sendError(res, "User already registered", BAD_REQUEST);

    // creating new user
    let newUser = new User({
        firstName: firstName,
        lastName: lastName,
        email: email,
        phoneNumber: phoneNumber,
        password: password,
        role: role,
    });
```

```
    // save new user in database
    newUser = await newUser.save();


    // generate token
    // const token = await newUser.generateAuthToken(
    //     firstName,
    //     lastName,
    //     role,
    //     email,
    // );

    // set token entry in database
    // setToken(String(newUser._id), token);

    return sendSuccess(res, 'Signup Successful');
}

// GET: cb for get user by id
const getUserById = async (req, res) => {
    const userEmail = req.user.email;

    const user = await User.findOne({email: email}).lean();

    // if user is not found
    if (!user)
        return sendError(res, "User not found", NOT_FOUND);

    return sendSuccess(res, user);
}

// GET: cb for getting all users
const getAllUsers = async (req, res) => {
    const userList = await User.find({}).lean();

    // if there is no user present
    if (userList.length == 0)
        return sendError(res, "No user found", OK);

    return sendSuccess(res, userList);
}

// UPDATE: cb for updating user
const updateUser = async (req, res) => {
    const userId = req.params.id;
    let user = await User.findById(userId).lean();

    // if user is not found
    if (!user)
```

```
        return sendError(res, "User not found", NOT_FOUND);

    user = await User.findByIdAndUpdate(userId, req.body);
    return sendSuccess(res, user);
}

// DELETE: cb for deleting user
const deleteUser = async (req, res) => {
    const userId = req.params.id
    let user = await User.findById(userId).lean();

    // check if user exists
    if (!user)
        return sendError(res, "User not found", NOT_FOUND);

    // delete token if its there
    if (deleteToken(userId) != "NOT FOUND") {
        console.log('Token is deleted');
    }else{
        console.log('Token is not deleted');
    }

    // delete the user
    user = await User.findByIdAndRemove(userId);
    return sendSuccess(res, user);
}

const makeAdmin = async (req, res) => {
    const userId = req.params.id;
    let user = await User.findById(userId).lean();

    if(!user){
        return sendError(res, 'User not found', NOT_FOUND);
    }

    if(user.role === 'ADMIN'){
        return sendError(res, 'User is already ADMIN', BAD_REQUEST);
    }

    user = await User.findByIdAndUpdate(userId, {
        role: 'ADMIN'
    });

    await user.save();

    sendSuccess(res, 'Role Changed');
}

module.exports = {
```

```
    createUser,
    getUserById,
    getAllUsers,
    updateUser,
    deleteUser,
    makeAdmin
}


// importing model
const user = require('../schemas/User');
const { findOneAndUpdate } = require('../schemas/User');
const User = require('../schemas/User');

// importing error handlers
const {
    sendError,
    setToken,
    sendSuccess,
    deleteToken,
} = require('../utilities/helpers');

// status codes
const {
    OK,
    BAD_REQUEST,
    NOT_FOUND,
} = require('../utilities/statusCodes');

// HASH LENGTH
const {
    USER_HASH_LENGTH
} = require('../configs/index');
const Token = require('../schemas/Token');

// POST: cb for creating user
const createUser = async (req, res) => {
    const {
        firstName,
        lastName,
        email,
        phoneNumber,
        password,
        role,
    } = req.body;

    // checking if this user is already in system
    const user = await User.findOne({
```

```
        email: email,
    });

    if (user)
        return sendError(res, "User already registered", BAD_REQUEST);

    // creating new user
    let newUser = new User({
        firstName: firstName,
        lastName: lastName,
        email: email,
        phoneNumber: phoneNumber,
        password: password,
        role: role,
    });

    // save new user in database
    newUser = await newUser.save();


    // generate token
    // const token = await newUser.generateAuthToken(
    //     firstName,
    //     lastName,
    //     role,
    //     email,
    // );

    // set token entry in database
    // setToken(String(newUser._id), token);

    return sendSuccess(res, 'Signup Successful');
}

// GET: cb for get user by id
const getUserById = async (req, res) => {
    const userEmail = req.user.email;

    const user = await User.findOne({email: email}).lean();

    // if user is not found
    if (!user)
        return sendError(res, "User not found", NOT_FOUND);

    return sendSuccess(res, user);
}

// GET: cb for getting all users
const getAllUsers = async (req, res) => {
```

```
    const userList = await User.find({}).lean();

    // if there is no user present
    if (userList.length == 0)
        return sendError(res, "No user found", OK);

    return sendSuccess(res, userList);
}

// UPDATE: cb for updating user
const updateUser = async (req, res) => {
    const userId = req.params.id;
    let user = await User.findById(userId).lean();

    // if user is not found
    if (!user)
        return sendError(res, "User not found", NOT_FOUND);

    user = await User.findByIdAndUpdate(userId, req.body);
    return sendSuccess(res, user);
}

// DELETE: cb for deleting user
const deleteUser = async (req, res) => {
    const userId = req.params.id
    let user = await User.findById(userId).lean();

    // check if user exists
    if (!user)
        return sendError(res, "User not found", NOT_FOUND);

    // delete token if its there
    if (deleteToken(userId) != "NOT FOUND") {
        console.log('Token is deleted');
    }else{
        console.log('Token is not deleted');
    }

    // delete the user
    user = await User.findByIdAndRemove(userId);
    return sendSuccess(res, user);
}

const makeAdmin = async (req, res) => {
    const userId = req.params.id;
    let user = await User.findById(userId).lean();

    if(!user){
        return sendError(res, 'User not found', NOT_FOUND);
```

```
    }

    if(user.role === 'ADMIN'){
        return sendError(res, 'User is already ADMIN', BAD_REQUEST);
    }

    user = await User.findByIdAndUpdate(userId, {
        role: 'ADMIN'
    });

    await user.save();

    sendSuccess(res, 'Role Changed');
}

module.exports = {
    createUser,
    getUserById,
    getAllUsers,
    updateUser,
    deleteUser,
    makeAdmin
}
```

3.2.Middlewares:

```
const jwt = require('jsonwebtoken');
const { sendError } = require('../utilities/helpers');
const { NOT_AUTHORIZED, FORBIDDEN } =
require('../utilities/statusCodes');

// authentication for all users
module.exports.allAuth = (req, res, next) => {
    const token = req.header("x-auth-token");
    if(!token)
        return sendError(
            res,
            "Access Denied. No token provided",
            NOT_AUTHORIZED
        );

    const decodePayload = jwt.verify(token, process.env.JWT_PRIVATE_KEY);
    req.user = decodePayload;

    return next();
}

// authentication for admin
```

```
module.exports.adminAuth = (req, res, next) => {
    const token = req.header("x-auth-token");
    if(!token)
        return sendError(
            res,
            "Access Denied. No token provided",
            NOT_AUTHORIZED
        );

    const decodePayload = jwt.verify(token, process.env.JWT_PRIVATE_KEY);
    console.log(decodePayload);
    if(decodePayload.role === 'ADMIN'){
        req.user = decodePayload;
        return next();
    }else{
        return sendError(res, "Forbidden", NOT_AUTHORIZED);
    }
};
const jwt = require('jsonwebtoken');
const { sendError } = require('../utilities/helpers');
const { NOT_AUTHORIZED, FORBIDDEN } =
require('../utilities/statusCodes');

// authentication for all users
module.exports.allAuth = (req, res, next) => {
    const token = req.header("x-auth-token");
    if(!token)
        return sendError(
            res,
            "Access Denied. No token provided",
            NOT_AUTHORIZED
        );

    const decodePayload = jwt.verify(token, process.env.JWT_PRIVATE_KEY);
    req.user = decodePayload;

    return next();
}

// authentication for admin
module.exports.adminAuth = (req, res, next) => {
    const token = req.header("x-auth-token");
    if(!token)
        return sendError(
            res,
            "Access Denied. No token provided",
            NOT_AUTHORIZED
        );
```

```
    const decodePayload = jwt.verify(token, process.env.JWT_PRIVATE_KEY);
    console.log(decodePayload);
    if(decodePayload.role === 'ADMIN'){
        req.user = decodePayload;
        return next();
    }else{
        return sendError(res, "Forbidden", NOT_AUTHORIZED);
    }
};
```

3.3 Schemas: Here we are using MongoDB as database.

3.3.1.Cart.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const cartSchema = new Schema({
    sessionId: {
        type: String,
        required: true,
    },
    item: {
        type: Schema.Types.ObjectId,
        ref: 'Pizza',
        required: true,
    },
    quantity: {
        type: Number,
        required: true,
    }
},
{
    timestamps: true,
});

const cart = mongoose.model('Cart', cartSchema);

module.exports = cart;
```

3.3.2. CartSession.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const sessionSchema = new Schema({
    customerId: {
        type: String,
```

```
        required: true,
    },
},
{
    timestamps: true,
});

const cartSession = mongoose.model('cartSession', sessionSchema);

module.exports  = cartSession;
```

3.3.3.Order.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const orderSchema = new Schema({
    customerId: {
        type: Schema.Types.ObjectId,
        required: true,
    },
    orderId: {
        type: String,
        required: true,
    },
    paymentId: {
        type: String,
        required: true,
    },
    signatureId: {
        type: String,
        required: true,
    },
},
{
    timestamps: true,
});

const order = mongoose.model('Order', orderSchema);

module.exports = order;
```

3.3.4.Pizza.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const pizzaSchema = new Schema({
```

```
    name: {
        type: String,
        required: true,
    },
    toppings: {
        type: [
            {
                type: Schema.Types.ObjectId,
                ref: 'Topping'
            }
        ],
        required: true,
    },
    imageUrl: {
        type: String,
        required: true,
    },
    style: {
        type: String,
        required: true,
        enum: ["CUSTOM", "ADMIN"],
        default: "ADMIN",
    },
    price: {
        type: Number,
        required: true,
    },
    crustType: {
        type: String,
        required: true,
        enum: ["SMALL", "MEDIUM", "LARGE"]
    }
},
{
    timestamps: true,
});

const pizza = mongoose.model('Pizza', pizzaSchema);

module.exports = pizza;
```

3.3.5.Token.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const tokenSchema = new Schema({
    value: {
```

```
        type: String,
        required: true,
    },
});

const Token = mongoose.model('Token', tokenSchema);

module.exports = Token;
```

3.3.6.Topping.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const toppingSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    price: {
        type: Number,
        required: true,
    },
    imageUrl: {
        type: String,
        required: true,
    },
    category: {
        type: String,
        enum: ["VEG", "NONVEG"],
        default: "VEG",
    }
},
{
    timestamps: true,
});

const topping = mongoose.model('Topping', toppingSchema);

module.exports = topping;
```

3.3.7.User.js:

```
// importing modules
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
```

```
const { JWT_PRIVATE_KEY, USER_HASH_LENGTH } =
require('../configs/index');

const { generateHash } = require('../utilities/helpers');

// user schema
const userSchema = new Schema({
    firstName: {
        type: String,
        required: true,
    },
    lastName: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
    },
    phoneNumber: {
        type: String,
        required: true,
    },
    password: {
        type: String,
        required: true,
    },
    role: {
        type: String,
        default: 'CUST',
        enum: ['CUST', 'ADMIN'],
        required: true,
    },
    orderHistory: {
        type: [Schema.Types.ObjectId],
        ref: 'Order',
    },
    lastLogin: { type: Date, default: Date.now },
    lastActiveAt: { type: Date },
},
    { timestamps: true },
);

// function before saving
userSchema.pre("save", async function(next){
    this.firstName = String(this.firstName).toLowerCase();
    this.email = String(this.email).toLowerCase();
    if(! this.isModified("password")) return next();
```

```
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(this.password, salt);
    this.password = hash;

    next();
});

// function to check if password is valid
userSchema.methods.isValidPwd = async (password, userPassword) => {
    console.log(password);
    const isMatchPwd = await bcrypt.compare(password, userPassword);
    return isMatchPwd;
}

// function to generate unique token
userSchema.methods.generateAuthToken = async (firstName, lastName, role,
email) => {
    const token = jwt.sign(
        {
            name: `${firstName} ${lastName}`,
            email: email,
            role: role,
        },
        JWT_PRIVATE_KEY,
    );

    return token;
}


// user model
const user = mongoose.model('User', userSchema);

module.exports = user;
```

CHAPTER 6

SOFTWARE TESTING

6.1. Software Testing

Software testing is a critical element of software quality assurance and represents the ultimate reuse of specification. Design and code testing represents interesting anomaly for the software during earlier definition and development phase, it was attempted to build software from an abstract concept to tangible implementation.

The testing phase involves, testing of the development of the system using various techniques such as White Box Testing, Control Structure Testing.

6.2. Testing Strategies:

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level against customer requirements.

- Unit Testing
- Integration Testing
- System Testing.
- Acceptance Testing.

6.3. Testing Techniques:

There are some techniques in testing.

6.3.1. White Box Testing:

White box testing is a test case design method that uses the control structure of the procedural design to derive test cases.

6.3.1.1. Control Structure Testing

The following tests were conducted and it was noted that the BCBS is performing them well.

- Basic path Testing
- Condition Testing
- Data Flow Testing
- Loop Testing

- **Implementation:**

There are many type of testing including in white box and black box testing.

6.3.1.2. Usability Testing:

- Usability testing is nothing but the User-friendliness check.
- In Usability testing, the application flow is tested so that a new user can understand the application easily.
- Basically, system navigation is checked in Usability testing.

- **Usability Test Cases:**

- Web page content should be correct without any spelling or grammatical errors.
- All fonts should be same as per the requirements.
- All the text should be properly aligned.
- All the error messages should be correct without any spelling or grammatical errors and the error message should match with the field label.
- Tool tip text should be there for every field.
- All the fields should be properly aligned.
- Enough space should be provided between field labels, columns, rows, and error messages.
- All the buttons should be in a standard format and size.
- Home link should be there on every single page.

If there is an error message on submit, the information filled by the user should be there.

- Title should display on each web page
- All fields (Textbox, dropdown, radio button, etc) and buttons should be accessible by keyboard shortcuts and the user should be able to perform all operations by using keyboard.
- Check if the dropdown data is not truncated due to the field size. Also, check whether the data is hardcoded or managed via administrator

6.3.1.3. Compatibility Testing-

Compatibility testing is used to determine if your software is compatible with other elements of a system with which it should operate, e.g. Browsers, Operating Systems, or hardware.

- **Compatibility Test Scenarios:**
  - Test the website in different browsers (IE, Firefox, Chrome, Safari and Opera) and ensure the website is displaying properly.
  - Test the HTML version being used is compatible with appropriate browser versions.
  - Test the images display correctly in different browsers.
  - Test the fonts are usable in different browsers.
  - Test the java script code is usable in different browsers.

6.3.1.4. Database Testing:

In Database testing backend records are tested which have been inserted through the web or desktop applications.

The data which is displaying in the web application should match with the data stored in the Database.

- **Test Cases for Database Testing:**
  - Verify the database name: The database name should match with the specifications.

Verify the Tables, columns, column types and defaults: All things should match with the specifications.

- Verify whether the column allows a null or not.
- Verify the Primary and foreign key of each table.
- Verify the Stored Procedure:
- Test whether the Stored procedure is installed or not.
- Verify the Stored procedure name
- Verify the parameter names, types and number of parameters.
- Test the parameters if they are required or not.
- Test the stored procedure by deleting some parameters Test when the output is zero, the zero records should be affected.
- Test the stored procedure by writing simple SQL queries.
- Test whether the stored procedure returns the values Test the stored procedure with sample input data.
- Verify the behavior of each flag in the table.
- Verify the data gets properly saved into the database after each page submission.
- Verify the data if the DML (Update, delete and insert) operations are performed.
- Check the length of every field: The field length in the back end and front end must be same.
- Verify the database names of QA, UAT and production. The names should be unique.
- Verify the encrypted data in the database.
- Verify the database size. Also test the response time of each query executed.
- Verify the data displayed on the front end and make sure it is same in the back end.
- Verify the data validity by inserting the invalid data in the database.

Verify the Triggers.

6.3.1.5. What is Security Testing?

Security Testing involves the test to identify any flaws and gaps from a security point of view.

- **Test Scenarios for Security Testing:**

  - Verify the web page which contains important data like password, credit card numbers, secret answers for security question etc should be submitted via HTTPS (SSL).
  - Verify the important information like password, credit card numbers etc should display in encrypted format.
  - Verify password rules are implemented on all authentication pages like Registration, forgot password, change password.
  - Verify if the password is changed the user should not be able to login with the old password.
  - Verify the error messages should not display any important information.
  - Verify if the user is logged out from the system or user session was expired, the user should not be able to navigate the site.
  - Verify to access the secured and non-secured web pages directly without login.
  - Verify the “View Source code” option is disabled and should not be visible to the user.
  - Verify the user account gets locked out if the user is entering the wrong password several times.
  - Verify the cookies should not store passwords.
  - Verify if, any functionality is not working, the system should not display any application, server, or database information. Instead, it should display the custom error page.
  - Verify the SQL injection attacks.
  - Verify the user roles and their rights. For Example, the requestor should not be able to access the admin page.
  - Verify the important operations are written in log files, and that information should be traceable.

- Verify the session values are in an encrypted format in the address bar.

- Verify the cookie information is stored in encrypted format.
- Verify the application for Brute Force Attacks

CHAPTER 7

MAINTENANCE

7.1. Introduction

Software maintenance refers to the process of modifying and updating a software system after it has been delivered to the customer. This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.

The goal of software maintenance is to keep the software system working correctly, efficiently, and securely, and to ensure that it continues to meet the needs of the users.

Maintenance is required to maintain a building's initial performance capacity. Without maintenance, performance will not meet the demand and eventually will drop below the limit of acceptance of residents. In practice, both the demand and the limit of acceptance will gradually rise over time as a result of improved technology, rising standards, and growing prosperity.

Improvement and renewal are required to answer the accordingly rising expectations. As a result, the total life cycle costs will generally be a multiple of the initial building costs. Maintenance is a combination of all technical and associated administrative actions during the service life to retain a building or its parts in a state in which it can perform its required functions.

7.2. Types of website maintenance services :

7.2.1. Updating Website Software:

If you are using any kind of content management system like Word Press or any kind of script like PHP, it needs to always be up to date. Word Press usually updates itself automatically but there are instances where it might need to be updated manually.

Additionally, if you are using a website host, a web service maintenance provider will ensure that they are updating their core server software on a regular basis. For instance, they will check if things like FTP service running on the server are up to date. If however, you run your own web server, they will manually test and apply updates.

7.2.2. Improving Website Speed:

Did you know that twenty five percent of users navigate out of a website if it takes more than three seconds to load? Page loading is obviously an important part of the user experience. However, we tend to let it slide in order to accommodate new nifty functionality or aesthetic website design.

In reality, web visitors do not care about all the bells and whistles. They care about accessing the information they want as fast as possible. Another important factor to consider is how page load speed affects your search engine rankings. A slow website increases bounce rate and if people are navigating out of your web page as fast as they got there, then Google will think your website isn't important so they will push it further down the rankings.

7.2.2.1. Fixing HTML Errors :

HTML Code isn't easy especially if you know nothing about website coding. You need to pay attention to detail to ensure that it's working well. What might seem like a minor error could lead to a number of problems such as funky looking pages, missing multimedia, etc. Reliable website support should be able to inspect, find and fix issues related to HTML.

7.2.2.2. Backing up Files:

Imagine how frustrating it would be if you were to lose all your website's essential files.

This includes images, pages, blog posts, plugging, etc. If you run your website on Word Press, you might have a few automated backup options. Website maintenance services ensure that file and database backups is automatically performed at least on a weekly basis.

7.2.2.3. Developing New Content:

Your content strategy might be on point but it's always a good idea to improve it by creating new types of content. Your web content also needs to be up to date. If you wrote a post that was published years ago, it might not be relevant right now and some of the information might need to be updated.

7.2.2.4. Search Engine Optimization:

Regularly checking your website for areas where search ability can be improved can go a long way to improving search engine optimization. Your blog posts can be altered to include popular and competitive keywords. Your landing pages can also be updated to include more relevant and timely keywords. Even you're "About Us" page can be updated regularly to take advantage of popular keywords in your industry. Keyword research is also important when creating new content for your website.

As part of the SEO strategy, it's also important to identify structural issues within your website that may affect how search engines view your site.

7.2.2.5. Ensuring Design Consistency across All Pages:

Your website should function properly in all the latest versions of major website browsers like Chrome, Safari, Firefox, etc. These browsers update frequently and if your website doesn't adapt to the updates it might not show up or it might not look the way you want it to. This could negatively affect your business and you're branding. It's also important to ensure that you have a responsive website design that looks good on mobile devices.

Another factor to consider about your website's design is uniformity. If you have a lot of branding elements that represent your company, they should be included uniformly across all your pages.

7.3. Fixing Broken Links:

Every link on your website should lead exactly to the right place but sometimes links can go bad. A site or resource you linked to earlier might disappear, a page on your website might become unavailable or you might move a post and forget to update others that link to it.

Broken links can be detrimental to your website. They could give visitors the impression that:

- Your website is not user friendly
- You don't regularly update your website
- You are not credible

7.4. Reviewing Your Website Analytics:

Your website's performance needs to be monitored to ensure that everything is working as it should. An experienced website support specialist will examine

high level metrics like how many new and returning visitors your website has received for the past week or which blog posts are the most read. They will then make adjustments your website accordingly to ensure all the metrics are at their peak.

This is just a minimum starting point of what website maintenance is all about. Some of these tasks can be overly time consuming. Additionally, if you do not know what you are doing, you could end up doing more damage. The better option is to hire professional website maintenance services.

CHAPTER 8

LITERATURE REVIEW

8.1. Introduction :

The literature review comprises of a selective and critical survey of the written works of this particular subject area. It includes personal communication, articles, books, published and unpublished papers, and unpublished works to be limited. All the authors have been acknowledged. It is as follows:

8.2. Theoretical Review

In today's age of fast food and take-out, many restaurants have chosen to focus on quick preparation and speedy delivery of orders rather than offering a rich dining experience. Until very recently, all of these delivery orders were placed over the phone, but there are many disadvantages to this system, including the inconvenience of the customer needing to have a physical copy of the menu, lack of a visual confirmation that the order was placed correctly, and the necessity for the restaurant to have an employee answering the phone and taking orders. According to (US9760958 B2, 2017) Techniques for restaurant transaction processing are provided. A handheld device of a waiter is used to automatically associate a check with a table at a restaurant and to recall and modify that check. In an embodiment, two waiters use one or more handheld devices to automatically transfer a customer's check to one another. According to(US20140330671 A1, 2014) a system and method for automatically submitting an online order from a customer to a restaurant will be more efficient. Input data and customer data is used by an order engine to select a deployment platform, such as social media networks, search engines, mobile applications, and related websites, for a user interface. The user interface automatically populates the restaurant's menu options and business data, allowing the customer to build an order. The order engine submits the order to the restaurant via a non-verbal communication platform. An automated confirmation call is generated to the restaurant confirming receipt of the order. From the confirmation call the restaurant may choose to repeat the message, accept the order, connect to the customer, connect to the service provider, decline the order, or opt-out. The order engine allows the restaurant to monitor online orders and to enroll in the above services for subsequent online orders. This invention is a system and method for managing restaurant customer data elements according to (US8799083 B1,

2014). According to (US6415555 B1,2002) a kiosk system and method is provided for accepting and processing customer orders and payments in a retail environment. The kiosk system and method is particularly applicable to the restaurant business and may include a consumer display screen for visually displaying product information of

products that can be ordered at the kiosk, structure that is operable by the consumer for placing a consumer order composed of at least one product selected from products for which information is displayed on the consumer display screen and structure for accepting payment for the order from the consumer, which typically will include a cash payment by the consumer. Also according to (US5907275 A, 1999) An order communication system is provided which allows audio visual interactive communication between a customer at a remote order station and an attendant receiving orders in a restaurant. The order communication system allows flexible display programming by restaurant employees and in particular allows the customer to view both a textual description and a graphical depiction of ordered items. The order communication system also allows the customer to view live video of the attendant, while allowing the attendant to view live video of the customer as well as the video image the customer is seeing. A method and system for providing an automated, extremely efficient, restaurant experience for the customers of a sit-down style restaurant. The system and method of the present invention presents options to the customer which include, but are not limited to: a virtual server as part of the user interface allowing a customer to have a more human-like interaction with the restaurant communication system; a datamining analysis tool for analyzing transactions performed by the restaurant communication system; functionality to allow the restaurant diner to pay for the food and drinks with cash, a check, credit card, or a gift certificate; Internet access to the restaurant diner for receiving information on movies; the option to purchase movie theatre tickets or gift certificates; a module to place an order from a remote location via the Internet; an incentive program to encourage the restaurant diner to order the food and the drinks; human resource capability for a restaurant; options to send Internet e-mail messages or messages to other diners; or voice recognition and voice synthesis to allow the restaurant diner to operate the restaurant communication system even with a vision impairment. The extreme versatility of the various embodiments of the present invention facilitate providing a highly customized system for any given restaurant business according to (US20030078793 A1, 2003)

8.3. System Review

According to (US20090167553 A1, 2009),

The invention relates to a new “mobile and online based ordering and reservation system” by integrating, synchronizing and utilizing the capability of wireless devices, Internet servers, business web sites and service aggregation portals, to automate the mobile and online ordering and reservation processes in real time, and therefore offering a plural of new mobile and online services and applications. More specifically, the Internet server serves as the host to the service aggregation and as the intermediary device between customers and business. It automatically synchronizes all of the service requests and responses. Therefore, this invention takes full advantage of the flexibility, mobility, availability and convenience of the wireless devices, and the reliability, scalability, the huge processing power of the Internet servers, and the great broadband

penetration of the computers into businesses and consumers. The unique and novel, mobile and online based ordering and reservation platform and system, for such time sensitive services, provides the mobile phone and Internet users and various types of business owners with comprehensive sets of options, including the mobile phones installed with the open source “Android” software platform developed by the “Open Handheld Alliance(OHA), such as the “g Phone” released by “Google”, to deliver the requests and responses automatically and instantly through means of synchronization between mobile and Internet communications. Based on the open source mobile phone platform and the Internet server infrastructure, an intuitive and easy-to-use mobile phone and online based ordering and reservation management system is uniquely defined in the architecture of the current invention to allow both the business owners and end consumers with real time communications for a plural of mobile and online ordering and reservation services.

8.4. Critique of the existing system.

The current mode of placing orders for a pizza for home delivery is via phone. The process seems easy to use but sometimes there is some miscommunication. Due to the fact that there is no visual menu shown during a phone call, 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 and also takes a lot of time of the pizzeria staff. It would be much more comfortable for the customers to have an online pizza ordering system. This is because it would be hassle free for users as they can select the pizzas they want and make payment upon delivery. Also, it will reduce the purchasing time for customers.

8.4.1. Weaknesses of the current system ....

- Inconvenience of customer needing to have a physical copy of the menu
- Time consuming
- Lack of visual confirmation that the order was placed correctly
- Necessity for restaurant to have an employee answering the phone and taking orders
- Difficulty in tracking customers past history
- Manual work and consumes large volumes of data
- Lack of data security

8.5. Summary

After the study of the existing system was done, the data collected was analysed and used to determine different requirements of the developed system. Data flow diagrams and ER diagrams were used in the analysis of the data collected for the proposed System.

CHAPTER 9

FUTURE SCOPE

The food industry is a combination of many diverse businesses and it is responsible for feeding the world population. This group excludes hunter-gatherers and those who do subsistence farming.

Parts of the food industry include agriculture, online food service, and much more. Since the growth of the food industry is assured, anxious promoters can invest their money in the food industry will reap benefits. Under food service, there are many places where these promoters can invest money in. Before, people used to buy food either directly from the restaurants or order over the phone. However, this has changed and people have started ordering online.

CHAPTER 10

CONCLUSION

Our project is only a humble venture to satisfy the needs to manage their project work several user-friendly coding have also adopted. this package shall prove to be a powerful package in satisfying all the requirements of the school. the objective of software planning is to provide a frame work that enables the manger to make reasonable estimates made within a limited time frame at the beginning of the software project and should be updated regularly as the project progresses.at the end it is concluded that we have made effort on following points...

- A description of the background and context of the project and its relation to work already done in the area.
- Made statement of the aims and objectives of the project.
- The description of purpose. scope, and applicability
- We define the problem on which we are working in the project.
- We describe the requirement specifications of the system and the actions that can be done on these things.
- We understand the problem domain and produce a model of the system, which describes operations that can be performed on the system.
- We included features and operations in detail, including screen layouts

REFERENCES

1 Colby, J. (2003). The Intranet As Communications Platform. In: Practical Intranet Development. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-5354-9_10

2 Segal, R.B., Kephart, J.O.: MailCat: An Intelligent Assistant for Organizing E-Mail. In: Proceedings of the 3rd International Conference on Autonomous Agents, Autonomous Agents 1999, Seattle, WA, USA (1999)

3 Takkinen, J., Shahmehri, N.: CAFÉ: A Conceptual Model for Managing Information in Electronic Mail. Copyright 1998 IEEE. Published in the Proceedings of the Hawaii International Conference on System Sciences, HICSS 1998, Kona, Hawaii, January 6-9 (1998)

4 Begbie, R., Chudry, F. The Intranet Chaos Matrix: A conceptual framework for designing an effective knowledge management intranet. J Database Mark Cust Strategy Manag 9, 325–338 (2002). https://doi.org/10.1057/palgrave.jdm.3240081

5 Weik, M.H. (2000). intranet. In: Computer Science and Communications Dictionary. Springer, Boston, MA. https://doi.org/10.1007/1-4020-0613-6_9536

6 Gennai, F., Martusciello, L., Buzzi, M.: A certified email system for the public administration in Italy. In: IADIS International Conference WWW/Internet, vol. 2, pp. 143–147 (2005)

7 Gao, Y., Song, J., Gao, J., Wang, J. (2022). The Research of Spam-Mail Governance Based on the Big Data Analysis of Enterprise E-mail System. In: Li, X. (eds) Advances in Intelligent Automation and Soft Computing. IASC 2021. Lecture Notes on Data Engineering and Communications Technologies, vol 80. Springer, Cham. https://doi.org/10.1007/978-3-030-81007-8_137

8 von Behren, J.R., Czerwinski, S., Joseph, A.D., Brewer, E.A., Kubiatowicz, J.: NinjaMail: the Design of a High-Performance Clustered, Distributed E-mail System. In: Proceeding of International Workshops on Parallel Processing 2000, Toronto, Canada, August 21–24, pp. 151–158 (2000)

9 Christenson, N., Bosserman, T., Beckemeyer, D.: EarthLink Network, Inc.: A Highly Scalable Electronic Mail Service Using Open Systems. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California (December 1997)

10 Saito, Y., Bershad, B.N., Levy, H.M.: Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service. In: 17th ACM symposium on Operating System Review, December 1999, vol. 34(5), pp. 1–15 (1999)

11 Bertolotti, L., Calzarossa, M.C.: WORKLOAD CHARACTERIZATION OF MAIL SERVERS. In: The proceedings of SPECT 2000, Vancouver, Canada, July 16–20 (2000)

Certified E-Mail Scheme, In ISOC 2001 Network and Distributed System Security Symposium (NDSS'01), San Diego, CA, USA, Feb 2001.

BIBLOGRAPHY

Material UI: - https://mui.com/material-ui/getting-started/overview/

React:https://reactjs.org/docs/add-react-to-a-website.html
https://madewithreact.com/tag/dashboards/ https://reactjsexample.com/tag/website/
Node JS: - https://nodejs.org/en/