

Food Ordering Website (ZAYKAA)

*Submitted in partial fulfillment of the requirements
for the award of the degree of*

Bachelor of Computer Applications (BCA)

To

Guru Gobind Singh Indraprastha University, Delhi

Guide:

Ms. Suman Singh
(Assistant Professor)

Submitted By:

Anmol Garg (00821102020)
Harsh Mahori (02221102020)



**Institute of Information Technology & Management,
New Delhi – 110058
Batch (2020-2023)**

Certificate

We, **Anmol Garg (00821102020) & Harsh Mahori (02221102020)** certify that the Minor Project Report (BCA-357) entitled “**Food Ordering Website**” is done by us and it is an authentic work carried out by us at Institute of Information, Technology and Management. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the Students

Date: 26-12-2022

Certified that the Project Report (BCA-357) entitled “**Food Ordering Website**” done by the above students is completed under my guidance.

Signature of the Guide:

Date: 26-12-2022

Name of the Guide: Ms. Suman Singh

Designation: Assistant Professor

Countersign
Head of the Department

Countersign
Director

ACKNOWLEDGEMENT

This is a great opportunity to acknowledge the remarkable contribution of all the persons whose support makes this project “**Food Ordering Website**” a reality.

We would like to add a few heartfelt words for the people who were part of this project in numerous ways.

It is my privilege to express deep sense of gratitude to my guide **Ms. Suman Singh, Assistant Professor, Institute Of Information Technology and Management, GGSIPU, New Delhi**, for her exemplary guidance, monitoring and encouragement throughout the course of this project. The blessing, benevolent discipline and guidance given by her time shall carry us a long way in the journey of life on which we about to embark.

We deeply indebted to all teaching and non-teaching staffs of our Institute for their continuous support and motivation.

Last but not the least, we would like to acknowledge the constant support of my parents and family members, friends whose patience and encouragement during project completion helped in making this project a reality.

Name: Anmol Garg (00821102020)

Harsh Mahori (02221102020)

TABLE OF CONTENTS

S No	Topic	Page No
1.	Certificate	-
2.	Acknowledgements	-
3.	List of Tables/Figures/Symbols	-
4.	Synopsis	1-5
5.	Chapter-1: Introduction	6-13
6.	1 Description of the Organization	6, 7
7.	2 Software Requirement Specification	8
8.	3 Overview	9
9.	4 Assumptions and Dependencies	9
10.	5 System Interfaces	9
11.	6 User Interfaces	10, 11
12.	7 Memory Constraints	11
13.	8 Specific Requirements	11-13
14.	Chapter-2: System Design	14-25
15.	1 Physical Design	14
16.	2 Block Diagram	14
17.	3 Use Case Diagram	15
18.	4 Data Flow Diagram	15-20
19.	5 Entity Relationship Diagram	21-24
20.	6 Interface Design	24, 25
21.	Chapter-3: System Development & Implementation	26-77
22.	1 Programme Development	26-69
23.	2 Testing & Debugging	69-76
24.	3 Project Planning Activities	77
25.	Chapter-4: Scope of Improvement, Summary and Conclusions	78
26.	1 Scope of Improvement	78
27.	2 Summary	78
28.	3 Conclusion	78
29.	Appendices	79

LIST OF FIGURES

Figure No	Title	Page No
1	Prototype model explaining the working of website	3
2	Gantt Chart	5
1.1	Interaction of Product with the environment	9
2.1	Block Diagram	14
2.2	Use Case Diagram	15
2.3	Level-0 DFD	16
2.4	Level-1 DFD	17
2.5	Level-2 DFD (Manage Category)	18
2.6	Level-2 DFD (Manage Item)	19
2.7	Level-2 DFD (Manage Order)	20
2.8	ERD	21
2.9	Schema of Table	22
3.1	Registration Form	74

LIST OF TABLES

Table No	Title	Page No
1	Hardware Specifications	1
2	Software Specifications	2
1.1	Member Wise Distribution Table	4
2.1	User Login Database	22
2.2	Admin Login Database	22
2.3	Admin Table	23
2.4	User Table	23
2.5	Category Table	23
2.6	Items Table	24
2.7	Orders Table	24
3.1	Functional test cases of order process of a food ordering system	70
3.2	Checklist for Testing User Interfaces	72
3.3	Navigation Test Cases	73
3.4	Test Cases of Registration Form	75
3.5	Sample Test Cases Based on a User Operation	76

SYNOPSIS

1. INTRODUCTION OF THE PROJECT:

Online food ordering is a system for delivering a food restaurant to users using websites. If any user wants to get food from restaurant at home, without travelling and not waste time he/she order food online using Zaykaa. Nowadays more people are ordering food from a restaurant instead of cooking at home. The online ordering system is helpful for both user and restaurant owners. Our platform connects customers and restaurant to serving their multiple needs. Customers use our platform to search and discover food item, read and write customer generated reviews and order food online.

2. OBJECTIVE:

- To provide 24*7 food ordering service to customers
- Remove the need for going to restaurant physically

3. SCOPE:

- Provides the searching for food ordering
- Be easy to understand by the user and operator
- Provide a good user interface
- To provide a platform for easy record maintenance

4. SYSTEM STUDY:

4.1 HARDWARE SPECIFICATIONS:

Categories	Requirement
RAM	2 GB (and higher)
HDD	5 GB (and higher)
Processor	i3 (and higher)

Table 1: Hardware requirement

4.2 SOFTWARE SPECIFICATIONS:

Software	Requirement
Operating System	Windows (7 and higher), Mac (10.12 and higher), Linux (ubuntu 14.10)
Front-End	Html 5, CSS 3, React 16.8.6, Express 3.18.1, Bootstrap 5, Tailwind CSS 3.1.7
Back-End	MongoDB 5.0, NodeJS 16.14.2

Table 2: Software requirement

5. METHODOLOGY:

The methodology employed in an experiment is essential to its success, and bad methodology has spoiled thousands of research projects. So, whenever a piece of research is published in a scientific or medical journal, the researchers always carefully describe their methodology; otherwise, other scientists couldn't possibly judge the quality of what they've done.

5.1 SDLC MODEL:

Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

Prototype Model: The prototyping model starts with the requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.

A '**quick design**' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.

Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continue until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

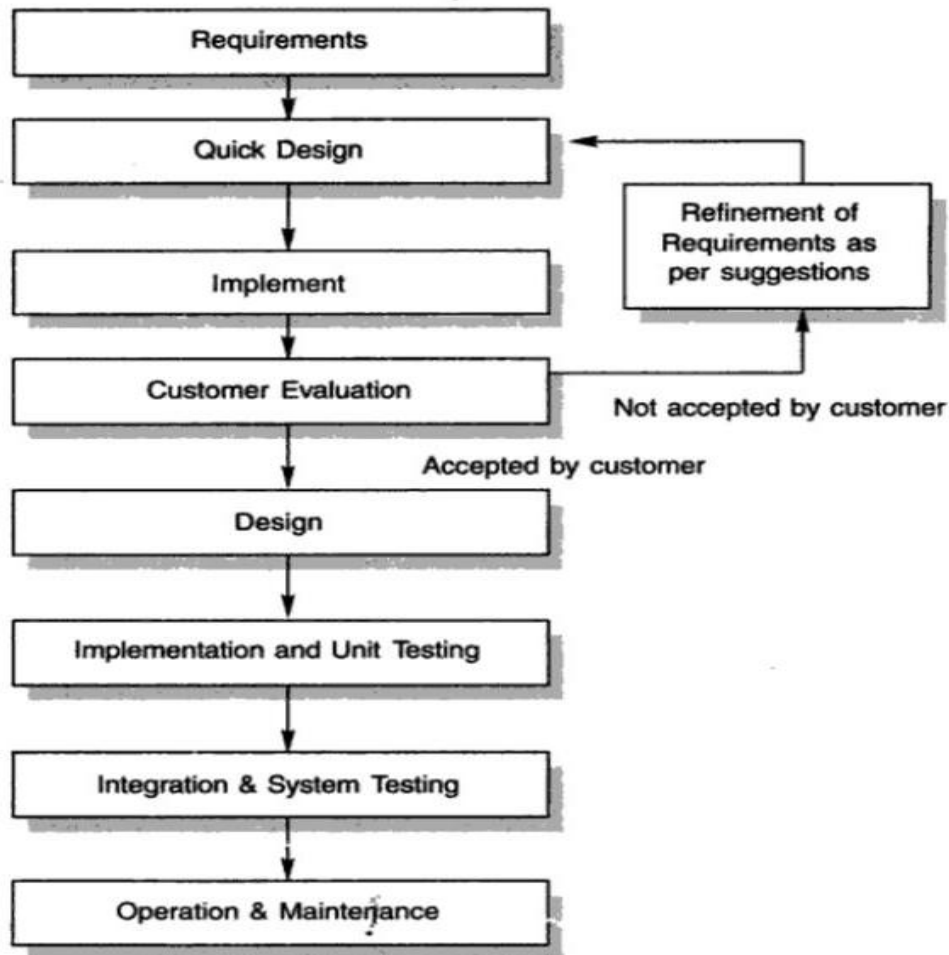


Fig 1: Prototype model explaining the working of website

5.2 DATA COLLECTION METHODS:

The website which we referred is swiggy.com & zomato.com.

Data collection is the process of gathering and measuring information on variables of interest, in an established systematic fashion that enables one to answer stated research questions, test hypotheses, and evaluate outcomes. The data collection component of research is common to all fields of study including physical and social sciences, humanities, business, etc. While methods vary by discipline, the emphasis on ensuring accurate and honest collection remains the same.

Types of data collection:

1. Primary data, new information collected specifically for your purposes, directly from people in the know. Methods of primary data collection vary based upon the goals of the research, as well as the type and depth of information being sought.
2. Secondary data is public information that has been collected by others. It is typically free or inexpensive to obtain and can act as a strong foundation to any research project

6. PROJECT PLANNING ACTIVITIES:

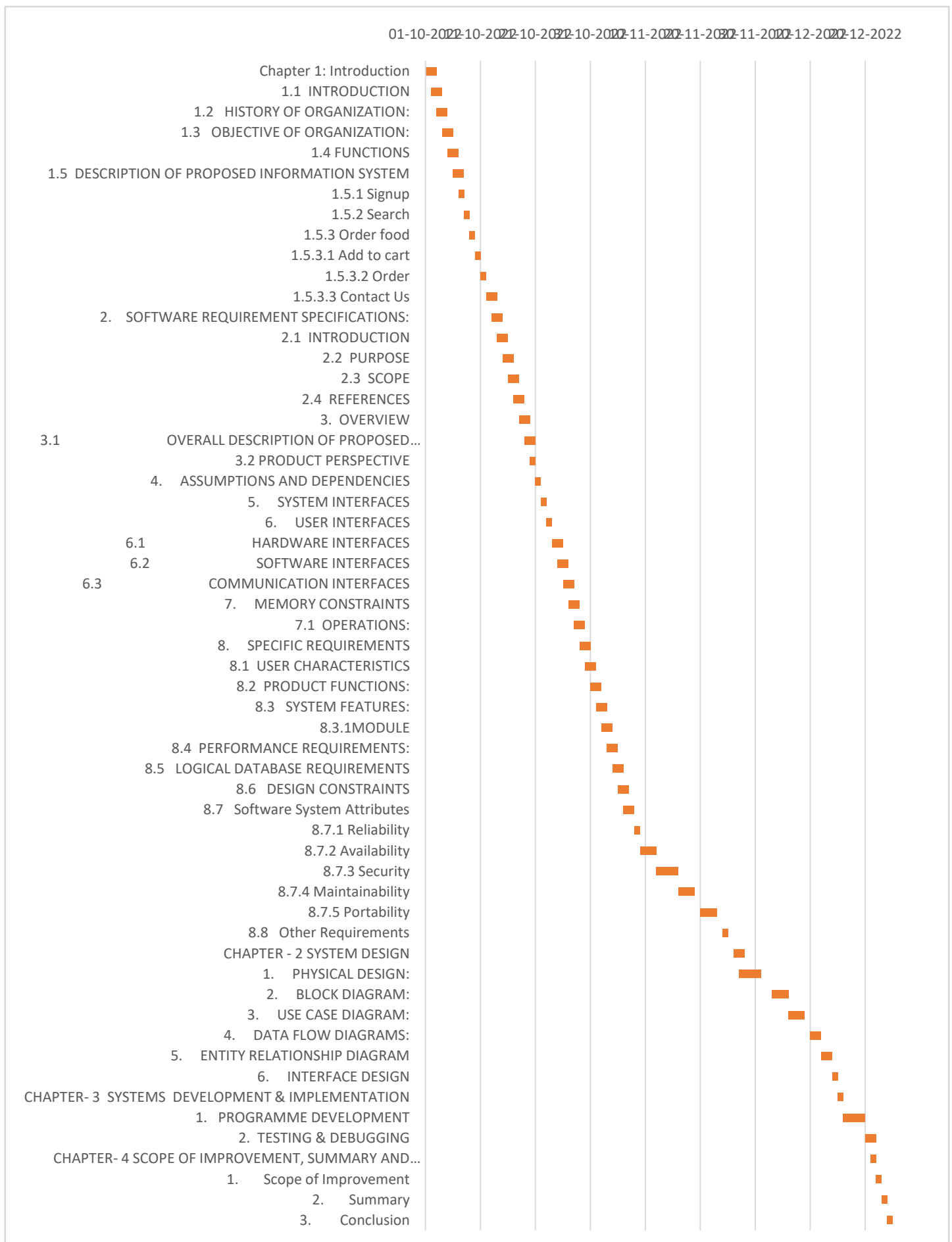
Project planning is part of project management, which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment. Project planning can be done manually or by the use of project management software.

A Gantt chart is a type of bar chart that illustrates a project schedule. This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements constitute the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities.

6.1 TEAM-MEMBER WISE WORK DISTRIBUTION TABLE:

Task	Start Date	Due Date	Duration	Task Prog	Assignee
Chapter 1: Introduction	01-10-2022	03-10-2022	2	100%	Anmol
1.1 INTRODUCTION	02-10-2022	04-10-2022	2	100%	Harsh
1.2 HISTORY OF ORGANIZATION:	03-10-2022	05-10-2022	2	100%	Anmol
1.3 OBJECTIVE OF ORGANIZATION:	04-10-2022	06-10-2022	2	100%	Harsh
1.4 FUNCTIONS	05-10-2022	07-10-2022	2	100%	Anmol
1.5 DESCRIPTION OF PROPOSED INFORMATION SYSTEM	06-10-2022	08-10-2022	2	100%	Harsh
1.5.1 Signup	07-10-2022	08-10-2022	1	100%	Anmol
1.5.2 Search	08-10-2022	09-10-2022	1	100%	Harsh
1.5.3 Order food	09-10-2022	10-10-2022	1	100%	Anmol
1.5.3.1 Add to cart	10-10-2022	11-10-2022	1	100%	Anmol
1.5.3.2 Order	11-10-2022	12-10-2022	1	100%	Anmol
1.5.3.3 Contact Us	12-10-2022	14-10-2022	2	100%	Harsh
2. SOFTWARE REQUIREMENT SPECIFICATIONS:	13-10-2022	15-10-2022	2	100%	Anmol
2.1 INTRODUCTION	14-10-2022	16-10-2022	2	100%	Harsh
2.2 PURPOSE	15-10-2022	17-10-2022	2	100%	Harsh
2.3 SCOPE	16-10-2022	18-10-2022	2	100%	Harsh
2.4 REFERENCES	17-10-2022	19-10-2022	2	100%	Anmol
3. OVERVIEW	18-10-2022	20-10-2022	2	100%	Harsh
3.1 OVERALL DESCRIPTION OF PROPOSED SYSTEM	19-10-2022	21-10-2022	2	100%	Anmol
3.2 PRODUCT PERSPECTIVE	20-10-2022	21-10-2022	1	100%	Harsh
4. ASSUMPTIONS AND DEPENDENCIES	21-10-2022	22-10-2022	1	100%	Anmol
5. SYSTEM INTERFACES	22-10-2022	23-10-2022	1	100%	Anmol
6. USER INTERFACES	23-10-2022	24-10-2022	1	100%	Anmol
6.1 HARDWARE INTERFACES	24-10-2022	26-10-2022	2	100%	Harsh
6.2 SOFTWARE INTERFACES	25-10-2022	27-10-2022	2	100%	Anmol
6.3 COMMUNICATION INTERFACES	26-10-2022	28-10-2022	2	100%	Harsh
7. MEMORY CONSTRAINTS	27-10-2022	29-10-2022	2	100%	Anmol
7.1 OPERATIONS:	28-10-2022	30-10-2022	2	100%	Harsh
8. SPECIFIC REQUIREMENTS	29-10-2022	31-10-2022	2	100%	Anmol
8.1 USER CHARACTERISTICS	30-10-2022	01-11-2022	2	100%	Anmol
8.2 PRODUCT FUNCTIONS:	31-10-2022	02-11-2022	2	100%	Anmol
8.3 SYSTEM FEATURES:	01-11-2022	03-11-2022	2	100%	Harsh
8.3.1 MODULE	02-11-2022	04-11-2022	2	100%	Anmol
8.4 PERFORMANCE REQUIREMENTS:	03-11-2022	05-11-2022	2	100%	Harsh
8.5 LOGICAL DATABASE REQUIREMENTS	04-11-2022	06-11-2022	2	100%	Harsh
8.6 DESIGN CONSTRAINTS	05-11-2022	07-11-2022	2	100%	Harsh
8.7 Software System Attributes	06-11-2022	08-11-2022	2	100%	Anmol
8.7.1 Reliability	08-11-2022	09-11-2022	1	100%	Harsh
8.7.2 Availability	09-11-2022	12-11-2022	3	100%	Anmol
8.7.3 Security	12-11-2022	16-11-2022	4	100%	Harsh
8.7.4 Maintainability	16-11-2022	19-11-2022	3	100%	Anmol
8.7.5 Portability	20-11-2022	23-11-2022	3	100%	Harsh
8.8 Other Requirements	24-11-2022	25-11-2022	1	100%	Anmol
CHAPTER - 2 SYSTEM DESIGN	26-11-2022	28-11-2022	2	100%	Anmol
1. PHYSICAL DESIGN:	27-11-2022	01-12-2022	4	100%	Anmol
2. BLOCK DIAGRAM:	03-12-2022	06-12-2022	3	100%	Harsh
3. USE CASE DIAGRAM:	06-12-2022	09-12-2022	3	100%	Anmol
4. DATA FLOW DIAGRAMS:	10-12-2022	12-12-2022	2	100%	Harsh
5. ENTITY RELATIONSHIP DIAGRAM	12-12-2022	14-12-2022	2	100%	Anmol
6. INTERFACE DESIGN	14-12-2022	15-12-2022	1	100%	Harsh
CHAPTER- 3 SYSTEMS DEVELOPMENT & IMPLEMENTATION	15-12-2022	16-12-2022	1	100%	Anmol
1. PROGRAMME DEVELOPMENT	16-12-2022	20-12-2022	4	100%	Harsh
2. TESTING & DEBUGGING	20-12-2022	22-12-2022	2	100%	Harsh
CHAPTER- 4 SCOPE OF IMPROVEMENT, SUMMARY AND CONCLUSIONS	21-12-2022	22-12-2022	1	100%	Harsh
1. Scope of Improvement	22-12-2022	23-12-2022	1	100%	Anmol
2. Summary	23-12-2022	24-12-2022	1	100%	Anmol
3. Conclusion	24-12-2022	25-12-2022	1	100%	Harsh

Table 3: Member Wise Distribution Table

6.2 GANTT CHART:**Fig 2: Gantt Chart**

CHAPTER - 1

INTRODUCTION

1. DESCRIPTION OF ORGANIZATION:

Online food ordering and delivery business have earned a great position in the e-commerce market. E-commerce or business through net means distributing, buying, selling, marketing, and servicing of products or services over electronic systems such as the Internet and other computer business networks. It manages all the information about Food Item, Confirm Order and Food Item. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for managing the Food Item, Category and Customer. It tracks all the details about the Customer, Order and Confirm Order.

➤ Role of Users:

- ❖ Without login portal
 - View Category
 - View Item
- ❖ With login portal
 - Order
 - Add to Cart
 - Profile

➤ Role of Admin:

- CRUD Category
- CRUD Item

1.1 INTRODUCTION:

Existing system is manual system. It requires a lot of manual work to be done, it is a time-consuming system. The existing system is maintained by hand, without using computer system. Earlier the user had to go to the shop and order the food. The user had to stand in a long queue to get food.

Online food ordering is a system for delivering a food restaurant to users using websites. If any user wants to get food from restaurant at home, without travelling and not waste time he/she order food online using Zaykaa. Nowadays more people are ordering food from a restaurant instead of cooking at home. The online ordering system is helpful for both user restaurant owner. Our platform connects customers and restaurant to serving their multiple needs. Customers use our platform to search and discover food item, read and write customer generated reviews and order food online.

1.2 HISTORY OF ORGANIZATION:

None

1.3 OBJECTIVE OF ORGANIZATION:

- To provide 24*7 food ordering service to customers.
- Remove the need for going to restaurant physically.

1.4 FUNCTIONS:

- User can Sign up in the platform to access cart.
- User can search food using search feature.
- User can give review and can also see reviews of others.
- User does not need to enter the address manually, the website will automatically detect the location.
- User will have to add the food into cart first then they can order food online.

1.5 DESCRIPTION OF PROPOSED INFORMATION SYSTEM:

1.5.1 Signup

This process is for both Admin and User. He/she will enter their Login Id and Password to get access to processes like Manage profile, Order Food or Review about Food etc.

1.5.2 Search

The search module lets users search for specific food on website. You can search both for categories and food items. When user is on the home tab of Search, he/she will be able to search for categories.

1.5.3 Order food

1.5.3.1 Add to cart:

In this process User can add food item and it will just keep temporary record in it until user proceed to payment or confirmation of food.

1.5.3.2 Order:

In this process user can order food stored in the cart and user will receive orders details or order id.

1.5.4 Contact Us

The contact us module is for user ease to contact with the customer care service regarding any questions or complaints through e-mail.

2. SOFTWARE REQUIREMENT SPECIFICATIONS:

2.1 INTRODUCTION:

It is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-functional depending upon type of requirement.

The following subsections of Software Requirement Specifications Document should facilitate in providing the entire overview of the Information system “Zaykaa” under development. This document aims at defining the overall software requirements for our website. Efforts have been made to define the requirements of the Information system exhaustively and accurately.

2.2 PURPOSE:

The main purpose of Software Requirement Specifications Document is to describe in a precise manner all the capabilities that will be provided by the Software Application “Zaykaa”. It also states the various constraints which the system will be abide to. This document further leads to clear vision of the software requirements, specifications and capabilities. These are to be exposed to the development, testing team and end users of the software.

2.3 SCOPE:

- Provides the searching for food ordering
- Be easy to understand by the user and operator
- Provide a good user interface
- To provide a platform for easy record maintenance

2.4 REFERENCES:

- 1) Bootstrap - <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
- 2) Tailwind CSS - <https://tailwindui.com/components?ref=sidebar>
- 3) Swiggy - <https://www.swiggy.com/>
- 4) Zomato - <https://www.zomato.com/>
- 5) K.K. Aggarwal & Yogesh Singh, “Software Engineering”, 2nd edition, New Age International, 2005.
- 6) R.S. Pressman, “Software Engineering – A practitioner’s approach”, 5th edition, PHI Learning Private Limited.
- 7) Jeffery C. Jackson, “Web Technologies: A Computer Science Perspective”, Pearson.
- 8) Internet and World Wide Web Deitel HM, Deitel, Goldberg, Third edition.

3. OVERVIEW:

The rest of this SRS document describes the various system requirements, interfaces, features and functionality in detail.

3.1 OVERALL DESCRIPTION OF PROPOSED SYSTEM:

Our site manages all the information about Food Item, Confirm Order and Food Item. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for managing the Food Item, Category and Customer. It tracks all the details about the Customer, Order and Confirm Order.

3.2 PRODUCT PERSPECTIVE:

The application will be windows-based, self-contained and independent software product.

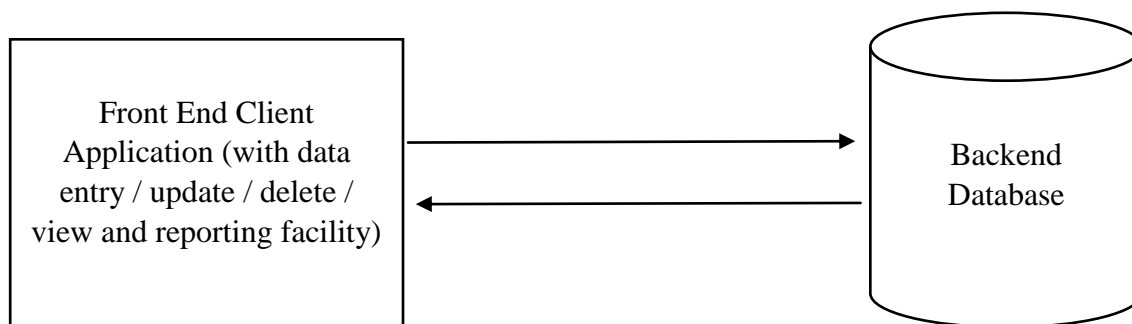


Fig 1.1: Interaction of Product with the environment

4. ASSUMPTIONS AND DEPENDENCIES:

The application is developed on the following assumptions.

- The Users has all the available hardware required to support the intended user load.
- Zaykaa depends on Information, Posts and data stored in database and hence these systems and database servers should be up and available for normal functioning of Zaykaa.

5. SYSTEM INTERFACES:

None

6. USER INTERFACES:

The application will have a user friendly and menu-based interface. Following screens will be provided.

- A Login Screen for entering username, password and role (Admin, User) will be provided. Access to different screens will be based upon the role of the user. It also provides features for "New user registration" and "Forgot password".
- Search page where registered user can search food items based on various attributes. User can search by name, brief description. Search should support intuitive features such as type-ahead, synonym support, categorized results grouping and spell correction.
- On the cart screen, the user will see the food items that he added to the cart in ordered manner and he/she will be able to order it by choosing one of the payment methods there on the payment page.
- User can give his overall views on review page about website, delivery, how much he likes the food, restaurant etc.

6.1 HARDWARE INTERFACES:

Categories	Requirement
RAM	2 GB (and higher)
HDD	5 GB (and higher)
Processor	i3 (and higher)

Hardware requirement

6.2 SOFTWARE INTERFACES:

Software	Requirement
Operating System	Windows (7 and higher), Mac (10.12 and higher), Linux (ubuntu 14.10)
Front-End	Html 5, CSS 3, React 16.8.6, Express 3.18.1, Bootstrap 5, Tailwind CSS 3.1.7
Back-End	MongoDB 5.0, NodeJS 16.14.2

Software requirement

6.3 COMMUNICATION INTERFACES:

None

7. MEMORY CONSTRAINTS:

7.1 OPERATIONS:

This product will not cover any automated housekeeping aspects of database. The DBA at client site will be manually deleting old/ non required data. Database backup and recovery will also have to be handled by DBA.

8. SPECIFIC REQUIREMENTS:

This section contains the software requirements to a level of detail sufficient to enable designers to design the system, and testers to test the system.

8.1 USER CHARACTERISTICS:

- 1) **Educational Level:** At least graduate and should be comfortable with English language.
- 2) **Technical Expertise:** Should be a high or middle level employee of the organization comfortable with using general purpose applications on a computer.

8.2 PRODUCT FUNCTIONS:

The system will allow access only to authorized users with specific roles (Administrator, Operator). Depending upon the user's role, he/she will be able to access only specific modules of the system.

A summary of the major functions that the software will perform:

- i. A Login facility for enabling only authorized access to the system.
- ii. Users will be able to add/update/delete the stored food item in the cart.
- iii. Users will be able to give review.
- iv. Admin will be able to add/update/delete the categories and food items.
- v. Admin manages the order details.

8.3 SYSTEM FEATURES:

8.3.1 MODULE:

1. **Items:** Used for managing the Food details.
2. **Order:** Used for managing the details of Order
3. **Category:** Used for managing the information and details of the Item Category.

4. **Cart:** Used for managing the Shopping Cart details
5. **Profile:** Used for managing the Customer information
6. **Login:** Used for managing the login details
7. **Search:** Used for searching categories and food items.
8. **Contact Us:** Used for contacting customer care service.

8.4 PERFORMANCE REQUIREMENTS:

None

8.5 LOGICAL DATABASE REQUIREMENTS

The proposed information system contains the following data tables in its database collection.

- Users
- Admins
- Categories
- Items
- Reviews

8.6 DESIGN CONSTRAINTS

Standard Compliance

None

8.7 Software System Attributes

8.7.1 Reliability

This application is a reliable product that produces fast and verified output of all its processes.

8.7.2 Availability

This application will be available to use for users and help them to carry out their operations conveniently.

8.7.3 Security

The application will be password protected. User will have to enter correct username, password and role in order to access the application.

8.7.4 Maintainability

The application will be designed in a maintainable manner. It will be easy to incorporate new requirements in the individual modules.

8.7.5 Portability

The application will be easily portable on any windows based system that has oracle installed.

8.8 Other Requirements

None

CHAPTER - 2

SYSTEM DESIGN

1. PHYSICAL DESIGN:

This website manages all the information about Food Item, Confirm Order and Food Item. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for managing the Food Item, Category and Customer. It tracks all the details about the Customer, Order and Confirm Order.

➤ Role of Users:

- ❖ Without login portal
 - View Category
 - View Item
- ❖ With login portal
 - Order
 - Add to Cart
 - Profile

➤ Role of Admin:

- CRUD Category
- CRUD Item

2. BLOCK DIAGRAM:

A block diagram is a diagram of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. They are heavily used in engineering in hardware design, electronic design, software design, and process flow diagrams.

Block diagrams are typically used for higher level, less detailed descriptions that are intended to clarify overall concepts without concern for the details of implementation. Contrast this with the schematic diagrams and layout diagrams used in electrical engineering, which show the implementation details of electrical components and physical construction.

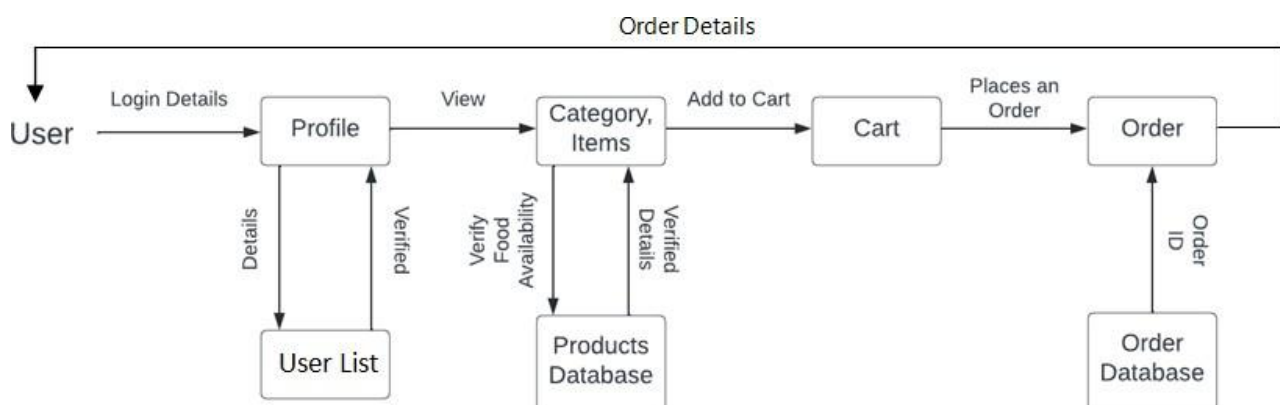


Fig 2.1: Block diagram

3. USE CASE DIAGRAM:

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

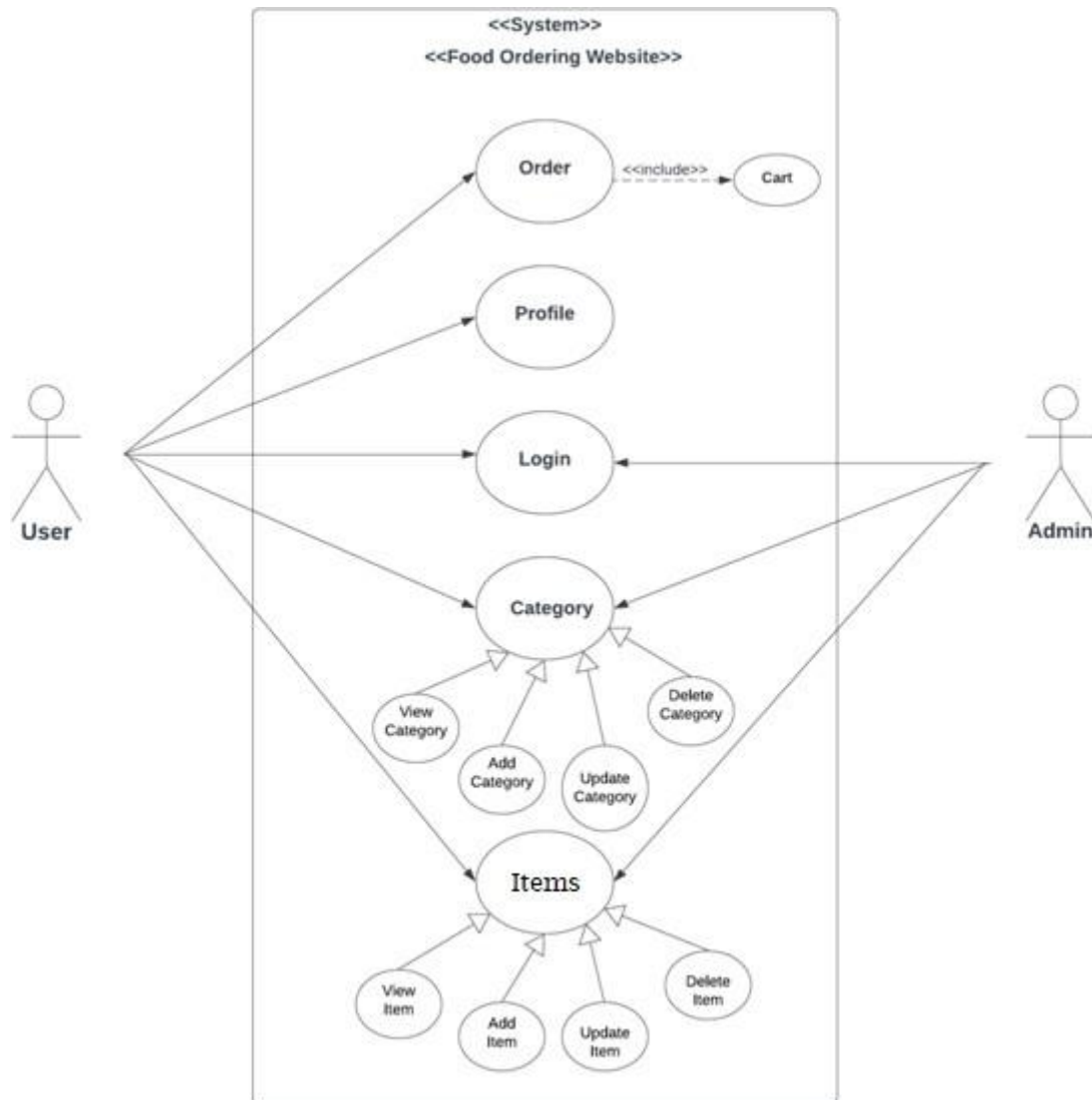


Fig 2.2: Use case diagram

4. DATA FLOW DIAGRAMS:

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. DFD can be drawn to represent the system of different levels of abstraction. Higher level DFDs are partitioned into low levels having more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond.

0 level DFD: It is also known as context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.

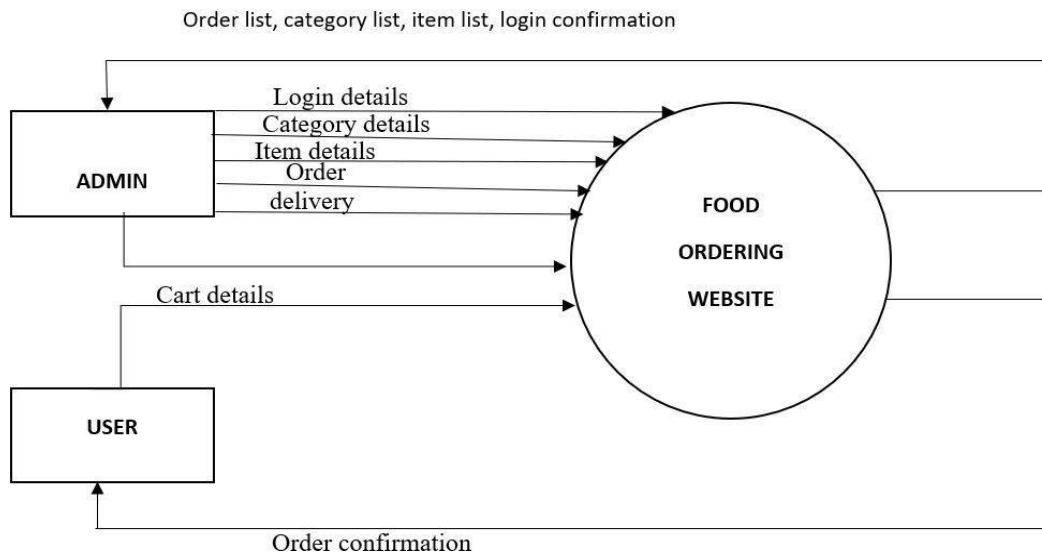


Fig 2.3: 0 Level Data Flow Diagram (DFD)

Login Details include: id, password.

Category Details include: cat_name, cat_subname, cat_desc.

Item Details include: item_name, item_amt, item_status, item_qty.

Order Details include: cat_name, item_name, payment mode, item_status.

1 level DFD: In 1-level DFD, context diagram is decomposed into multiple bubbles/processes. In this level we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into sub processes.

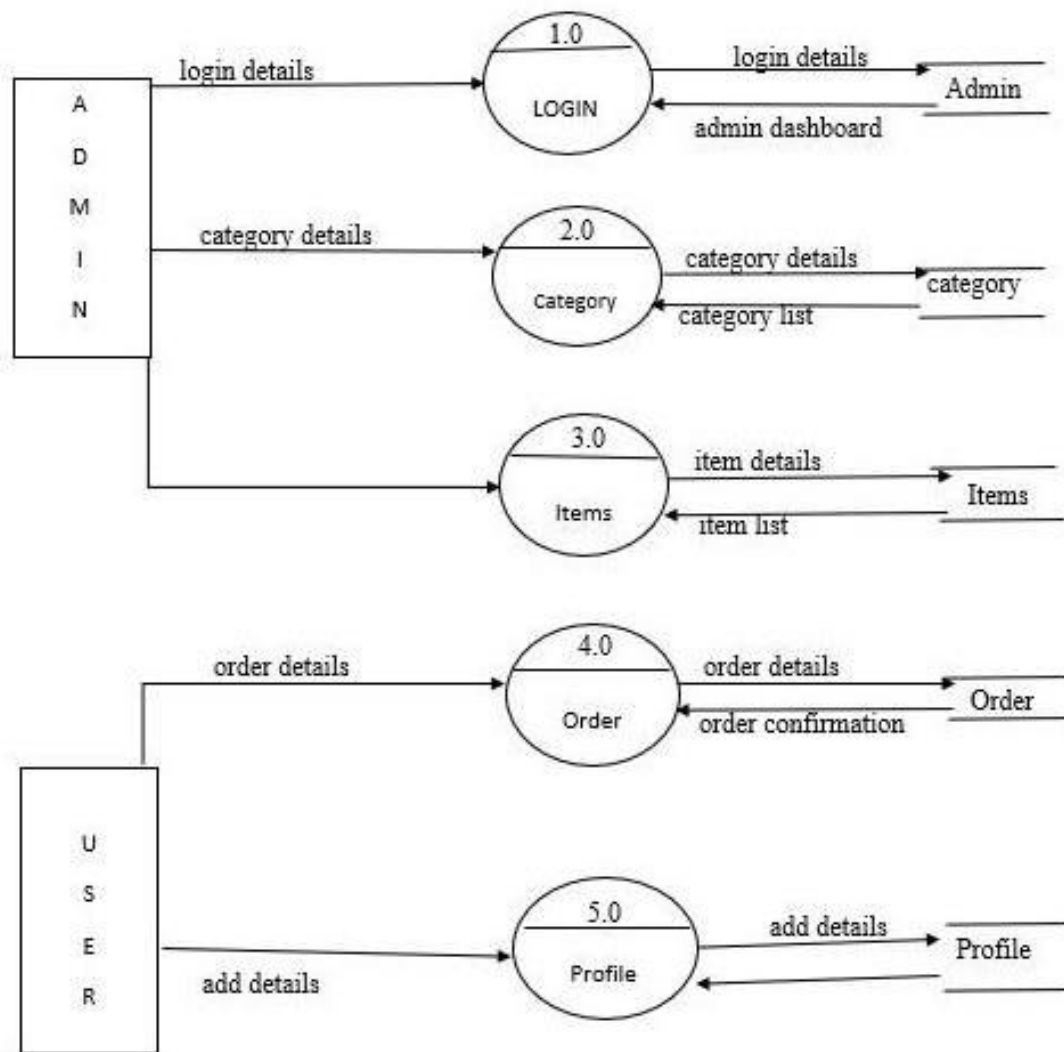


Fig 2.4: 1 Level Data Flow Diagram (DFD)

2 level DFD: 2-level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.

MANAGE CATEGORY

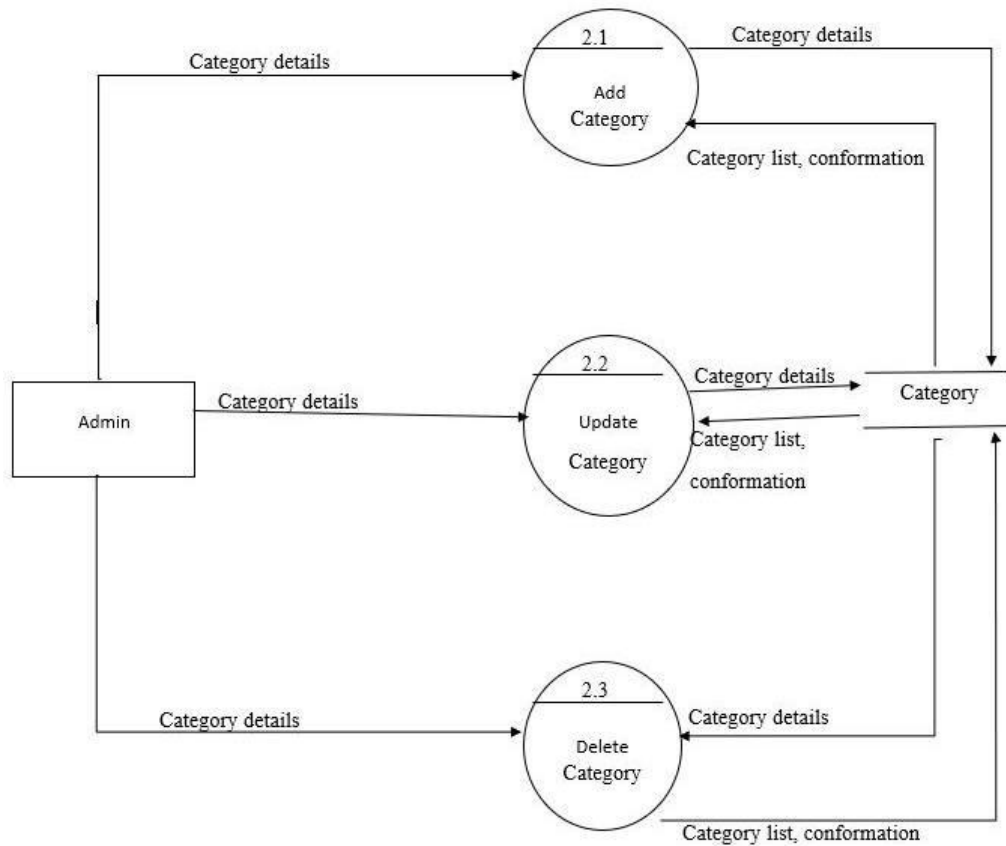
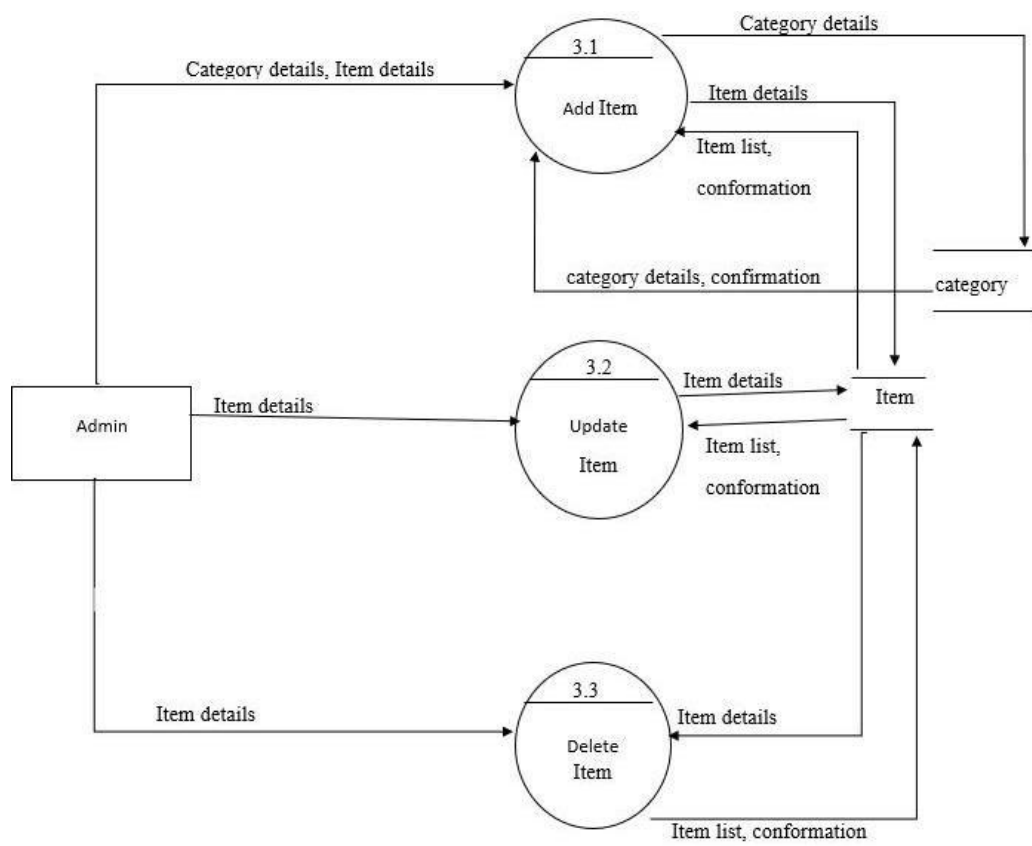
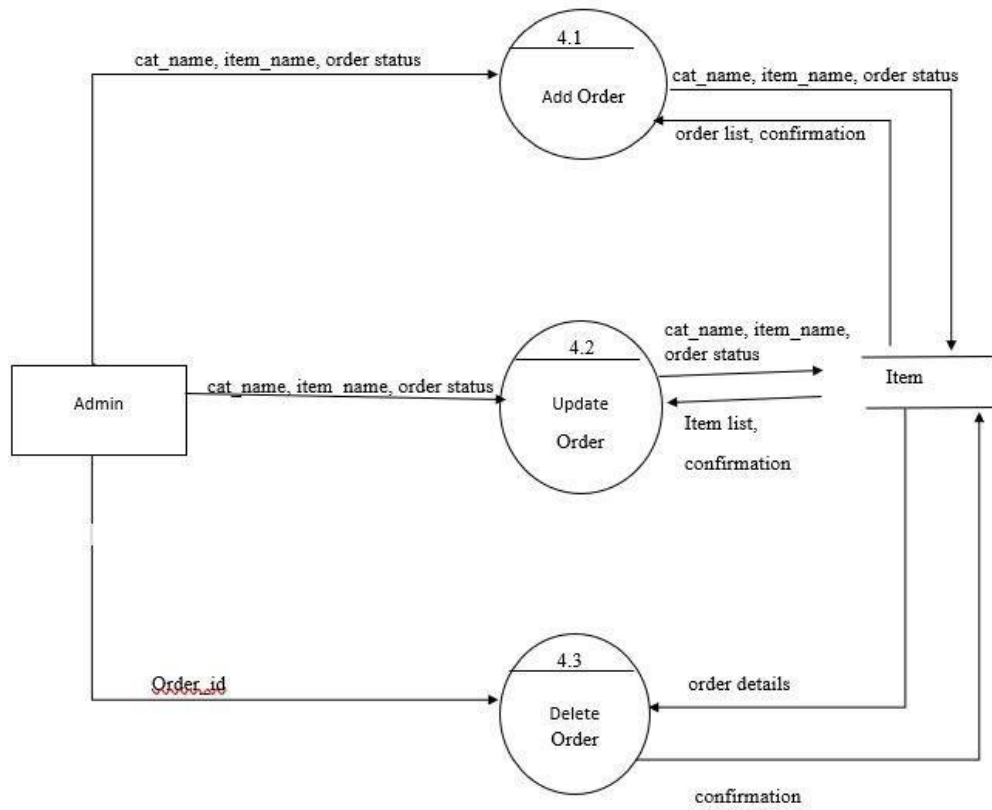


Fig 2.5: 2 Level Data Flow Diagram (DFD)

MANAGE ITEM**Fig 2.6: Level 2 Data Flow Diagram (DFD)**

MANAGE ORDER**Fig 2.7: Level 2 Data Flow Diagram (DFD)**

5. ENTITY RELATIONSHIP DIAGRAM:

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

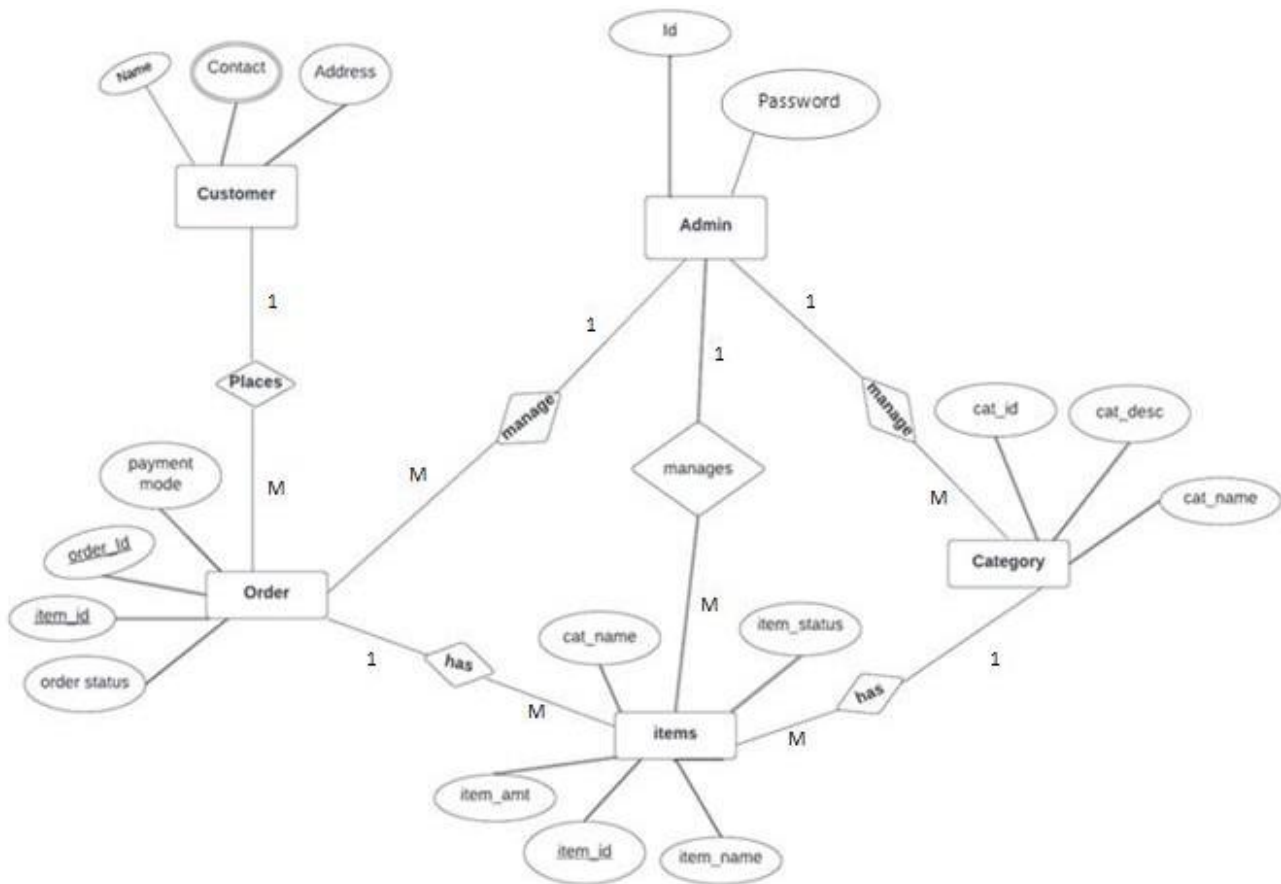


Fig 2.8: Entity Relationship Diagram:

5.1 DATABASE DESIGN:

The information system of “Placement Tracker” performs its function with the help of the data store in certain repositories called Databases of the system. Detailed descriptions of the various databases included in the information systems are tabulated as follows:

5.1.1 LOGIN DATABASE:

Table 2.1: User Login Database

No.	Field Name	Input	Output
1	Name	Character(x30) in length	This field is a mandatory field for a successful login.
2	Phone_no	Integer(x10) in length	This field is a mandatory field for a successful login.
3	Email	Character(x20) in length	This field is a mandatory field for a successful login.
4	U_password	Character(x20) in length	This field is a mandatory field for a successful login.

Table 2.2: Admin Login Database

No.	Field Name	Input	Output
1	A_Email	Character(x20) in length	This field is a mandatory field for a successful login.
2	A_Password	Character(x20) in length	This field is a mandatory field for a successful login.

SCHEMA OF TABLE PROPERLY LABELED:

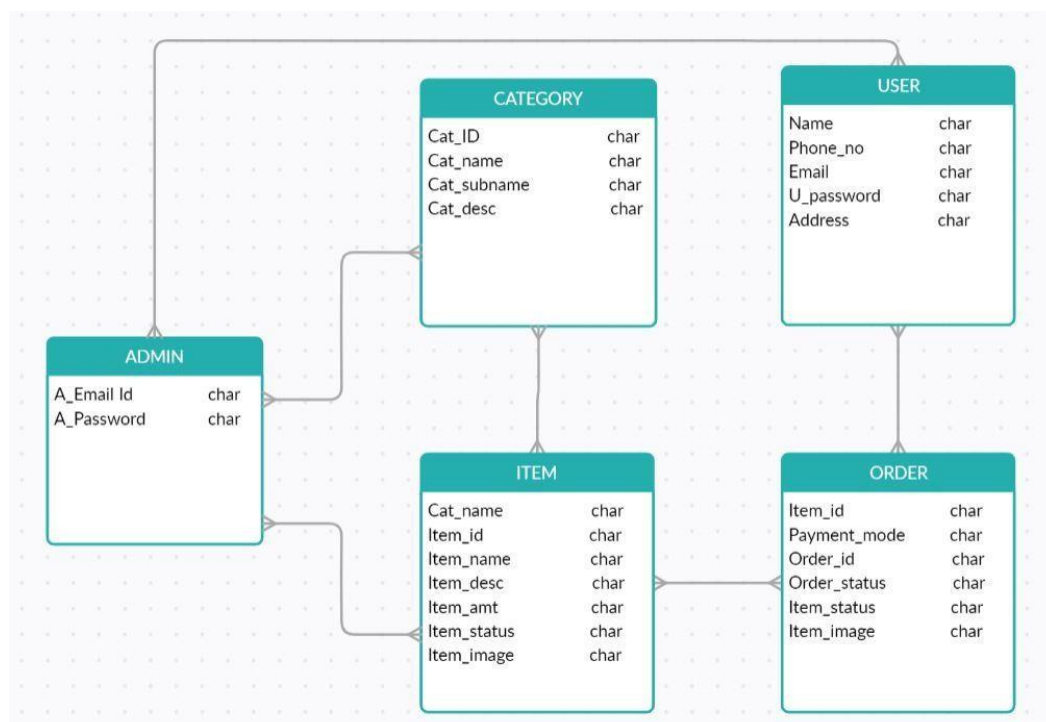


Fig 2.9: Schema of Table

5.2 DATA DICTIONARY:

A data dictionary is metadata repository or a centralized repository of information about data such as meaning, relationships to other data, origin, usage and format. The term may have one of several closely related meaning pertaining to databases and database management systems (DBMS):

A document describing a database or collection of databases. An integral component of DBMS that is required to determine its structure.

A piece of middleware that extends or supplants the native data dictionary of a DBMS.

Table 2.3: Admin Table

No.	Column Name	Field Type	Size	Constraints	Description
1	A_Email Id	Character	20	Primary key	Admin email id for login
2	A_Password	Character	20	Not null	Login password for admin

Table 2.4: User Table

No.	Column Name	Field Type	Size	Constraints	Description
1	Name	Character	30	Not null	Name of the user
2	Phone_no	Integer	10	Not null	Phone number of the user
3	Email	Character	20	Primary Key	Email of the user
4	U_password	Character	20	Not null	Login password of user
5	Address	Character	100	Not null	Address of user
6	Cart	Array	20	Not null	Items added to be in the user cart

Table 2.5: Category Table

No.	Column Name	Field Type	Size	Constraints	Description
1	Cat_ID	Character	30	Primary key	Unique id for category
2	Cat_name	Character	20	Not null	Name of the category
3	Cat_subname	Character	40	Not null	Sub Name of the category
4	Cat_desc	Character	100	Not null	Description of category

Table 2.6: Items Table

No.	Column Name	Field Type	Size	Constraints	Description
1	Cat_name	Character	20	Not null	Name of the category
2	Item_id	Character	20	Primary Key	Unique id of the item
3	Item_name	Character	60	Not null	Name of the item
4	Item_desc	Character	100	Not null	Description of the item
5	Item_amt	Character	04	Not null	Amount of the item
6	Item_status	Character	20	Not null	Status of the item
7	Item_image	Character	150	Not null	Path of the file to be uploaded

Table 2.7: Orders Table

No.	Column Name	Field Type	Size	Constraints	Output
1	Item_id	Character	20	Not null	ID of the item
2	Payment_mode	Character	20	Not null	Payment mode of the order
3	Order_id	Character	20	Primary Key	Order id of the placed order
4	Order_status	Character	20	Not null	Status of the order

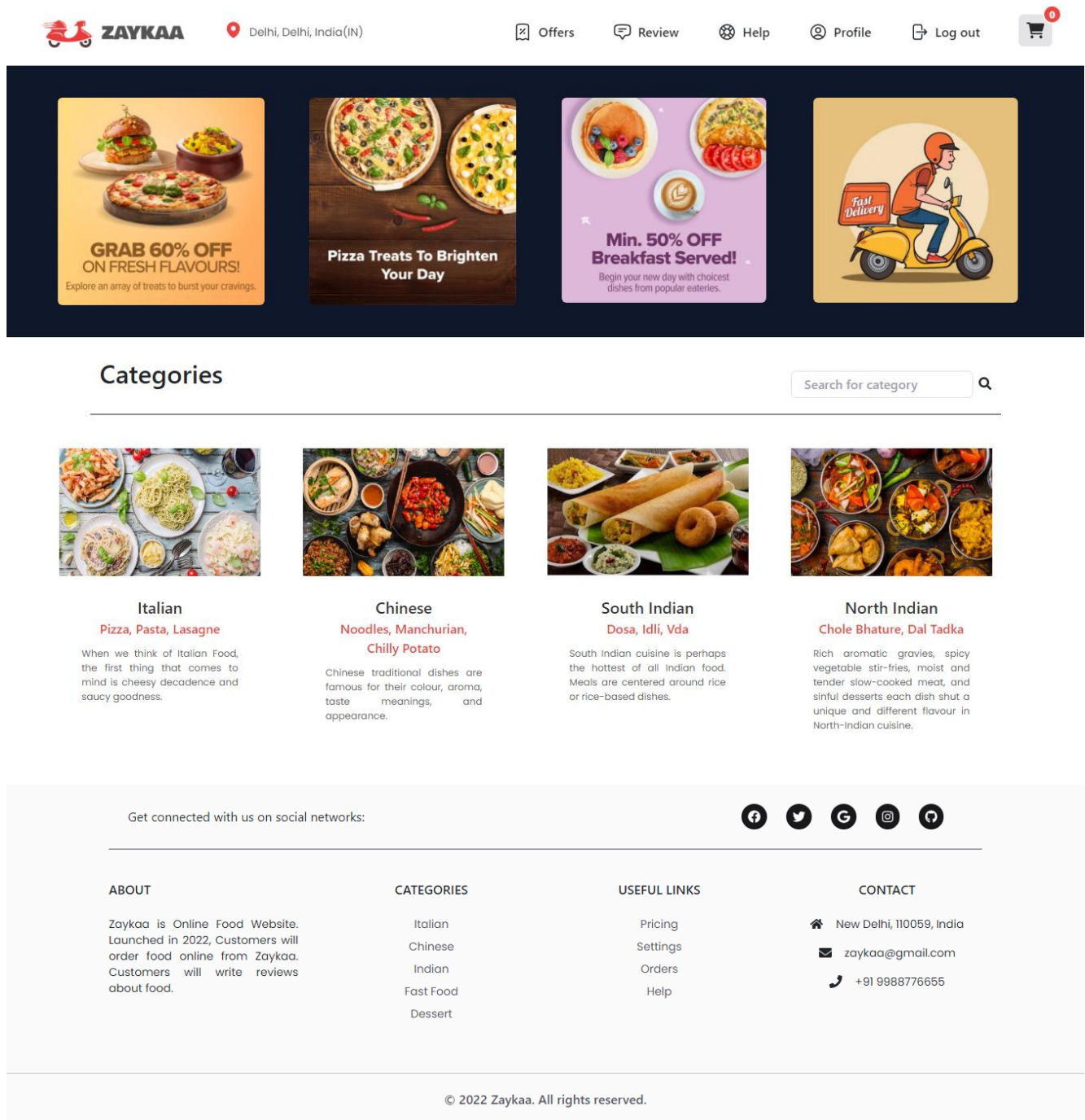
6. INTERFACE DESIGN:

The interface design consists of the input and output source layouts. i.e. the input forms and screens and the report layouts that form as a source of outcome and income in the design and implementation of the information system under study.

6.1 INPUT DESIGN:

The input specifications of the existing information system include the illustration of the detailed characteristics of contents included in each Input Screen and documents. The description for each graphical user interface has been mentioned.

EXISTING SYSTEM DESIGN (Graphical User Interface)



CHAPTER - 3

SYSTEMS DEVELOPMENT & IMPLEMENTATION

1. PROGRAMME DEVELOPMENT

- **DATABASE CONNECTION**

app.js

```
const createError = require("http-errors");
const express = require("express");
const path = require("path");
const dotenv = require("dotenv").config();
const cookieParser = require("cookie-parser");
const logger = require("morgan");
const cors = require("cors");
const mongoose = require("mongoose");
const usersRouter = require("./routes/users");
const adminRouter = require("./routes/admin/admin");
const CategoryRouter = require("./routes/admin/category");
const ItemRouter = require("./routes/admin/items");
const cartRouter = require("./routes/carts");

const app = express();
app.use(cors());

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "jade");

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use("/users", usersRouter);
app.use("/admin", adminRouter);
app.use("/cart", cartRouter);
app.use("/category", CategoryRouter);
app.use("/item", ItemRouter);
app.use("/uploads", express.static("./uploads"));

// catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});
```

```
// MongoDB Connectivity with Mongoose starts here

mongoose.connect(
  // "mongodb://127.0.0.1:27017/Zaykaa",
  process.env.MONGO_CONNECTION_URL,
  { useNewUrlParser: true },
  (err) => {
    if (err) {
      console.log(err);
    } else {
      console.log("Db Connected");
    }
  }
);

// MongoDB Connectivity with Mongoose ends here

// error handler
app.use(function (err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render("error");
});

module.exports = app;
```

- **MODELS (Tables Schema)**

admin.js

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

var admin = new Schema(
  {
    role: {
      type: String,
      default: 'admin'
    },
    email : {
      type: String,
      required: true,
      trim: true,
      unique: true
    }
  }
);
```



```
    },
    password : {
      type: String,
      required: true,
      minlength: 8
    }
  }
};

module.exports = mongoose.model('Admin', admin);
```

user.js

```
var mongoose = require("mongoose");
const Schema = mongoose.Schema;
const jwt = require("jsonwebtoken");

var user = new Schema(
  {
    role: {
      type: String,
      default: "customer",
    },
    name: {
      type: String,
      required: true,
      maxlength: 100,
    },
    email: {
      type: String,
      required: true,
      trim: true,
      unique: true,
    },
    number: {
      type: String,
      required: true,
      minlength: 10,
      maxlength: 10,
    },
    address: {
      type: String,
      // required: true,
      maxlength: 100,
    },
    password: {
      type: String,
      required: true,
```

```
    minlength: 8,
  },
  tokens: [
    {
      token: {
        type: String,
        required: false,
      },
    },
  ],
  carts: Array,
  emailToken: {
    type: String,
  },
  isVerified: {
    type: Boolean,
  },
  verifytoken: {
    type: String,
  },
},
{ timestamps: true }
);
```

```
user.methods.generatAuthtoken = async function () {
  try {
    let token = jwt.sign({ _id: this._id }, process.env.SECRET_KEY, {
      expiresIn: "1d",
    });
    this.tokens = this.tokens.concat({ token: token });
    await this.save();
    return token;
  } catch (error) {
    console.log(error);
  }
};
```

```
// add to cart data
user.methods.addcartdata = async function (cart) {
  try {
    this.carts = this.carts.concat(cart);
    await this.save();
    return this.carts;
  } catch (error) {
    console.log(error);
  }
};
```

```
module.exports = mongoose.model("User", user);
```

item.js

```
var mongoose = require("mongoose");  
var Schema = mongoose.Schema;
```

```
var ItemSchema = new Schema({  
  catname: {  
    type: String,  
    required: true,  
    maxlength: 60,  
  },  
  imgpath: {  
    type: String,  
    required: false,  
  },  
  name: {  
    type: String,  
    required: true,  
    maxlength: 60,  
  },  
  size: {  
    type: Object,  
    // required: true,  
  },  
  price: {  
    type: Object,  
    // required: true,  
  },  
});
```

```
module.exports = mongoose.model("Items", ItemSchema);
```

category.js

```
var mongoose = require("mongoose");  
var Schema = mongoose.Schema;
```

```
var CategorySchema = new Schema({  
  imgpath: {  
    type: String,  
    required: false,  
  },  
  name: {  
    type: String,  
    required: true,  
    maxlength: 60,  
  },  
});
```

```
    },
    subname: {
      type: String,
      required: true,
      maxlength: 60,
    },
    desc: {
      type: String,
      required: true,
      maxlength: 300,
    },
  },
});
```

```
module.exports = mongoose.model("Category", CategorySchema);
```

- **ROUTES**

- users.js**

```
const express = require("express");
const router = express.Router();
const User = require("../models/user");
const Item = require("../models/item");
const crypto = require("crypto");
const bcrypt = require("bcryptjs");
const nodemailer = require("nodemailer");
const { verifyEmail } = require("../utils/JWT");
const authenticate = require("../middleware/authenticate.js");
const jwt = require("jsonwebtoken");

let {
  encryptPassword,
  comparePasswords,
  generateJwt,
} = require("../utils/auth");

router.get("/", function (req, res, next) {
  res.send("respond with a resource");
});

router.post("/adduser", async (req, res) => {
  try {
    let user = await new User(req.body).save();
    res.json({ message: "User Added Successfully", data: user, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

```
router.get("/getuser", async (req, res) => {
  try {
    const getuser = await User.find().exec();
    res.json({ message: "User Details", data: getuser, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

```
router.post("/updateuser", async (req, res) => {
  try {
    let user = await User.findByIdAndUpdate(req.body.id, {
      name: req.body.name,
    }).exec();
    res.json({
      message: "User Successfully Updated",
      data: user,
      success: true,
    });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

```
router.post("/deleteuser", async (req, res) => {
  try {
    await User.findByIdAndRemove(req.body.id).exec();
    res.json({ message: "Successfully Deleted", success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

//Email AUTH

```
const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: "noreply.zaykaa@gmail.com",
    pass: process.env.PASS,
  },
  tls: {
    rejectUnauthorized: false,
  },
});
```

// User Register API starts here

```
router.post("/register", async (req, res) => {
  try {
    const { name, email, number, password } = req.body;
    const user = new User({
      name,
      email,
      number,
      password,
      emailToken: crypto.randomBytes(64).toString("hex"),
      isVerified: false,
    });

    const userEmailCheck = await User.findOne({
      email: new RegExp(`^${req.body.email}$`, "i"),
    }).exec();

    const minnumber = number.length;
    if (minnumber !== 10)
      return res.status(400).send({ message: "Number must be 10 digits" });

    const userNumberCheck = await User.findOne({
      number: new RegExp(`^${req.body.number}$`, "i"),
    }).exec();

    const minpassword = password.length;
    if (minpassword < 8)
      return res.status(400).send({ message: "Password must be 8 digits" });

    if (!!name, !!email, !!number, !!password)
      return res.status(400).send({ message: "All Fields are Required" });

    if (userEmailCheck)
      return res.status(401).send({ message: "Email Already Registered" });

    if (userNumberCheck)
      return res
        .status(402)
        .send({ message: "Mobile Number Already Registered" });

    req.body.password = await encryptPassword(req.body.password);

    let newUser = await new User(req.body).save();

    //send verification mail to user
    var mailOptions = {
      from: ' "Verify your email" <anmol.garg594@gmail.com> ',
      to: user.email,
```

```

    subject: "Zaykaa -verify your email",
    html: `

## 


```

```
});

// User Login API Starts Here

router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({
      email: new RegExp(`^${req.body.email}$`, "i"),
    }).exec();

    if (!user) return res.status(401).json({ message: "Email not registered" });

    const minpassword = password.length;
    if (minpassword < 8)
      return res.status(400).send({ message: "Password must be 8 digits" });

    if (!!email, !password)
      return res.status(400).send({ message: "All Fields are Required" });

    if (user) {
      const checkPassword = await comparePasswords(
        req.body.password,
        user.password
      );

      if (!checkPassword)
        return res.status(402).send({ message: "Invalid Password" });

      const token = await user.generatAuthToken();

      res.cookie("Zaykaa", token, {
        expires: new Date(Date.now() + 2589000),
        httpOnly: true,
      });
      res.status(200).send({ data: token, message: "Logged in successfully" });
      console.log(token);
    }
  } catch (err) {
    console.error(err);
    if (err.message) {
      res.status(500).send({ message: "Internal Server Error" });
    } else res.json({ message: "Error", data: err, success: false });
  }
});

// User Login Api Ends

// send email Link For reset Password
```



```
router.post("/sendpasswordlink", async (req, res) => {
  console.log(req.body);

  const { email } = req.body;

  if (!email) {
    res.status(401).json({ status: 401, message: "Enter Your Email" });
  }

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

    if (!userfind)
      return res.status(401).json({ message: "Email not registered" });

    // token generate for reset password
    const token = jwt.sign({ _id: userfind._id }, process.env.SECRET_KEY, {
      expiresIn: "120s",
    });

    const setusertoken = await User.findByIdAndUpdate(
      { _id: userfind._id },
      { verifytoken: token },
      { new: true }
    );

    if (setusertoken) {
      const mailOptions = {
        from: "noreply.zaykaa@gmail.com",
        to: email,
        subject: "Sending Email For password Reset",
        text: `This Link Valid For 2 MINUTES http://localhost:3000/forgot-password/${userfind._id}/${setusertoken.verifytoken}`,
      };

      transporter.sendMail(mailOptions, (error, info) => {
        if (error) {
          console.log("error", error);
          res.status(401).json({ status: 401, message: "email not send" });
        } else {
          console.log("Email sent", info.response);
          res
            .status(201)
            .json({ status: 201, message: "Email sent Successfully" });
        }
      });
    }
  } catch (error) {
```

```
    res.status(401).json({ status: 401, message: "invalid user" });
  }
});

// verify user for forgot password time
router.get("/forgot-password/:_id/:token", async (req, res) => {
  const { _id, token } = req.params;
  try {
    const validuser = await User.findOne({ _id: _id, verifytoken: token });
    const verifyToken = jwt.verify(token, process.env.SECRET_KEY);
    console.log(verifyToken);
    if (validuser && verifyToken._id) {
      res.status(201).json({ status: 201, validuser });
    } else {
      res.status(401).json({ status: 401, message: "user not exist" });
    }
  } catch (error) {
    res.status(401).json({ status: 401, error });
  }
});

// change password
router.post("/:_id/:token", async (req, res) => {
  const { _id, token } = req.params;
  const { password } = req.body;

  try {
    const validuser = await User.findOne({ _id: _id, verifytoken: token });
    const verifyToken = jwt.verify(token, process.env.SECRET_KEY);

    if (validuser && verifyToken._id) {
      const minpassword = password.length;
      if (minpassword < 8)
        return res.status(400).send({ message: "Password must be 8 digits" });

      if (!password) return res.status(400).send({ message: "Required" });

      const newpassword = await encryptPassword(password);
      console.log(newpassword);
      const setnewuserpass = await User.findByIdAndUpdate(
        { _id: _id },
        { password: newpassword }
      );
      setnewuserpass.save();
      res.status(201).json({ status: 201, setnewuserpass });
    } else {
      res.status(401).json({ status: 401, message: "user not exist" });
    }
  }
});
```

```
    } catch (error) {
      res.status(401).json({ status: 401, error });
    }
  });

// get user is login or not
router.get("/validuser", authenticate, async (req, res) => {
  try {
    const validuserone = await User.findOne({ _id: req.userID });
    res.status(201).json(validuserone);
  } catch (error) {
    console.log(error);
  }
});

//profile
router.get("/profile", authenticate, async (req, res) => {
  console.log("Profile");
  res.send(req.rootUser);
});

//profile update
router.post("/updateprofile", authenticate, async (req, res) => {
  try {
    const { name, email, number, address } = req.body;
    if (!name || !email || !number || !address) {
      console.log("All Fields are required");
      return res.json({ error: "All Fields are required" });
    }
    let item = await User.findByIdAndUpdate(req.userID, {
      name,
      email,
      number,
      address,
    }).exec();
    res.json({
      message: "Profile Successfully Updated",
      data: item,
      success: true,
    });
  } catch (error) {
    console.log(error);
  }
});

//delete account
router.delete("/deleteaccount", authenticate, async (req, res) => {
  const result = await User.deleteOne({ _id: req.userID });
```

```
    res.send(result);
  });

// for userlogout
router.get("/logout", authenticate, async (req, res) => {
  try {
    req.rootUser.tokens = req.rootUser.tokens.filter((curelem) => {
      return curelem.token !== req.token;
    });

    res.clearCookie("Zaykaa", { path: "/" });
    req.rootUser.save();
    res.status(201).json(req.rootUser.tokens);
    console.log("user logout");
  } catch (error) {
    console.log(error);
  }
});

module.exports = router;
```

admin.js

```
var express = require("express");
var router = express.Router();
var Admin = require("../models/admin");

let {
  encryptPassword,
  comparePasswords,
  generateJwt,
} = require("../utils/auth");

router.get("/", function (req, res, next) {
  res.send("respond with a resource");
});

router.get("/getadmin", async (req, res) => {
  try {
    const getadmin = await User.find().exec();
    res.json({ message: "Admin Details", data: getadmin, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.post("/deleteadmin", async (req, res) => {
  try {
```

```
    await User.findByIdAndRemove(req.body.id).exec();
    res.json({ message: "Successfully Deleted", success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

// Admin Register API starts here

```
router.post("/adminregister", async (req, res) => {
  try {
    const adminEmailCheck = await Admin.findOne({
      email: new RegExp(`^${req.body.email}$`, "i"),
    }).exec();

    if (adminEmailCheck) throw new Error("Email already registered");

    req.body.password = await encryptPassword(req.body.password);

    let admin = await new Admin(req.body).save();
    res.status(200).json({
      message: "Admin Register Successfully",
      data: admin,
      success: true,
    });
  } catch (err) {
    console.error(err);
    if (err.message)
      res.json({ message: err.message, data: err, success: false });
    else res.json({ message: "Error", data: err, success: false });
  }
});
```

// Admin Register API Close

// Admin Login API Starts Here

```
router.post("/adminlogin", async (req, res) => {
  try {
    const admin = await Admin.findOne({
      email: new RegExp(`^${req.body.email}$`, "i"),
    }).exec();

    if (!admin) return res.status(401).json({ message: "Invalid Email" });

    const checkPassword = await comparePasswords(
      req.body.password,
      admin.password
    );
```

```
);

if (!checkPassword)
  return res.status(401).json({ message: "Invalid Password" });

const token = await generateJwt(Admin._id);
res.json({ message: "Logged In", data: token, success: true });
} catch (err) {
  console.error(err);
  if (err.message) res.status(500).send({ message: "Internal Server Error" });
  else res.json({ message: "Error", data: err, success: false });
}
});

// Admin Login Api Ends
```

```
module.exports = router;
```

category.js

```
const express = require("express");
const router = express.Router();
const multer = require("multer");
const Category = require("../models/category");

//img storage path
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "/uploads");
  },
  filename: (req, file, cb) => {
    cb(null, `category-${Date.now()}.${file.originalname}`);
  },
});

//img filter
const isImage = (req, file, cb) => {
  if (file.mimetype.startsWith("image")) {
    cb(null, true);
  } else {
    cb(new Error("Only images is allowed"));
  }
};

const upload = multer({
  storage: storage,
  fileFilter: isImage,
});
```

```
router.post("/addcategory", upload.single("image"), async (req, res) => {
  const { filename } = req.file;
  const { name } = req.body;
  const { subname } = req.body;
  const { desc } = req.body;

  if (!filename || !name || !subname || !desc) {
    res.status(401).json({ status: 401, message: "All fields are required" });
  }

  try {
    let category = await new Category({
      imgpath: filename,
      name,
      subname,
      desc,
    }).save();
    res.json({
      message: "Category Added Successfully",
      data: category,
      success: true,
    });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.get("/getcategory", async (req, res) => {
  try {
    const getcategory = await Category.find();
    res.json({ message: "Category Details", data: getcategory, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.post("/updatecategory/:_id", async (req, res) => {
  try {
    let category = await Category.findByIdAndUpdate(req.body._id, {
      name: req.body.name,
      subname: req.body.subname,
      desc: req.body.desc,
    }).exec();
    res.json({
      message: "Category Successfully Updated",
      data: category,
      success: true,
    });
  }
});
```

```

    });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.delete("/deletecategory/:id", async (req, res) => {
  const result = await Category.deleteOne({
    _id: new mongoose.ObjectId(req.params.id),
  });
  res.send(result);
});

router.get("/search/:key", async (req, resp) => {
  let result = await Category.find({
    $or: [
      { name: { $regex: req.params.key } },
      { subname: { $regex: req.params.key } },
    ],
  });
  resp.send(result);
});

//get individual category
router.get("/category/:name", async (req, res) => {
  try {
    const { name } = req.params;
    // console.log(name);

    const individualcategory = await Category.findOne({ name: name });
    // console.log(individualcategory);
    res.status(201).json(individualcategory);
  } catch (error) {
    res.status(400).json(error);
  }
});

module.exports = router;

```

items.js

```

var express = require("express");
var router = express.Router();
const multer = require("multer");
var Item = require("../models/item");

//img storage path
const storage = multer.diskStorage({

```



```
destination: (req, file, cb) => {
  cb(null, "/uploads");
},
filename: (req, file, cb) => {
  cb(null, `item-${Date.now()}.${file.originalname}`);
},
});

//img filter
const isImage = (req, file, cb) => {
  if (file.mimetype.startsWith("image")) {
    cb(null, true);
  } else {
    cb(new Error("Only images is allowed"));
  }
};

const upload = multer({
  storage: storage,
  fileFilter: isImage,
});

router.post("/additem", upload.single("image"), async (req, res) => {
  const { filename } = req.file;
  const { catname } = req.body;
  const { name } = req.body;
  const { size } = req.body;
  const { price } = req.body;

  if (!filename || !catname || !name || !size || !price) {
    res.status(401).json({ status: 401, message: "All fields are required" });
  }

  try {
    let item = await new Item({
      imgpath: filename,
      catname,
      name,
      size,
      price,
    }).save();
    res.json({
      message: "Item Added Successfully",
      data: item,
      success: true,
    });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});
```

```
    }
  });

router.get("/getitem/:name", async (req, res) => {
  try {
    const { name } = req.params;
    const getitem = await Item.find({ catname: name }).exec();
    res.json({ message: "Item Details", data: getitem, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.get("/getitem", async (req, res) => {
  try {
    const getitem = await Item.find().exec();
    res.json({ message: "Item Details", data: getitem, success: true });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.post("/updateitem/:_id", async (req, res) => {
  try {
    // const { _id } = req.body;
    let item = await Item.findByIdAndUpdate(req.body._id, {
      name: req.body.name,
      catname: req.body.catname,
      price: req.body.price,
    }).exec();
    res.json({
      message: "Item Successfully Updated",
      data: item,
      success: true,
    });
  } catch (err) {
    res.json({ message: err.message, success: false });
  }
});

router.delete("/deleteitem/:id", async (req, res) => {
  const result = await Item.deleteOne({ _id: req.params.id });
  // console.log(req.params.id);
  res.send(result);
});

router.get("/search/:key", async (req, resp) => {
  let result = await Item.find({
```

```

    $or: [
      { name: { $regex: req.params.key } },
      { catname: { $regex: req.params.key } },
    ],
  });
  resp.send(result);
});

```

```
module.exports = router;
```

carts.js

```

const express = require("express");
const router = express.Router();
const User = require("../models/user");
const Item = require("../models/item");
const authenticate = require("../middleware/authenticate.js");

// adding the data into cart
router.post("/addtocart/:_id", authenticate, async (req, res) => {
  try {
    const { _id } = req.params;
    console.log(_id);
    const cart = await Item.findOne({ _id: _id });
    console.log(cart);

    const Usercontact = await User.findOne({ _id: req.userID });
    console.log(Usercontact);

    if (Usercontact) {
      const cartData = await Usercontact.addcartdata(cart);
      await Usercontact.save();
      res.status(201).json(Usercontact);
    } else {
      return res.status(402).send({ message: "Please LogIn first" });
    }
  } catch (error) {
    console.log(error);
  }
});

// get data into the cart
router.get("/cartdetails", authenticate, async (req, res) => {
  try {
    const buyuser = await User.findOne({ _id: req.userID });
    res.status(201).json(buyuser);
  } catch (error) {
    console.log(error);
  }
});

```

```

    }
  });

  // remove item from the cart

  router.delete("/remove/:_id", authenticate, async (req, res) => {
    try {
      const { _id } = req.params;

      req.rootUser.carts = req.rootUser.carts.filter((cruval) => {
        return cruval._id !== _id;
      });

      req.rootUser.save();
      res.status(201).json(req.rootUser);
      console.log("Item remove");
    } catch (error) {
      console.log(error + "jwt provide then remove");
      res.status(400).json(error);
    }
  });

  module.exports = router;

```

- **AUTHENTICATE**

authenticate.js

```

const jwt = require("jsonwebtoken");
const User = require("../models/user");
const secretkey = process.env.SECRET_KEY;

const authenticate = async (req, res, next) => {
  try {
    const token = req.cookies.Zaykaa;

    const verifyToken = jwt.verify(token, secretkey);

    const rootUser = await User.findOne({
      _id: verifyToken._id,
      "tokens.token": token,
    });

    if (!rootUser) {
      throw new Error("User Not Found");
    }

    req.token = token;

```

```
    req.rootUser = rootUser;
    req.userID = rootUser._id;

    next();
  } catch (error) {
    res.status(401).send("Unauthorized:No token provided");
    console.log(error);
  }
};

module.exports = authenticate;
```

auth.js

```
let bcrypt = require("bcryptjs");
let jwt = require("jsonwebtoken");

const encryptPassword = async (password) => {
  const hashPassword = await new Promise((resolve, reject) => {
    bcrypt.hash(password, 10, function (err, hash) {
      if (err) reject(err);
      resolve(hash);
    });
  });
  return hashPassword;
};

const generateJwt = async (id, obj = {}) => {
  let expiry = new Date();
  expiry.setDate(expiry.getDate() + 7);
  return jwt.sign(
    {
      _id: id,
      exp: parseInt(expiry.getTime()),
      ...obj,
    },
    process.env.SECRET_KEY
  );
};

const comparePasswords = async (password, hash) => {
  const val = await bcrypt.compare(password, hash);
  return val;
};

module.exports = { encryptPassword, generateJwt, comparePasswords };
```

- **FRONTEND ROUTES**

index.js

```

import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import Error from "./components/error";
import Dashboard from "./components/dashboard/user/dashboard";
import Home from "./components/header/home";
import AdminDashboard from "./components/dashboard/admin/dashboard";
import Header from "./components/header/header";
import Footer from "./components/footer/footer";
import Review from "./components/review/review";
import Help from "./components/help/help";
import Offer from "./components/offer/offer";
import About from "./components/about/about";
import Cart from "./components/cart/cart";
import NLCart from "./components/cart/nlcart";
import Profile from "./components/profile/profile";
import AddItem from "./components/dashboard/admin/additem";
import ViewItem from "./components/dashboard/admin/viewitem";
import UpdateItem from "./components/dashboard/admin/updateitem";
import AddCategory from "./components/dashboard/admin/addcategory";
import UpdateCategory from "./components/dashboard/admin/updatecategory";
import Items from "./components/product/product";
import Login from "./components/auth/user/login";
import PasswordReset from "./components/auth/user/passwordreset";
import ForgotPassword from "./components/auth/user/forgotpassword";
import AdminLogin from "./components/auth/admin/adminlogin";
import Signup from "./components/auth/user/signup";
import Contextprovider from "./components/context/ContextProvider";
import reportWebVitals from "./reportWebVitals";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
import "../node_modules/jquery/dist/jquery.min.js";
import "../node_modules/bootstrap/dist/js/bootstrap.min.js";

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById("root"));
const user = localStorage.getItem("token");
root.render(
  <Router>
    <Contextprovider>
      { !user && <Header /> }
      { user && <Home /> }
    <Routes>
      <Route path="/" element={ <Error /> } />

```

```

    <Route path="/" element={ <Dashboard /> } />
    <Route path="/additem" element={ <AddItem /> } />
    <Route path="/updateitem/:_id" element={ <UpdateItem /> } />
    <Route path="/addcategory" element={ <AddCategory /> } />
    <Route path="/updatecategory/:_id" element={ <UpdateCategory /> } />
    <Route path="/admin/home" element={ <AdminDashboard /> } />
    <Route path="/login" element={ <Login /> } />
    <Route path="/password-reset" element={ <PasswordReset /> } />
    <Route
      path="/forgot-password/:_id/:token"
      element={ <ForgotPassword /> }
    />
    <Route path="/admin/login" element={ <AdminLogin /> } />
    <Route path="/signup" element={ <Signup /> } />
    <Route path="/checkout" element={ <Cart /> } />
    <Route path="/checkout-login" element={ <NLCart /> } />
    <Route path="/profile" element={ <Profile /> } />
    <Route path="/viewitem" element={ <ViewItem /> } />
    <Route path="/category/:name" element={ <Items /> } />
    <Route path="/review" element={ <Review /> } />
    <Route path="/customer-support" element={ <Help /> } />
    <Route path="/about" element={ <About /> } />
    <Route path="/offer" element={ <Offer /> } />
  </Routes>
  <Footer />
</Contextprovider>
</Router>
);
reportWebVitals();

```

• HEADER

home.js

```

import "../header.css";
import logo from "../../images/bikelogo.png";
import React, { useContext, useEffect, useState } from "react";
import { Logincontext } from "../../context/ContextProvider";
import axios from "axios";

const Home = () => {
  const handleLogout = async () => {
    localStorage.removeItem("token");
    const res2 = await axios.get("/users/logout");
    if (res2.status !== 201) {
      console.log("error");
    } else {
      console.log("LOGOUT");
    }
  };

```

```

    window.location = "/login";
  }
};

const [userlocation, setUserlocation] = useState("");
const { account, setAccount } = useContext(Logincontext);

const getvaliduser = async () => {
  const res = await axios.get("/users/validuser");
  if (res.status !== 201) {
    console.log("error");
  } else {
    setAccount(res.data.carts);
  }
};

useEffect(() => {
  location();
  getvaliduser();
}, []);

const location = () => {
  const success = (position) => {
    // console.log(position);
    const latitude = position.coords.latitude;
    const longitude = position.coords.longitude;

    const geoApiUrl = `https://api.bigdatacloud.net/data/reverse-geocode-client?latitude=${latitude}&longitude=${longitude}&localityLanguage=en`;
    fetch(geoApiUrl)
      .then((res) => res.json())
      .then((data) => {
        setUserlocation(
          data.city +
            ", " +
            data.principalSubdivision +
            ", " +
            data.countryName +
            "(" +
            data.countryCode +
            ")"
        );
      });
  };
};

const error = () => {
  // console.log("error");
  setUserlocation("Unable to retrieve your location");
};

```



```

    };

    navigator.geolocation.getCurrentPosition(success, error);
  };

  return (
    <section className="sticky top-0 z-50">
      <nav className="mx-auto flex items-center justify-between py-1 shadow-sm appearance-none bg-white">
        <div className="flex items-center">
          <div className="ml-1 shrink-0 md:ml-6 lg:ml-8 xl:ml-10 w-56 md:w-60 lg:w-52 xl:w-48 icon">
            <a href="/">
              <img src={logo} alt="logo" />
            </a>
          </div>

          <div>
            <button onClick={location}>
              <svg
                xmlns="http://www.w3.org/2000/svg"
                viewBox="0 0 24 24"
                class="w-6 h-6 fill-red-500 ml-10"
              >
                <path
                  fill-rule="evenodd"
                  d="M11.54 22.351l.07.04.028.016a.76.76 0 00.723 .01.028-.015.071-.041a16.975 16.975 0 001.144-.742 19.58 19.58 0 002.683-2.282c1.944-1.99 3.963-4.98 3.963-8.827a8.25 8.25 0 00-16.5 0c0 3.846 2.02 6.837 3.963 8.827a19.58 19.58 0 002.682 2.282 16.975 16.975 0 001.145.742zM12 13.5a3 3 0 100-6 3 3 0 00 6z"
                  clip-rule="evenodd"
                />
              </svg>
            </button>
          </div>
          <div>
            <p className="mt-3 ml-2 font-light text-neutral-600 text-sm">
              {userlocation}
            </p>
          </div>
        </div>

        <div className="pt-3 mr-10">
          <ul className="flex items-center">
            <li className="ml-12 flex items-center">
              <svg
                xmlns="http://www.w3.org/2000/svg"
                fill="none"
                viewBox="0 0 24 24"

```

```

        stroke-width="1.5"
        stroke="currentColor"
        class="w-6 h-6"
      >
      <path
        stroke-linecap="round"
        stroke-linejoin="round"
        d="M9 14.25l6-6m4.5-3.493V21.75l-3.75-1.5-3.75 1.5-3.75-1.5-3.75 1.5V4.757c0-
1.108.806-2.057 1.907-2.185a48.507 48.507 0 0111.186 0c1.1.128 1.907 1.077 1.907 2.185zM9.75
9h.008v.008H9.75V9zm.375 0a.375.375 0 11-.75 0 .375.375 0 01.75 0zm4.125 4.5h.008v.008h-
.008V13.5zm.375 0a.375.375 0 11-.75 0 .375.375 0 01.75 0z"
      />
    </svg>
    <a
      href="/offer"
      className="text-dark decoration-transparent font-semibold ml-2"
    >
      Offers
    </a>
  </li>
  <li className="ml-12 flex items-center">
    <svg
      xmlns="http://www.w3.org/2000/svg"
      fill="none"
      viewBox="0 0 24 24"
      stroke-width="1.5"
      stroke="currentColor"
      class="w-6 h-6"
    >
      <path
        stroke-linecap="round"
        stroke-linejoin="round"
        d="M7.5 8.25h9m-9 3H12m-9.75 1.51c0 1.6 1.123 2.994 2.707 3.227 1.129.166 2.27.293
3.423.379.35.026.67.21.865.501L12 21l2.755-4.133a1.14 1.14 0 01.865-.501 48.172 48.172 0 003.423-
.379c1.584-.233 2.707-1.626 2.707-3.228V6.741c0-1.602-1.123-2.995-2.707-3.228A48.394 48.394 0
0012 3c-2.392 0-4.744.175-7.043.513C3.373 3.746 2.25 5.14 2.25 6.741v6.018z"
      />
    </svg>
    <a
      href="/review"
      className="text-dark decoration-transparent font-semibold ml-2"
    >
      Review
    </a>
  </li>
  <li className="ml-12 flex items-center">
    <svg
      xmlns="http://www.w3.org/2000/svg"

```

```

        fill="none"
        viewBox="0 0 24 24"
        stroke-width="1.5"
        stroke="currentColor"
        class="w-6 h-6"
      >
      <path
        stroke-linecap="round"
        stroke-linejoin="round"
        d="M16.712 4.33a9.027 9.027 0 011.652 1.306c.51.51.944 1.064 1.306 1.652M16.712 4.33l-
3.448 4.138m3.448-4.138a9.014 9.014 0 00-9.424 0M19.67 7.288l-4.138 3.448m4.138-3.448a9.014
9.014 0 010 9.424m-4.138-5.976a3.736 3.736 0 00-.88-1.388 3.737 3.737 0 00-1.388-.88m2.268
2.268a3.765 3.765 0 010 2.528m-2.268-4.796a3.765 3.765 0 00-2.528 0m4.796 4.796c-.181.506-
.475.982-.88 1.388a3.736 3.736 0 01-1.388.88m2.268-2.268l4.138 3.448m0 0a9.027 9.027 0 01-1.306
1.652c-.51.51-1.064.944-1.652 1.306m0 0l-3.448-4.138m3.448 4.138a9.014 9.014 0 01-9.424 0m5.976-
4.138a3.765 3.765 0 01-2.528 0m0 0a3.736 3.736 0 01-1.388-.88 3.737 3.737 0 01-.88-1.388m2.268
2.268L7.288 19.67m0 0a9.024 9.024 0 01-1.652-1.306 9.027 9.027 0 01-1.306-1.652m0 0l4.138-
3.448M4.33 16.712a9.014 9.014 0 010-9.424m4.138 5.976a3.765 3.765 0 010-2.528m0 0c.181-
.506.475-.982.88-1.388a3.736 3.736 0 011.388-.88m-2.268 2.268L4.33 7.288m6.406 1.18L7.288
4.33m0 0a9.024 9.024 0 00-1.652 1.306A9.025 9.025 0 004.33 7.288"
      />
    </svg>
    <a
      href="/customer-support"
      className="text-dark decoration-transparent font-semibold ml-2"
    >
      Help
    </a>
  </li>
  <li className="ml-12 flex items-center">
    <svg
      xmlns="http://www.w3.org/2000/svg"
      fill="none"
      viewBox="0 0 24 24"
      stroke-width="1.5"
      stroke="currentColor"
      class="w-6 h-6"
    >
    <path
      stroke-linecap="round"
      stroke-linejoin="round"
      d="M17.982 18.725A7.488 7.488 0 0012 15.75a7.488 7.488 0 00-5.982 2.975m11.963 0a9 9
0 10-11.963 0m11.963 0A8.966 8.966 0 0112 21a8.966 8.966 0 01-5.982-2.275M15 9.75a3 3 0 11-6 0
3 0 016 0z"
    />
    </svg>
    <a
      href="/profile"

```

```

        className="text-dark decoration-transparent font-semibold ml-2"
      >
        Profile
      </a>
    </li>
    <li className="ml-12 flex items-center">
      <svg
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        stroke-width="1.5"
        stroke="currentColor"
        class="w-6 h-6"
      >
        <path
          stroke-linecap="round"
          stroke-linejoin="round"
          d="M15.75 9V5.25A2.25 2.25 0 013.5 3h-6a2.25 2.25 0 00-2.25 2.25v13.5A2.25 2.25 0
007.5 21h6a2.25 2.25 0 02.25-2.25V15m3 0l3-3m0 0l-3-3m3 3H9"
        />
      </svg>
      <button
        className="text-dark decoration-transparent font-semibold ml-2"
        onClick={handleLogout}
      >
        Log out
      </button>
    </li>
  </div>
  <div
    className="fas fa-bars px-2.5 py-2 bg-zinc-200 text-xl hover:text-slate-50
rounded transition duration-200 text-zinc-800 hidden"
    id="menu-btn"
  ></div>
  <li className="ml-12">
    <a href="/checkout">
      <div>
        <div
          className="fas fa-shopping-cart shopping-cart text-xl px-2.5 py-2.5 bg-zinc-200 hover:bg-
orange-500 hover:text-slate-50
rounded transition duration-200 text-zinc-800"
        ></div>
        <div class="inline-flex absolute top-2 right-7 justify-center items-center w-6 h-6 text-xs font-
bold text-white bg-red-500 rounded-full border-2 border-white dark:border-gray-900">
          {account.length}
        </div>
      </div>
    </a>
  </li>

```

```

    </ul>
  </div>
</nav>
</section>
);
};

```

```
export default Home;
```

• USER DASHBOARD

dashboard.js

```

import "../dashboard.css";
import axios from "axios";
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import dashboard1 from "../images/dashboard1.PNG";
import dashboard2 from "../images/dashboard2.PNG";
import dashboard3 from "../images/dashboard3.PNG";
import delivery from "../images/delivery.png";

function Dashboard() {
  const [category, setCategory] = useState([]);

  const getCategory = async () => {
    const response = await axios.get("/category/getcategory");
    console.log(response.data.data);
    setCategory(await response.data.data);
  };

  const searchHandle = async (e) => {
    let key = e.target.value;
    if (key) {
      let result = await axios.get(`/category/search/${key}`);
      if (result) {
        setCategory(result.data);
      }
    } else {
      getCategory();
    }
  };

  useEffect(() => {
    getCategory();
  }, []);

  return (

```

```

<div className="bg-slate-900">
  <div className="md:shrink-0 mx-16 grid grid-cols-2 md:grid-cols-4 lg:grid-cols-4 xl:grid-cols-4
gap-10">
    { /* <div className="my-10 transition duration-500 hover:scale-105 w-full"> */}
    <img
      src={dashboard3}
      width={259}
      className="rounded my-10 transition duration-500 hover:scale-105"
      alt=""
    />
    { /* </div> */}
    { /* <div className="my-10 transition duration-500 hover:scale-105 w-full"> */}
    <img
      src={dashboard2}
      height={259}
      width={259}
      className="rounded my-10 transition duration-500 hover:scale-105"
      alt=""
    />
    { /* </div> */}
    { /* <div className="my-10 transition duration-500 hover:scale-105"> */}
    <img
      src={dashboard1}
      height={256}
      width={256}
      className="rounded my-10 transition duration-500 hover:scale-105"
      alt=""
    />
    { /* </div> */}
    { /* <div className="my-10 transition duration-500 hover:scale-105"> */}
    <img
      src={delivery}
      height={256}
      width={256}
      className="rounded my-10 transition duration-500 hover:scale-105"
      alt=""
    />
    { /* </div> */}
  </div>
</div>

<section className="my-10">
  <div className="container flex justify-between justify-center border-b">
    <h2 className="mx-0 pb-4 font-semibold border-b-zinc-400">
      Categories
    </h2>

```

```

<div className="flex justify-center pl-16 md:block">
  <div className="mt-3 w-70">
    <div className="input-group relative flex flex-wrap items-stretch mb-3 rounded">
      <input
        type="search"
        className="relative flex-auto appearance-none border rounded px-3 py-1.5 leading-tight
text-gray-700 focus:outline-none focus:shadow-outline"
        placeholder="Search for category"
        aria-label="Search"
        aria-describedby="button-addon2"
        onChange={ searchHandle }
      />
      <div className="pl-2 py-2">
        <span id="basic-addon2">
          <svg
            aria-hidden="true"
            focusable="false"
            data-prefix="fas"
            data-icon="search"
            className="w-4"
            role="img"
            xmlns="http://www.w3.org/2000/svg"
            viewBox="0 0 512 512"
          >
            <path
              fill="currentColor"
              d="M505 442.7L405.3 343c-4.5-4.5-10.6-7-17-7H372c27.6-35.3 44-79.7 44-128C416
93.1 322.9 0 208 0 93.1 0 208 93.1 208 208 208c48.3 0 92.7-16.4 128-44v16.3c0 6.4 2.5 12.5 7
17 199.7 99.7c9.4 9.4 24.6 9.4 33.9 0 128.3-28.3c9.4-9.4 9.4-24.6 1-34zM208 336c-70.7 0-128-57.2-128-
128 0-70.7 57.2-128 128-128 70.7 0 128 57.2 128 128 0 70.7-57.2 128-128 128z"
            ></path>
          </svg>
        </span>
      </div>
    </div>
  </div>
</div>

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 col-gap-12 gap-y-11
mx-16 mt-10 mb-14">
  {category.length > 0 ? (
    category.map((cate) => {
      return (
        <div>
          <div className="rounded-2xl hover:shadow-2xl border-transparent border-2 hover:border-
neutral-300 hover:border-solid transition duration-300 ease-in-out w-full md:w-64">
            <Link to={` /category/${cate.name} `}>

```

```

      <img
        className="h-40 mb-4 mx-auto"
        src={`../uploads/${cate.imgpath}`}
        alt=""
      />
    </Link>
    <div className="text-center content p-7 pt-0">
      <div className="title">{cate.name}</div>
      <div className="sub-title">{cate.subname}</div>
      <div className="bottom">
        <p>{cate.desc}</p>
      </div>
    </div>
  </div>
</div>
);
})
):(
  <h3 className="text-center"> No Result Found </h3>
)
</div>
</section>
</>
);
}

```

```
export default Dashboard;
```

• CART

cart.js

```

import "../cart.css";
import ecart from "../../images/ecart.PNG";
import axios from "axios";
import { useContext, useState, useEffect } from "react";
import Subtotal from "../subtotal";
import cart from "../../images/cart-black.png";
import { Logincontext } from "../context/ContextProvider";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

function Cart() {
  const [cartdata, setCartdata] = useState("");
  const { account, setAccount } = useContext(Logincontext);

  const getcartdata = async () => {
    const res = await axios.get("/cart/cartdetails");

```



```

    if (res.status !== 201) {
      console.log("error");
    } else {
      setCartdata(res.data.carts);
    }
  };

```

```

const removedata = async (_id) => {
  try {
    const res = await axios.delete(`/cart/remove/${_id}`);
    console.log(res);
    if (res.status === 400 || !res) {
      console.log("error");
    } else {
      setAccount(res.data.carts);
      getcartdata();
      toast.error("Item remove from cart!", {
        position: "top-right",
        autoClose: 2500,
        theme: "colored",
      });
    }
  } catch (error) {
    console.log(error);
  }
};

```

```

useEffect(() => {
  getcartdata();
}, []);

```

```

return (
  <>
    { cartdata.length ? (
      <section className="cart py-16">
        <div className="mx-auto lg:w-1/2">
          <div className="container">
            <div className="flex items-center border-b border-gray-300 pb-4">
              <img src={cart} alt="cart" />
              <h1 className="font-bold ml-4 text-2xl">Order summary</h1>
            </div>
            <div className="pizza-list">
              { cartdata.map((cart, key) => {
                return (
                  <>
                    <div className="flex items-center my-8">
                      <img

```

```

        className="w-36 rounded-lg mr-5"
        src={`\uploads/${cart.imgpath}`}
        alt=""
    />
    <div className="flex-1 ml-4">
        <h1 className="text-lg">{cart.name}</h1>
        <span>{cart.size}</span>
    </div>
    <span className="flex-1 ml-3">1 Pcs</span>
    <div className="flex-1">
        <button onClick={() => removedata(cart._id)}>
            <svg
                xmlns="http://www.w3.org/2000/svg"
                fill="none"
                viewBox="0 0 24 24"
                stroke-width="1.5"
                stroke="red"
                class="w-6 h-6"
            >
                <path
                    stroke-linecap="round"
                    stroke-linejoin="round"
                    d="M14.74 9l-.346 9m-4.788 0L9.26 9m9.968-3.21c.342.052.682.107 1.022.166m-1.022-.165L18.16 19.673a2.25 2.25 0 01-2.244 2.077H8.084a2.25 2.25 0 01-2.244-2.077L4.772 5.79m14.456 0a48.108 48.108 0 00-3.478-.397m-12 .562c.34-.059.68-.114 1.022-.165m0 0a48.11 48.11 0 013.478-.397m7.5 0v-.916c0-1.18-.91-2.164-2.09-2.201a51.964 51.964 0 00-3.32 0c-1.18.037-2.09 1.022-2.09 2.201v.916m7.5 0a48.667 48.667 0 00-7.5 0"
                />
            </svg>
        </button>
    </div>
    <span className="font-bold text-lg">₹{cart.price}</span>
</div>
</>
);
}}}
</div>
<hr />
<div className="text-right">
    <Subtotal item={cartdata} />

<div>
    <form className="mt-12">
        <div className="relative w-1/2 ml-auto mb-4">
            <select className="block appearance-none w-full bg-white border border-gray-400
                hover:border-gray-500 px-4 py-2 pr-8 rounded leading-tight focus:outline-none focus:shadow-outline">
                <option value="cod">Cash on delivery</option>
            </select>

```

```

      <div className="pointer-events-none absolute inset-y-0 right-0 flex items-center px-2
text-gray-700">
        <svg
          className="fill-current h-4 w-4"
          xmlns="http://www.w3.org/2000/svg"
          viewBox="0 0 20 20"
        >
          <path d="M9.293 12.951 7.071 15.657 8 14.141 10 10.828 5.757 6.586 4.343
8z" />
        </svg>
      </div>
    </div>
    <input
      name="phone"
      className="border border-gray-400 p-2 w-1/2 mb-4"
      type="text"
      placeholder="Phone number"
    />
    <br />
    <input
      name="address"
      className="border border-gray-400 p-2 w-1/2"
      type="text"
      placeholder="Address"
    />
    <div className="mt-4">
      <button
        className="bg-primary1 hover:bg-primaryhover rounded-full text-white font-bold py-2
px-6 focus:outline-none focus:shadow-outline"
        type="submit"
      >
        Order Now
      </button>
    </div>
  </form>
</div>
</div>
</div>
</div>
<ToastContainer />
</section>
) : (
  <section className="cart py-16">
    <div className="mb-36">
      <div>
        <div className="empty-cart-cls text-center ">
          <img
            src={ecart}

```

```

        width={270}
        height={270}
        className="mx-auto mt-4"
      />
      <p className="text-xl text-neutral-600 font-semibold mb-1">
        Your cart is empty
      </p>
      <p className="text-neutral-500 text-sm">
        You can go to home page to view more cuisines or food
      </p>
      <br></br>
      <a
        href="/"
        className="bg-orange-500 hover:bg-orange-600 rounded text-white font-bold py-2 px-7
focus:outline-none focus:shadow-outline decoration-transparent transition duration-300"
      >
        CONTINUE SHOPPING
      </a>
    </div>
  </div>
</div>
</section>
  )}
</>
);
}
export default Cart;

```

nlcart.js

```

import pizza from "../images/pizza.png";

function NLCart() {
  return (
    <>
    <section className="cart py-16 bg-zinc-100">
      { /* <div className="mb-36">
        <div>
          <div className="empty-cart-cls text-center ">
            <img src="" width={270} height={270} className="mx-auto mt-4" />
            <p className="text-xl text-neutral-600 font-semibold mb-1">
              Your cart is empty
            </p>
            <p className="text-neutral-500 text-sm">
              You can go to home page to view more cuisines or food
            </p>
            <br></br>
            <a

```

```

      href="/"
      className="bg-orange-500 hover:bg-orange-600 rounded text-white font-bold py-2 px-7
focus:outline-none focus:shadow-outline decoration-transparent transition duration-300"
    >
      CONTINUE SHOPPING
    </a>
  </div>
</div>
</div> */}
<div className="flex">
  <div className="left my-24 ml-44">
    <p className="italic font-light mb-3">Are you hungry?</p>
    <p className="text-5xl font-extrabold">Don't Wait!</p>
    <div className="mt-5">
      <a
        href="/"
        className="bg-orange-500 hover:bg-orange-600 rounded-full text-white font-bold py-2.5 px-
6 focus:outline-none focus:shadow-outline decoration-transparent transition duration-300"
      >
        Order Now
      </a>
    </div>
    <div className="mt-4">
      <p className="font-extralight italic">
        To order you have to login first, to login{ " "}
      <a
        href="/login"
        className=" hover:text-cyan-600 decoration-transparent transition duration-300"
      >
        Click Here
      </a>
    </p>
    </div>
  </div>
  <div className="right ml-48">
    <img src={pizza} width={500} />
  </div>
</div>
</section>
</>
);
}
export default NLCart;

```

subtotal.js

```

import React from "react";
import { useEffect, useState } from "react";

```

```

const Subtotal = ({ item }) => {
  const [price, setPrice] = useState(0);

  useEffect(() => {
    totalAmount();
  }, [item]);

  const totalAmount = () => {
    let cost = 0;
    item.map((items) => {
      cost += parseInt(items.price);
    });
    setPrice(cost);
  };

  return (
    <div className="text-right">
      <span className="text-lg font-bold">
        Total Amount ({item.length} items):
      </span>
      <span className="amount text-2xl font-bold ml-2 text-primary1">
        ₹{price}
      </span>
    </div>
  );
};

export default Subtotal;

```

• ADMIN DASHBOARD

dashboard.js

```

import "./dashboard.css";
import axios from "axios";
import React, { useEffect, useState } from "react";

function Dashboard() {
  const [category, setCategory] = useState([]);

  const getCategory = async () => {
    const response = await axios.get("/category/getcategory");
    console.log(response.data.data);
    setCategory(await response.data.data);
  };

  useEffect(() => {

```

```

    getCategory();
  }, []);

const setData = (data) => {
  let { _id, name, subname, desc } = data;
  localStorage.setItem("ID", _id);
  localStorage.setItem("name", name);
  localStorage.setItem("subname", subname);
  localStorage.setItem("desc", desc);
};

const onDelete = (_id) => {
  axios.delete(`/category/deletcategory/${_id}`).then(() => {
    getCategory();
  });
};

return (
  <
    <section className="my-10">
      <h2 className="mx-16 pb-4 font-semibold border-b-zinc-400 border-b">
        Categories
      </h2>

      <a
        href="/addcategory"
        className="bg-primary1 decoration-transparent ml-16 mt-2 hover:bg-primaryhover rounded-full
text-white font-bold py-2 px-6 focus:outline-none focus:shadow-outline"
        type="button"
      >
        Add Category
      </a>

      <a
        href="/additem"
        className="bg-primary1 decoration-transparent mx-5 mt-2 hover:bg-primaryhover rounded-full
text-white font-bold py-2 px-6 focus:outline-none focus:shadow-outline"
        type="button"
      >
        Add Item
      </a>

      <a
        href="/viewitem"
        className="bg-primary1 decoration-transparent mx-2 mt-2 hover:bg-primaryhover rounded-full
text-white font-bold py-2 px-6 focus:outline-none focus:shadow-outline"
        type="button"
      >

```

View Item


```

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 col-gap-12 gap-y-11
mx-16 mt-10 mb-14">
  {category.map((cate) => {
    return (
      <div>
        <div className="rounded-2xl hover:shadow-2xl border-transparent border-2 hover:border-
neutral-300 hover:border-solid transition duration-300 ease-in-out w-full md:w-64">
          <img
            className="h-40 mb-4 mx-auto"
            src={`\uploads/${ cate.imgpath }`}
            alt=""
          />
          <div className="text-center content p-7">
            <div className="title">{ cate.name }</div>
            <div className="sub-title">{ cate.subname }</div>
            <div className="bottom">
              <p>{ cate.desc }</p>
            </div>
            <div className="bottom flex items-center justify-around mt-6 mb-6">
              <a href={`\updatecategory/" + cate._id`}>
                <button
                  onClick={() => setData(cate)}
                  className="hover:bg-primary1 hover:border-pure hover:text-pure transition duration-
200 py-1 px-6 font-medium items-center add-to-cart"
                >
                  <span className="font-medium">Modify</span>
                </button>
              </a>
              <button
                data-bs-toggle="modal"
                data-bs-target="#exampleModalCenter"
                className="hover:bg-red-600 hover:border-pure border-red-600 delete text-red-600
hover:text-pure transition duration-200 py-1 px-6 font-medium items-center"
              >
                <span className="font-medium">Delete</span>
              </button>
            <div
              class="modal fade backdrop-blur-sm fixed top-0 left-0 hidden w-full h-full outline-none
overflow-x-hidden overflow-y-auto"
              id="exampleModalCenter"
              tabindex="-1"
              aria-labelledby="exampleModalCenterTitle"
              aria-modal="true"
              role="dialog"
            >

```



```

<div class="modal-dialog modal-dialog-centered relative w-auto pointer-events-none">
  <div class="modal-content border-none shadow-lg relative flex flex-col w-full pointer-
events-auto bg-white bg-clip-padding rounded-md outline-none text-current">
    <div class="flex flex-shrink-0 items-center justify-between p-4">
      <button
        type="button"
        class="btn-close box-content ml-auto w-4 h-4 p-1 text-black border-none rounded-
none opacity-50 focus:shadow-none focus:outline-none focus:opacity-100 hover:text-black
hover:opacity-75 hover:no-underline"
        data-bs-dismiss="modal"
        aria-label="Close"
      ></button>
    </div>
    <svg
      aria-hidden="true"
      class="mb-4 mx-auto w-14 h-14 text-gray-400 dark:text-gray-200"
      fill="none"
      stroke="currentColor"
      viewBox="0 0 24 24"
      xmlns="http://www.w3.org/2000/svg"
    >
      <path
        stroke-linecap="round"
        stroke-linejoin="round"
        stroke-width="2"
        d="M12 8v4m0 4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z"
      ></path>
    </svg>
    <div class="relative">
      <h3 class="text-lg font-normal text-center text-gray-500 dark:text-gray-400">
        Are you sure you want to delete this category?
      </h3>
    </div>
    <div class="flex flex-shrink-0 flex-wrap justify-center items-center p-4">
      <button
        type="button"
        onClick={() => onDelete(cate._id)}
        data-bs-dismiss="modal"
        class="text-white bg-red-600 hover:bg-red-800 focus:ring-4 focus:outline-none
focus:ring-red-300 font-medium rounded-lg text-base inline-flex items-center px-4 py-2.5 text-center
mr-2"
      >
        Yes, I'm sure
      </button>
      <button
        type="button"

```

```
export default Dashboard;
```

2.1. FUNCTIONAL TESTING

Test case id	Description	Inputs	Expected output
Tc1	1. Search the item gallery to decide which items to order.	Search string	List of items searched are displayed
Tc2	1. Register on the website to place an order	Email Id, password, shipping details, and billing details.	If the information entered is correct, user is registered successfully, otherwise an appropriate error message is displayed.
Tc3	1. Log into the website.	Email id, password	Item is successfully added in the
	2. Select item to be purchased and its quantity	Item Id, item name, quantity	

	3. Add selected item to the shopping cart	-	shopping cart.
Tc4	1. Log into the website.	Email id, password	Item is not added in the shopping cart.
	2. Select item to be purchased and its quantity	Item Id, item name, quantity	
	3. Do not Add selected item to the shopping cart	-	
Tc5	1. Log into the website.	Email id, password	Items are successfully added in the shopping cart.
	2. Select item to be purchased and its quantity	Item Id, item name, quantity	
	3. Add selected item to the shopping cart	Item Id, item name, quantity	
	4. Select some more items and add them to the shopping cart.	Item Id, item name, quantity	
Tc6	1. Log into the website.	Email id, password	If deletion is confirmed, item is successfully deleted from the shopping cart.
	2. Select 2 or more items to be purchased and Their quantity	Item number, item name, quantity	
	3. Add selected items to the shopping cart	Item Id, item name, quantity	
	4. Delete one item from the shopping cart.	Item Id	
Tc7	1. Log into the website.	Email id, password	If updation is confirmed, quantity is successfully updated.
	2. Select item to be purchased and its quantity	Item Id, item name, quantity	
	3. Add selected item to the shopping cart	Item Id, item name, quantity	
	4. Update quantity of the item added to the shopping cart.	Quantity	

Table 3.1: Functional Test Cases of Order Process of a Food Ordering Website.

2.2. USER INTERFACE TESTING

User interface testing checklist may provide the opportunities for testers to ensure the proper functioning of user interface features such as hyperlinks, tables, frames, forms, buttons, menus, dialog boxes and error messages. A checklist may also discipline and organize testing activities. However organizations and website owners may modify the checklist according to requirements of their web application.

S. no	Description	Yes/ No/ NA	Remarks
Hyperlinks			
1.	Are the links meaningful?	Yes	
2.	Are there any broken links?	No	
3.	Do all internal links work correctly?	Yes	
4.	Do all external links work correctly?	Yes	
5.	Are all links to external sites in the website tested?	Yes	
6.	Are images correctly hyperlinked?	Yes	
7.	Does every hyperlink exist on the site map?	Yes	
8.	Are the hyperlink's color standard	Yes	
9.	Does the link bring the user to the correct web page?	Yes	
10.	Can the user navigate using text only?	Yes	
Tables			
11.	Are the columns wide enough or the text wraps around the rows?	Yes	
12.	Are the row and columns headings of tables appropriate?	Yes	
13.	Are the complex tables broken down into simpler ones, wherever required?	Yes	
14.	Does the user have to scroll right constantly in order to see the contents in a table?	Yes	
15.	Are the captions meaningful?	Yes	
Frames			
16.	Is every frame associated with a title?	Yes	
17.	Can the user resize the frame?	Yes	
18.	Is the frame size appropriate?	Yes	
19.	Does the horizontal and vertical scrollbar appear wherever required?	Yes	
20.	Does any frame handling mechanism exist for browsers that do not support frames?	Yes	
Forms			
21.	Are keyboard shortcuts provided for movement between different fields of forms?	No	
22.	Are the mandatory fields marked clearly?	Yes	
23.	Are descriptive labels for all fields provided?	Yes	
24.	Is information formatted , wherever required?	Yes	

25.	Are error messages meaningful and appropriate?	Yes	
26.	Does the size of text fields give enough room for the user to type?	Yes	
27.	Are fields used appropriately?	Yes	
28.	Is any information asked more than once in the form? Is the user prevented from entering the same data multiple times?	No	
29.	Does the form include 'reset' button to clear its contents?	No	
Text fields, buttons, list boxes, check boxes			
30.	Does the text field accept invalid characters and special characters?	No	
31.	Can text be selected using shift + arrow key?	Yes	
32.	Is the user able to select any combination of options in check boxes?	No	
33.	Can the user select more than 1 option in radio buttons?	No	
34.	Does the button click trigger the required action?	Yes	
35.	Can the user add text in the combo boxes?	No	
36.	Can the user add text in the list boxes?	No	
37.	Do the required commands and options exist in each menu?	Yes	
38.	Are abbreviations used in list boxes/ buttons?	No	
39.	Are the label names meaningful?	Yes	
40.	Are mouse actions consistent across web pages?	Yes	
41.	Is red color used to highlight active items?	No	
42.	Is all the data inside the list/ combo box listed in chronological order?	Yes	
43.	Are validation checks for text fields present?	Yes	
44.	Do fields with numeric values handle upper and lower range of values appropriately?(BVA)	Yes	
45.	Does the back navigation button work as required?	Yes	
46.	Do the text fields accept maximum permissible data?	Yes	
47.	Can an alphanumeric character be entered in numeric fields?	No	
48.	Are the command buttons disabled when they are not in use?	No	
49.	Is there any spelling or grammatical mistakes in captions or labels?	No	


Table 3.2: Checklist for testing user interfaces

2.3. NAVIGATION TESTING

Test case id	Description	Inputs	Expected output
Tc1	Check all links on each webpage.	Link 1 = About Link 2 = Offers Link 3 = Help Link 4 = Sign In Link 6 = Shopping Cart Link 7= Review Link 8= Profile	Appropriate web page is opened with respect to each link.
Tc2	Click on all links on each web page to test the appearance of content on each web page.	Link 1 = About Link 2 = Offers Link 3 = Help Link 4 = Sign In Link 6 = Shopping Cart Link 7= Review Link 8= Profile	Appropriate horizontal and vertical scroll bars are present and the user can view the page contents properly.
Tc 3	Search for category, items in the available home page and particular category respectively.	Search string	The user is able to navigate across multiple search pages successfully.
Tc4	Click on 'back' link present on each page.		The appropriate page is displayed.

Table 3.3: Navigation Test Cases of a Food Ordering Website.

2.4. FORM TESTING



[About](#) [Offers](#) [Help](#) [Sign In](#) [Cart](#)

Sign up

or [login to your account](#)

Name *

Email *

Mobile Number *

Password *

[Sign Up](#)

Get connected with us on social networks:

[Facebook](#) [Twitter](#) [Google](#) [Instagram](#) [Pinterest](#)

ABOUT	CATEGORIES	USEFUL LINKS	CONTACT
Zaykaa is Online Food Website. Launched in 2022, Customers will order food online from Zaykaa. Customers will write reviews about food.	Italian Chinese Indian Fast Food Dessert	Pricing Settings Orders Help	New Delhi, 110059, India zaykaa@gmail.com +91 9988776655

© 2022 Zaykaa. All rights reserved.

Fig 3.1: Registration Form

Test case id	Description	Inputs	Expected output
Tc1	Navigate using tabs from one field to another in the form	-	The user follows the correct sequence while navigating from one field to another.
Tc2	Check maximum and minimum length of all fields in the Form.	Email id, password, shipping details, and billing details.	If the characters are entered within the range, the info. Is added successfully; otherwise an appropriate error
			Message displayed.
TC3	Check data validations of all fields in the form	Login id, password, shipping details, and billing details.	If the Characters are valid The information is added successfully, otherwise an appropriate error message is displayed
TC4	Check whether all the mandatory fields are entered in the form.	Email id, password, confirm password, shipping details, and billing details.	If the required fields are entered the Information is added successfully, otherwise an appropriate error message is displayed

Table 3.4 : Test Cases of Registration Form

2.5. DATABASE TESTING

Web applications, many applications are database driven. It is important for these applications to provide security to the user's sensitive data such as personal details and credit card information.

For example, consider the example for purchasing items from an online store. The user performs a search based on some keywords and price preferences; the database server initiates a database query. Suppose due to some programming fault in the query, the query does not consider the price preferences given by the customer, this will produce erroneous results. These types of faults must be tested and removed during database testing.

Important issues in database testing may include:

- i. Data validation
- ii. Data consistency
- iii. Data integrity
- iv. Concurrency control and recovery
- v. Data manipulation operations such as addition, deletion, updating and retrieval of data.
- vi. Database security

A database must be tested for administrative level operations such as adding, deleting and updating an item in the database, and user operations such as searching an item from the database or providing personal details. In the example of the online shopping website, the most common administrative operations and user operations include:

A. Administrative operations

- i. Inserting a new item into the database
- ii. Deleting an existing item from the database
- iii. Updating an existing item from the database
- iv. Viewing an existing item from the database

B. User operations

- i. searching items from the database
- ii. registering into the website involves storing the user's personal details
- iii. placing an order involves storing user preferences and purchase details into the database
- iv. providing feedback involves storing information in the database
- v. tracking status of the order placed

Test case id	Description(steps followed)	Inputs	Expected output
Tc1	Search the product gallery to decide which items to purchase	Search string	List of items searched are correctly displayed from the database That satisfy the search criteria selected by the user.
Registering of user on the website			
Tc2	Register on the website to place an order	Email id, password, mobile number, shipping details, billing details	If the information entered is valid, the user information is successfully added into the database
Providing feedback on this website			
Tc3	Enter fields	-	This information is successfully added to database.

Table 3.5: Sample Test Cases Based on a User Operation

3. PROJECT PLANNING ACTIVITIES

It defines design as to make preliminary sketches of; to sketch a pattern or outline for plan. To plan and carry out especially by artistic arrangement or in a skilful way. System analysis and design can be characterized as a set of techniques and processes, a community of interests, a culture and an intellectual orientation.

The various tasks in the system analysis include the following.

1. Understanding application.
2. Planning.
3. Scheduling.
4. Performing trade studies.

CHAPTER – 4

SCOPE OF IMPROVEMENT, SUMMARY AND CONCLUSIONS

4.1 SCOPE OF IMPROVEMENT:

- We can give more advance software for Online Food Ordering System including more facilities.
- We will host the platform on online server to make it accessible worldwide.
- Implement the backup mechanism for taking backup of database on a regular basis.
- We can give online payment facility for easy payment.
- We can provide real time delivery tracking facilities.

4.2 SUMMARY:

The basic aim in the system was to provide all improved functionality and flavor system minus the entire drawbacks or shortcoming analysed. With front end like React & Javascript and backend like NodeJs, ExpressJs & Mongo Database, most of the major irritants due the conventional file system, were removed without many effort. Providing a web-based food ordering function where a customer can order food through the online system without going at restaurant.

4.3 CONCLUSIONS:

An online food ordering system is developed where the customers can make an order for the food and avoid the hassles of waiting for the order to be taken by the waiter. Using the application, the end users register online, read the category items, and select the food for ordering online. Once the customer selects the required food item the restaurant will be able to see the results on the screen and start processing the food. This application nullifies the need of a waiter or reduces the workload of the waiter. The advantage is that in a crowded restaurant there will be chances that the waiters are overloaded with orders, and they are unable to meet the requirements of the customer in a satisfactory manner. Therefore, by using this application the users can directly place the order for food to the restaurant online. In conclusion restaurant won't have to pay any charges or commission on any orders, and increasing their profit margins.

REFERENCES

- 1) Bootstrap - <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
- 2) Tailwind CSS - <https://tailwindui.com/components?ref=sidebar>
- 3) Swiggy - <https://www.swiggy.com/>
- 4) Zomato - <https://www.zomato.com/>
- 5) K.K. Aggarwal & Yogesh Singh, “Software Engineering”, 2nd edition, New Age International, 2005.
- 6) R.S. Pressman, “Software Engineering – A practitioner’s approach”, 5th edition, PHI Learning Private Limited.
- 7) Jeffery C. Jackson, “Web Technologies: A Computer Science Perspective”, Pearson.
- 8) Internet and World Wide Web Deitel HM, Deitel, Goldberg, Third edition.