

CRAFTEDEUPHORIA

Online Gift Store

Mini Project Report

Submitted by

Sradha Ann George

Reg. No.: AJC19MCA-I055

In Partial fulfillment for the Award of the Degree of

INTEGRATED MASTER OF COMPUTER APPLICATIONS

(INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with 'A' grade. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2023-2024

DEPARTMENT OF COMPUTER APPLICATIONS
AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY



CERTIFICATE

This is to certify that the Project report, “**CRAFTEDEUPHORIA**” is the bona fide work of **SRADHA ANN GEORGE (Regno:AJC19MCA-I055)** in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2019-24.

Ms. Nimmy Francis

Internal Guide

Ms. Meera Rose Mathew

Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report “**CRAFTEDEUPHORIA**” is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Integrated Master of Computer Applications (INMCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date: 30-10-2023

SRADHA ANN GEORGE

KANJIRAPPALLY

Reg: AJC19MCA-I055

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lilly Kutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Nimmy Francis** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

SRADHA ANN GEORGE

ABSTRACT

It creates a user-friendly, intuitive, and responsive online gift store with a comprehensive catalog of products for various occasions. The checkout process is designed with security in mind, incorporating industry-standard encryption and authentication measures. User registration and authentication allow users to create accounts securely, facilitating personalized experiences and order tracking. The system uses data-driven algorithms to provide personalized product recommendations, enhancing user engagement and the likelihood of successful purchases.

Social sharing and gift ideas encourage community engagement, potentially driving more traffic to the platform. The administrative panel provides efficient order management capabilities, allowing administrators to view, track, and manage incoming orders in real-time. The system also includes tracking shipping and delivery, generating invoices for completed orders, and implementing a user review and rating system. A subscription system and advanced personalization options can further enhance the platform's marketing and customer retention efforts.

The project includes three modules:

- Users
- Admin
- Community

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	2
2	SYSTEM STUDY	4
2.1	INTRODUCTION	5
2.2	EXISTING SYSTEM	5
2.3	DRAWBACKS OF EXISTING SYSTEM	7
2.4	PROPOSED SYSTEM	7
2.5	ADVANTAGES OF PROPOSED SYSTEM	8
3	REQUIREMENT ANALYSIS	9
3.1	FEASIBILITY STUDY	10
3.1.1	ECONOMICAL FEASIBILITY	10
3.1.2	TECHNICAL FEASIBILITY	11
3.1.3	BEHAVIORAL FEASIBILITY	11
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	12
3.2	SYSTEM SPECIFICATION	15
3.2.1	HARDWARE SPECIFICATION	15
3.2.2	SOFTWARE SPECIFICATION	15
3.3	SOFTWARE DESCRIPTION	15
3.3.1	PYTHON	15
3.3.2	DJANGO	15
3.3.3	SQULITE	16
4	SYSTEM DESIGN	17
4.1	INTRODUCTION	18
4.2	UML DIAGRAM	18
4.2.1	USE CASE DIAGRAM	19
4.2.2	SEQUENCE DIAGRAM	21
4.2.3	STATE CHART DIAGRAM	23
4.2.4	ACTIVITY DIAGRAM	24
4.2.5	CLASS DIAGRAM	26
4.2.6	OBJECT DIAGRAM	28

4.2.7	COMPONENT DIAGRAM	29
4.2.8	DEPLOYMENT DIAGRAM	31
4.2.9	COLLABORATION DIAGRAM	33
4.3	USER INTERFACE DESIGN USING FIGMA	33
4.4	DATABASE DESIGN	36
5	SYSTEM TESTING	42
5.1	INTRODUCTION	43
5.2	TEST PLAN	43
5.2.1	UNIT TESTING	44
5.2.2	INTEGRATION TESTING	44
5.2.3	VALIDATION TESTING	45
5.2.4	USER ACCEPTANCE TESTING	45
5.2.5	AUTOMATION TESTING	45
5.2.6	SELENIUM TESTING	46
6	IMPLEMENTATION	54
6.1	INTRODUCTION	55
6.2	IMPLEMENTATION PROCEDURE	55
6.2.1	USER TRAINING	56
6.2.2	TRAINING ON APPLICATION SOFTWARE	56
6.2.3	SYSTEM MAINTENANCE	56
7	CONCLUSION & FUTURE SCOPE	57
7.1	CONCLUSION	58
7.2	FUTURE SCOPE	59
8	BIBLIOGRAPHY	60
9	APPENDIX	62
9.1	SAMPLE CODE	63
9.2	SCREEN SHOTS	87

List of Abbreviation

HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheet
SQLITE	–	Structured Query Language LITE
UML	–	Unified Modelling Language
JS	–	JavaScript
AJAX	–	Asynchronous JavaScript and XML Environment

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

CraftedEuphoria, as an innovative online gift store project, is not just about shopping; it's a holistic experience. Beyond its core functions of user registration, product browsing, secure payments, and wish listing, it seeks to foster a sense of community by encouraging social sharing and interaction. This platform goes the extra mile with features like personalized gift recommendations, underpinned by user data and browsing history. It ensures transparency through order history and tracking, giving users complete control and insight into their purchases. Customer feedback is valued through a review and rating system, allowing shoppers to make informed decisions. Exclusive deals and discounts are tailored to individual preferences, adding an element of personalization to the shopping journey.

The project is designed for today's dynamic world, prioritizing mobile app compatibility, robust security, and multi-lingual support to cater to a diverse global audience. With data analytics at its core, CraftedEuphoria aims to continually evolve, staying ahead of trends and user behavior. Whether through user education or facilitating third-party vendor integration, it aspires to be more than just a shopping platform; it's a dynamic, engaging, and evolving community for users and administrators alike. In doing so, CraftedEuphoria sets out to revolutionize the online gift shopping experience, bringing together convenience, innovation, and personalization.

1.2 PROJECT SPECIFICATION

CraftedEuphoria is an online gift store designed to offer a personalized and user-friendly platform for users to explore, select, and purchase gifts for various occasions. The project comprises several modules, each with its unique features and functionalities, aimed at enhancing the user experience, administrative capabilities, and community engagement.

- **User Module:**

- Users can register by providing their full name, email address, and a secure password.
- Users can browse through a diverse catalog of gift products categorized by occasions, types, and themes.
- Users can access a detailed order history, including order numbers, dates, and order statuses.
- Users can leave detailed product reviews and rate products on a scale.

- The system tracks user preferences and past purchases to offer exclusive deals.
- **Admin Module:**
 - A product approval system can be implemented for quality control.
 - View, process, and manage incoming orders through an intuitive dashboard.
 - Order status updates, such as "Pending," "Processing," "Shipped," and "Delivered," can be managed easily.
 - Configure and manage payment gateways, including settings for transaction
- **Community Module:**
 - Admins can approve or reject reseller applications and manage their product listings.
 - Users can access a budget planner tool to set budgets for different occasions, such as birthdays or holidays.
 - Users can create their own product collections within the community.
 - Communities can personalize their profiles by adding a profile picture, bio, location, and optional details.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

A system study is a critical phase in the lifecycle of any software or organizational project. It is a systematic and structured process of analysing, understanding, and evaluating the existing systems, processes, and technologies within an organization. The primary objective of a system study is to identify inefficiencies, shortcomings, and areas for improvement. This process serves as the foundation for making informed decisions about implementing new systems, software, or process enhancements.

System studies involve a detailed examination of the organization's current operations, which can range from business processes to information systems and technologies in use. The goal is to gain a comprehensive understanding of how the organization operates and how it can be optimized.

2.2 EXISTING SYSTEM

It relies on a website-based platform where users can browse and purchase a variety of gift items for different occasions. Product categories are organized logically, making it relatively easy for users to navigate and explore available options. Users can create accounts and add products to their shopping carts, eventually proceeding to checkout. The system offers a standard payment gateway for online transactions and maintains user order history.

However, the existing system may have limitations in terms of personalization, with limited use of user data for recommendations. Moreover, it may not be optimized for mobile devices, and its inventory management and payment gateway options could be enhanced for a more seamless user experience. Overall, the existing system provides a foundation for the online gift store but presents opportunities for improvement and expansion to better meet user expectations and industry standards.

2.2.1 NATURAL SYSTEM STUDIED

It encompasses the users, administrators, and the broader community as integral components. Users play a central role, both as consumers and content creators, driving engagement and sales. Their preferences and behaviors are crucial in shaping the system's success. Administrators manage the product catalog, orders, and overall system performance, ensuring seamless

operations. The community module, with its potential for third-party vendor integration, adds diversity and enriches the ecosystem.

User-generated content, such as product collections and reviews, enhances the platform's value. The delivery system introduces real-world dynamics, with delivery agents ensuring timely and accurate deliveries while handling returns. This natural system is characterized by its adaptability, responsiveness to user needs, and its potential for growth through community and vendor contributions. Understanding and harmonizing these natural components is essential for the online gift store's continued success and evolution.

2.2.2 DESIGNED SYSTEM STUDIED

It introduces several crucial improvements, including personalized recommendations driven by user data, a user-friendly mobile app for convenient shopping on various devices, and a multi-lingual support system catering to a global audience. The integration of secure payment gateways, offering multiple payment options, enhances user trust and convenience. Moreover, the designed system fosters community engagement by enabling third-party vendors to sell products, broadening the product range, and introducing a gift budget planner for better financial management. User reviews and ratings, order tracking, and exclusive deals provide a comprehensive user experience.

The behavioural feasibility aspects have been carefully considered, ensuring the system aligns with user needs and preferences. Ultimately, the designed system offers an optimized, feature-rich platform that aims to revolutionize the online gift shopping experience, combining innovation and practicality for both users and administrators.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Limited Personalization:** The existing system lacks robust personalization features, such as tailored recommendations based on user preferences and browsing history. This limits the user's ability to discover relevant gift ideas.
- **Price Comparisons:** It might be challenging for customers to compare features and costs across several stores, which could lead to overpaying for an item.
- **Time-consuming:** Visiting several physical stores to evaluate appliances takes time, which may discourage buyers and lead them to make a less-than-ideal decision.
- **Geographical restrictions:** Customers may have fewer options due to limited access to brick-and-mortar stores in rural or less populated locations.
- **Inconvenient Returns and Exchanges:** Compared to online platforms with simple return procedures, returning or exchanging appliances purchased from a physical store might be more difficult and time-consuming.

2.4 PROPOSED SYSTEM

Represents a comprehensive and user-centric platform that aims to transform the online gift shopping experience. It encompasses several key enhancements, beginning with the user module, which includes features for customer support, a gift finder quiz, and a review and rating system for product feedback. The administrative module introduces advanced order tracking, personalized exclusive deals, and discount mechanisms based on user behaviour and preferences. The community module expands to include third-party vendor integration, broadening the product range and incorporating a gift budget planner.

Additionally, a delivery boy module ensures timely and accurate deliveries, confirms order statuses, and handles returns efficiently. The entire system places a strong emphasis on personalization, user engagement, and user-friendliness, from tailored product recommendations and mobile responsiveness to improved payment gateways and user education materials.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Expanded Product Availability:** The proposed system can provide a broader range of home appliance options compared to traditional offline stores. It can connect customers with various appliance suppliers, including local and international sources, increasing the availability of different appliance models. This expanded product availability gives customers more choices and opportunities to find specific appliances they are looking for.
- **Detailed Product Information:** The proposed system can provide detailed product information for each appliance. Customers can access comprehensive descriptions, including details about the appliance's specifications, features, dimensions, and potential uses. This information allows customers to make informed decisions and select appliances that align with their specific needs and preferences.
- **User Reviews and Ratings:** The proposed system can incorporate user reviews and ratings for appliances. Customers can read feedback and experiences shared by other buyers who have previously purchased and used the appliances. User reviews can provide valuable insights into the quality, performance, and durability of the appliances, helping customers make more informed purchasing decisions.
- **Efficient Search Options:** The proposed system can include advanced search and filtering options to facilitate appliance selection. Customers can use specific criteria such as appliance type, size, energy efficiency, or preferred features to narrow down their options and find the desired appliances more efficiently. This saves time and helps customers find appliances that meet their specific requirements.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software.

Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance is considered during the feasibility study. The results of the feasibility study should be a report that recommends whether it is worth carrying on with the requirements engineering and system development process.

The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards.

Various other objectives of feasibility study are listed below.

- To analyse whether the software will meet organizational requirements.
- To determine whether the software can be implemented using the current technology and within the specified budget and schedule,
- To determine whether the software can be integrated with other existing software.

3.1.1 Economic Feasibility

Economic feasibility refers to the ability of a project or business venture to generate enough revenue to cover its costs and provide a reasonable return on investment. It involves analyzing the costs and benefits of a project, including the costs of materials, labor, and equipment, as well as the projected revenue from sales or other sources of income. Economic feasibility is an important consideration when determining whether a project or venture should be undertaken, and it is often used in conjunction with other types of feasibility analysis, such as technical feasibility and operational feasibility.

An extensive analysis of the project's financial viability is necessary to determine its economic viability. This comprises making an initial expenditure estimate for the creation of a website, the integration of a payment gateway, and marketing initiatives. To assess the project's viability, it is crucial to predict the running costs, which include maintenance, server hosting, and customer support. Assessing possible revenue streams, such as product sales, advertising, and fees from outside vendors, can be done by performing a cost-benefit analysis. Finding

development possibilities and potential difficulties will be made easier with the aid of competitor and market demand analysis.

3.1.2 Technical Feasibility

A Technical Feasibility Study is an essential assessment conducted to evaluate the practicality and viability of a proposed project from a technological standpoint. It aims to determine whether the project's implementation is technically feasible and can be executed using available technology, resources, and expertise. The study involves a comprehensive analysis of the project's technical requirements, including infrastructure, software, hardware, programming languages, and integration capabilities. Ultimately, the Technical Feasibility Study serves as a foundation for effective project planning and ensures that the proposed initiative can be realistically implemented using available technical resources and expertise.

Personalized Online Gift Store project's technical feasibility points cover several important factors. First, the front-end and back-end development of the website should make use of the right programming languages and frameworks. Given the utilization of cloud hosting services and load balancing techniques, scalability is crucial to support possible expansion in user traffic and data. For a seamless user experience across many devices, mobile responsiveness is essential. Database management should reliably and securely handle user data, order information, and product information. Integrating a secure payment channel is essential for safeguarding sensitive user data. For seamless and safe transactions, payment gateway integration should include dependable alternatives supporting multiple payment methods.

3.1.3 Behavioral Feasibility

Behavioral feasibility refers to the analysis and assessment of whether the proposed project or system is acceptable and practical from the perspective of its intended users and stakeholders. It focuses on understanding and predicting how individuals and groups will respond to and interact with the project, considering their attitudes, preferences, behavior, and willingness to adopt and use the system. Behavioral feasibility aims to identify potential barriers, resistance, or challenges that might arise during the implementation and operation of the project, and it seeks to ensure that the project aligns with the needs and expectations of its target audience.

Examining the platform's acceptance and usability by its target users is one of the behavioral feasibility criteria in personalized online gift shop. To learn more about what customers want, need, and anticipate from an online gift shop, you can conduct user surveys or focus groups. The platform's functionality and design can be tailored to the preferences of users by considering their behavior and adoption trends. To make sure that users have a seamless and satisfying experience, user training and onboarding resources may be required. You may increase your online gift shop's chances of success and customer happiness by assessing the behavioral factors and making sure it is user-friendly, interesting, and caters to the needs of its target markets.

3.1.4 Feasibility Study Questionnaire

1. Project Overview?

CraftedEuphoria the personalized gift store is an interactive and user-friendly platform where users can easily explore, select, and purchase a diverse range of gifts for various occasions. The website's layout features a clean and attractive design, with a prominent search bar on the homepage for quick and efficient gift searches. Users can navigate through different categories and product listings, view detailed descriptions and images, and add items to their carts seamlessly. The platform also offers personalized recommendations based on user preferences and past purchases, enhancing the shopping experience. Secure payment gateways ensure safe online transactions, and an efficient order management system allows admin to handle incoming orders effectively. Additionally, the inclusion of a community module encourages users to share and discover unique gift ideas, fostering a sense of engagement and collaboration among community members. Overall, the project view reflects a well-curated and user-centric online gift store that caters to the diverse needs of its audience and provides a delightful shopping experience.

2. To what extend the system is proposed for?

The proposed system is intended to provide a comprehensive and user-friendly platform for users to explore, select, and purchase gifts for various occasions. It includes features such as personalized recommendations, secure payment transactions, and community engagement to enhance user satisfaction and create a seamless shopping experience.

3. Specify the Viewers/Public which is to be involved in the System?

General Viewers, Registered Users

4. List the Modules included in your System?

User, Admin, Community and Delivery Boy

5. Who owns the system?

Admin

6. System is related to which firm/industry/organization?

Ladies Centre Industry

7. Details of person that you have contacted for data collection?

Shaji Jose (Kizhakkeparambil Pearl Store, Erumely)

Questionnaire to collect details about the project?**1. What types of products do you offer in your gift store?**

Greeting cards and stationery, Decorative items, Personalized gifts, Jewelry, and accessories, Candles and scented products, Gift baskets and hampers, Artwork, and prints, Seasonal gifts for holidays and special occasions.

2. How do you manage and organize your inventory?

Stores keep safety stock to handle unexpected demand fluctuations.

3. What is your process for restocking items?

The newly restocked items are placed on display or stored appropriately for easy access and visibility to customers.

4.How do you handle gift wrapping services, if available?

If a customer purchases multiple gifts, the store may offer to consolidate them into one beautifully wrapped package.

5. What are the popular gift items that customers often purchase?

Greeting Cards, Personalized Gifts, Accessories, Gift Baskets

6. How do you handle customer returns and exchanges?

Customers may be given a limited window of time, typically lasting a few days to several weeks, in which to return or exchange things. Customers may be given a limited window of time, typically lasting a few days to several weeks, in which to return or exchange things.

7. What payment methods do you accept in your store?

Cash, Google pay

8. How do you promote and advertise your gift store?

Advertise in local newspapers, magazines to reach the local audience.

9. How do you handle seasonal or holiday sales and promotions?

Hosting special events, such meetings with Santa, craft sessions, or activities with a holiday theme, can draw families and customers.

10. How do you handle exchanges for items with different sizes, colors, or variations?

Customers are usually required to present a valid receipt or proof of purchase for the item

they wish to exchange to verify that the product was purchased from the store.

3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz

RAM - 8.00 GB

SSD - 512 GB

3.2.2 Software Specification

Front End - HTML, CSS

Back End - Python, Django

Database - SQLite

Client on PC - Windows 7 and above.

Technologies used - JS, HTML5, AJAX, J Query, PHP, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 Python

Python is a versatile and widely used programming language, prized for its readability and ease of use. Its simple syntax and extensive library support, including frameworks like Django and Flask, expedite development. Python finds applications across web development, data analysis, machine learning, and more. Its platform independence and scalability make it suitable for various operating systems and evolving project needs. In the proposed system, Python forms the foundation for backend development, core functionality, data processing, and integration with different tools and frameworks, enabling the creation of an efficient online platform for purchasing gifts.

3.3.2 Django

Django is a high-level Python web framework known for its simplicity and efficiency in web application development. It follows the "batteries-included" philosophy, offering a wide range of

built-in features and tools that empower developers to create robust, scalable, and secure web applications. Django promotes the DRY (Don't Repeat Yourself) principle, making it easy to build complex applications with clean, maintainable code. It includes an Object-Relational Mapping (ORM) system for database interactions, a templating engine for designing user interfaces, and a secure authentication system. Django's built-in admin interface simplifies content management and data handling. With a strong community, extensive documentation, and a rich ecosystem of third-party packages, Django remains a top choice for web developers aiming to streamline the development process and deliver high-quality web applications.

3.3.3 SQLite

SQLite is a lightweight and self-contained relational database management system. It's known for its simplicity, speed, and ease of integration, making it a popular choice for embedded systems and applications. SQLite operates without a separate server process and allows direct access to the database using a simple and efficient query language. It's ideal for small to medium-sized applications, especially those that need a local data store. In the proposed system, SQLite serves as the backend database, efficiently managing and storing essential data related to home appliances, user accounts, transactions, and more, ensuring a seamless and reliable user experience.

- **Self-Contained:** SQLite is a self-contained DBMS, meaning it doesn't require a separate server process. The entire database is stored in a single file on the disk.
- **Zero Configuration:** Unlike many other database systems, SQLite requires minimal to no configuration. You can start using it by just including the library in your project.
- **Serverless Architecture:** It operates without a central server, allowing applications to access the database directly, simplifying setup and reducing latency.
- **ACID Compliant:** SQLite ensures data reliability through ACID (Atomicity, Consistency, Isolation, Durability) compliance, guaranteeing that transactions are processed reliably.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System design is a critical phase in the software development process, serving as the bridge between high-level requirements and actual implementation. It involves the detailed planning, structuring, and specification of a software system's architecture, components, and functionalities. The primary goal of system design is to transform the abstract concepts and ideas gathered during the requirements analysis phase into a concrete, well-defined blueprint for building the software.

Effective system design is crucial for the successful development and deployment of software systems. It minimizes the risk of misunderstandings, errors, and costly revisions by providing a clear and comprehensive plan that guides the development team throughout the implementation phase.

4.2 UML DIAGRAM

A UML (Unified Modeling Language) diagram is a graphical representation used in software engineering and other fields to visualize, model, and communicate various aspects of a system, such as its structure, behavior, and interactions. UML diagrams provide a standardized and universally accepted way to convey complex system designs. There are several types of UML diagrams, each serving a specific purpose.

- Class Diagram
- Object Diagram
- Use case Diagram
- Sequence Diagram
- Collaboration Diagram
- Activity Diagram
- State Chart Diagram
- Deployment Diagram
- Component Diagram

4.2.1 USE CASE DIAGRAM

A use case diagram is a type of Unified Modelling Language (UML) diagram that is used to visually represent the functional requirements of a system or software application from the perspective of its users. It provides a high-level view of how a system interacts with its various actors (users, other systems, or external entities) to accomplish specific tasks or functions. Use case diagrams are often employed in the early stages of software development to help define and understand the system's behaviour and its interactions with external elements. Here are some key components and concepts associated with use case diagrams:

- **Actors:** Actors are external entities that interact with the system or software. They can be individuals, other systems, or even hardware devices. Actors are depicted as stick figures or other simple shapes. They are not part of the system but have specific roles or responsibilities in the system's functionality.
- **Use Cases:** Use cases represent specific tasks or functions that the system performs to achieve a particular goal or objective. They describe the system's behaviour in response to a user's or actor's interaction. Use cases are typically represented as ovals. Each use case has a name that describes the specific functionality, such as "Create Account" or "Place Order."
- **Relationships:**
 - Association: A solid line connecting an actor to a use case indicates that the actor interacts with that use case.
 - Inclusion: A dashed line with an arrow pointing to an included use case indicates that one use case includes or invokes another use case. This helps in avoiding redundancy by reusing common functionality.
 - Extension: A dashed line with an arrow pointing to an extended use case indicates that one use case can be extended by another use case under certain conditions. This is used to show optional or alternative behaviour.
- **System Boundary:** The use case diagram typically includes a system boundary, represented as a rectangle that encloses all the use cases and actors. It defines the scope of

the system under consideration.

- **Multiplicity Notation:** Sometimes, you can use multiplicity notation to show how many instances of an actor or use case are involved in a particular interaction.

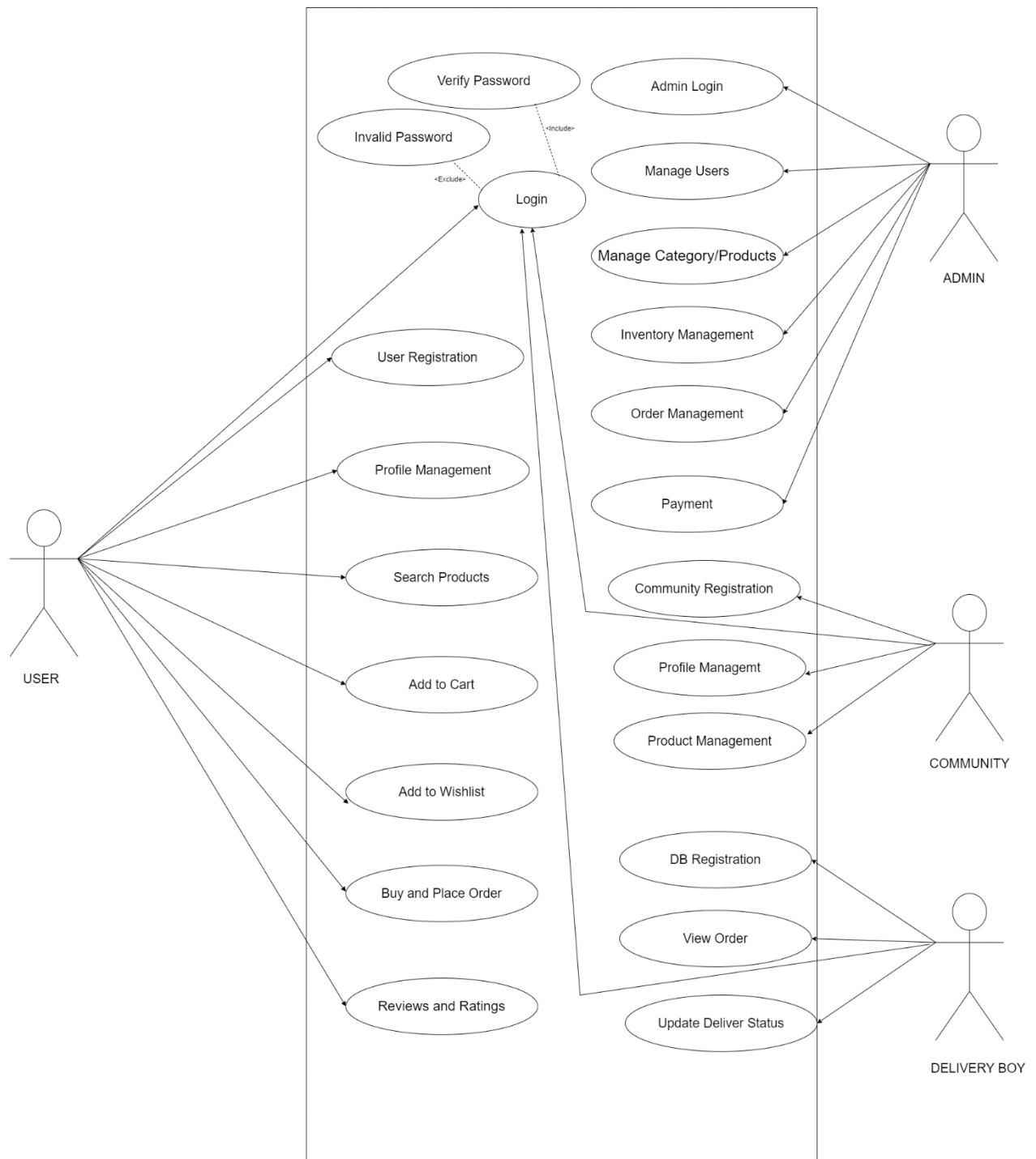


Fig 1: Use case diagram for Online Gift Store

4.2.1 SEQUENCE DIAGRAM

A sequence diagram is a type of Unified Modelling Language (UML) diagram that illustrates the interactions and order of messages between objects or components in a system, showing how they collaborate and work together over time. Sequence diagrams are particularly useful for visualizing and understanding the dynamic behaviour of a system, especially during the execution of specific use cases or scenarios. Here are the key elements and concepts associated with sequence diagrams:

- **Lifelines:** Lifelines represent objects or components participating in the sequence of interactions. Each lifeline is depicted as a vertical dashed line. It typically corresponds to a class or instance of a class in object-oriented design. The lifeline's name or label indicates the object or component it represents.
- **Messages:** Messages are arrows that indicate the flow of communication or interaction between lifelines. They show the order and direction of the interactions. There are two main types of messages:
 - **Synchronous Messages:** These are shown as solid arrows and represent a direct, synchronous call between objects. The sender waits for a response from the receiver before continuing.
 - **Asynchronous Messages:** These are shown as dashed arrows and represent an asynchronous call where the sender does not wait for a response from the receiver. It indicates that the sender sends a message and continues its work without waiting.
- **Activation Bars:** Activation bars represent the period during which an object or component is actively processing a message or is in focus. They are shown as horizontal lines extending from the lifelines, indicating when an object is busy or "activated" by a particular message.
- **Return Messages:** Return messages are used to represent the return or response from a method call. They connect the activation bar of the called object to the activation bar of the calling object.
- **Found Messages:** These messages represent the creation of an object or its appearance in

the system. They are often used to illustrate the creation of new instances.

- **Notes:** Notes or comments can be added to provide additional information or explanations about the interactions or specific elements in the diagram.

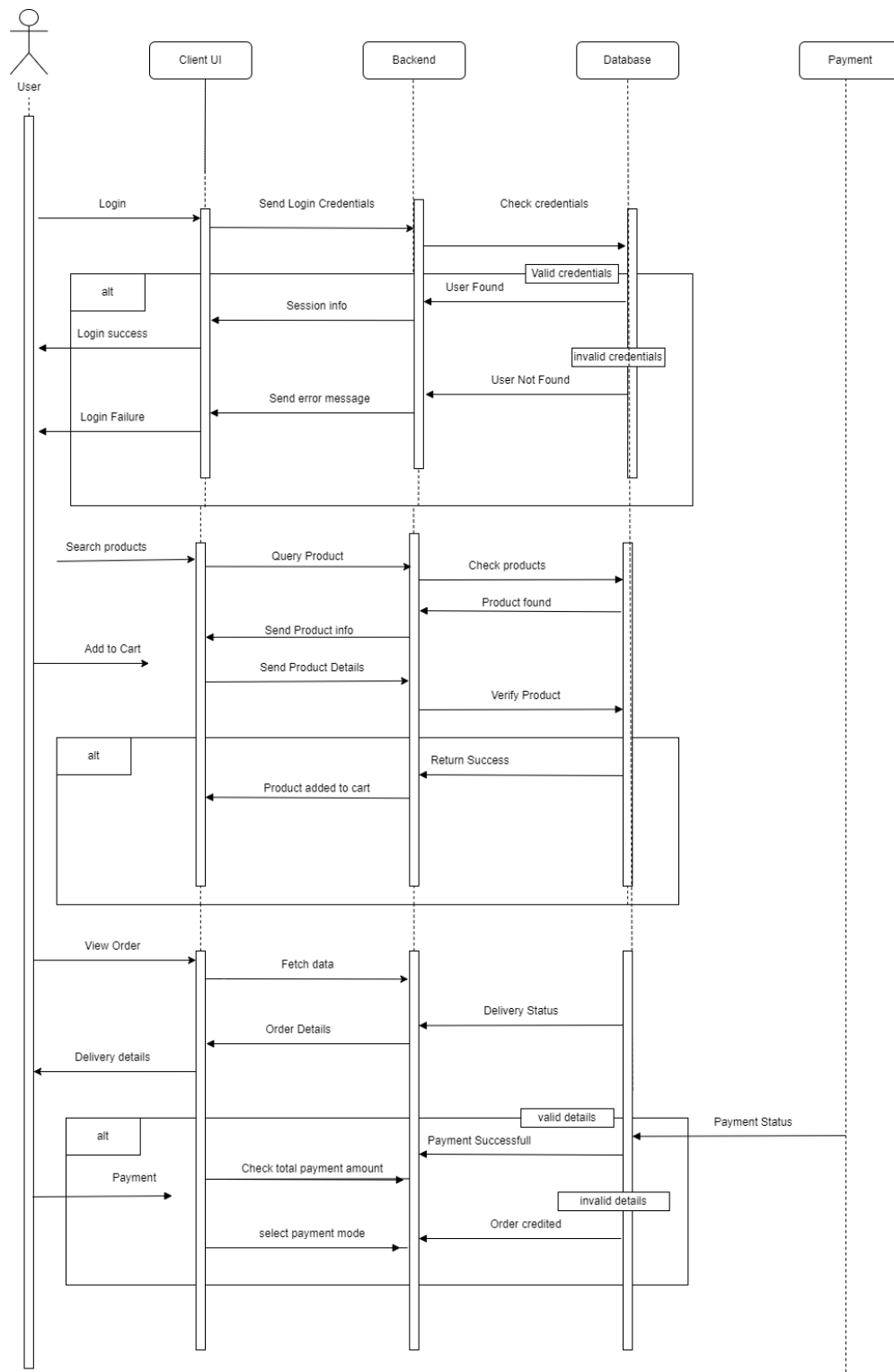


Fig 2: Sequence diagram for Online Gift Store

4.2.2 State Chart Diagram

A state chart diagram, also known as a state machine diagram, is a type of Unified Modeling Language (UML) diagram used to depict the various states that an object or system can be in and the transitions between those states. It is particularly useful for modelling the dynamic behaviour of a system over time and in response to specific events or conditions. Here are the key components and concepts associated with state chart diagrams:

- **States:** A state represents a specific condition or situation that an object or system can be in. States are typically depicted as rectangles with rounded corners and labelled with a name, such as "Idle," "Active," or "Error."
- **Transitions:** Transitions are arrows that show how an object or system moves from one state to another in response to an event, condition, or action. Transitions can be labeled with the triggering event or condition that causes the transition, and they often include guard conditions that must be satisfied for the transition to occur.
- **Events:** Events are occurrences or stimuli that trigger transitions between states. These can include user actions, system events, or the passage of time. Events are represented as ovals and labeled with names like "Start," "Stop," or "Button Click."
- **Initial State:** An initial state (usually represented by a filled black circle) shows the state from which the object or system begins its life cycle when it is first created or initialized.
- **Final State:** A final state (usually represented by a circle with a dot inside) indicates the termination or completion of the object's or system's life cycle.
- **Concurrent States:** State chart diagrams can depict concurrent states, where multiple states can be active at the same time. This is useful for modeling systems that have parallel processes or activities.

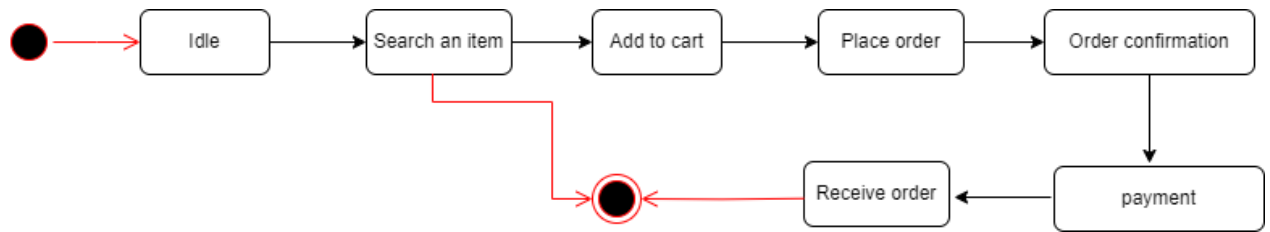


Fig 3: State Chart diagram for Online Gift Store

4.2.2 Activity Diagram

An activity diagram is a type of Unified Modeling Language (UML) diagram that visualizes the workflow or business processes within a system or a specific use case. It is particularly useful for representing the sequential and parallel activities, actions, decisions, and control flows that make up a complex process or algorithm. Here are the key components and concepts associated with activity diagrams:

- **Activity:** An activity is a specific task or operation within the system. It is represented as a rounded rectangle with a label, such as "Process Order" or "Validate User."
- **Initial Node:** The initial node is a small solid circle that indicates the starting point of the activity diagram. It shows where the process begins.
- **Final Node:** The final node is a solid circle with a dot inside that indicates the endpoint or completion of the process.
- **Action:** An action represents a specific operation or task that is performed within the workflow. Actions are typically depicted as rectangles with rounded corners.
- **Control Flow:** Control flows are arrows that show the sequence of activities or actions within the diagram. They connect the activities and indicate the order in which they are performed.
- **Decision Node:** A decision node is a diamond-shaped symbol used to represent a point in the process where a decision or choice must be made. It typically has multiple outgoing control flows, each associated with a different condition.

- **Merge Node:** A merge node is used to bring multiple control flows back together into a single flow after a decision or branching point.
- **Fork Node:** A fork node, represented as a black bar, is used to indicate that multiple actions or activities can be performed in parallel.
- **Join Node:** A join node, represented as a white bar, is used to show where parallel flows converge into a single flow.

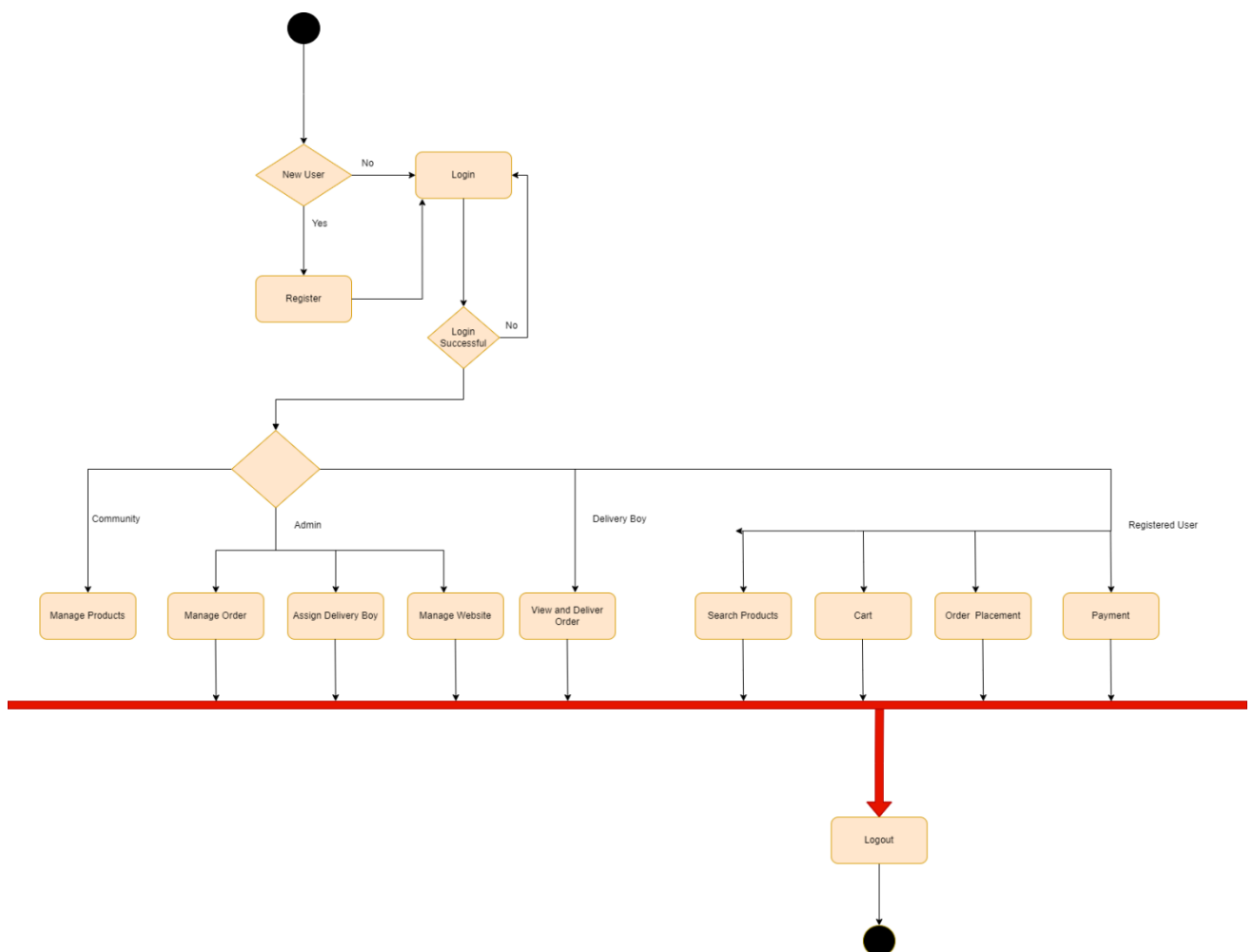


Fig 4: Activity diagram for Online Gift Store

4.2.3 Class Diagram

A class diagram is a type of Unified Modeling Language (UML) diagram used to visualize and represent the static structure of a system or software application. It primarily focuses on the classes, objects, relationships, and attributes that make up the system, offering a clear and organized way to model the system's structure. Here are the key components and concepts associated with class diagrams:

- **Class:** A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have. Classes are typically represented as rectangles with three compartments: the top compartment contains the class name, the middle compartment lists the attributes, and the bottom compartment lists the methods.
- **Object:** An object is an instance of a class. It represents a specific real-world entity or concept within the system. Objects are typically not shown in class diagrams but are derived from classes at runtime.
- **Attributes:** Attributes are the characteristics or properties of a class. They describe the data that an object of the class will hold. Attributes are typically listed in the middle compartment of the class rectangle and may include data types and visibility (e.g., public, private, protected).
- **Methods:** Methods represent the behaviors or operations that an object of the class can perform. They are typically listed in the bottom compartment of the class rectangle and include the method name, parameters, return type, and visibility.
- **Associations:** Associations represent the relationships between classes. They indicate how classes are related or connected. Associations can have names, multiplicities (e.g., one-to-one, one-to-many), and roles to specify the nature of the relationship.
- **Aggregation and Composition:** These are special types of associations that represent part-whole relationships. Aggregation represents a "whole-part" relationship, while composition indicates a stronger "exclusive ownership" relationship.

- **Generalization/Inheritance:** Generalization represents the inheritance relationship between classes. It indicates that one class (the subclass or child class) inherits attributes and behaviors from another class (the superclass or parent class).
- **Interfaces:** Interfaces define a contract for a group of classes. They specify a set of methods that classes implementing the interface must provide. Interfaces are represented as a dashed line with a triangle at one end.
- **Dependency:** Dependency is a weaker relationship between classes. It indicates that one class uses or depends on another class without a structural relationship.

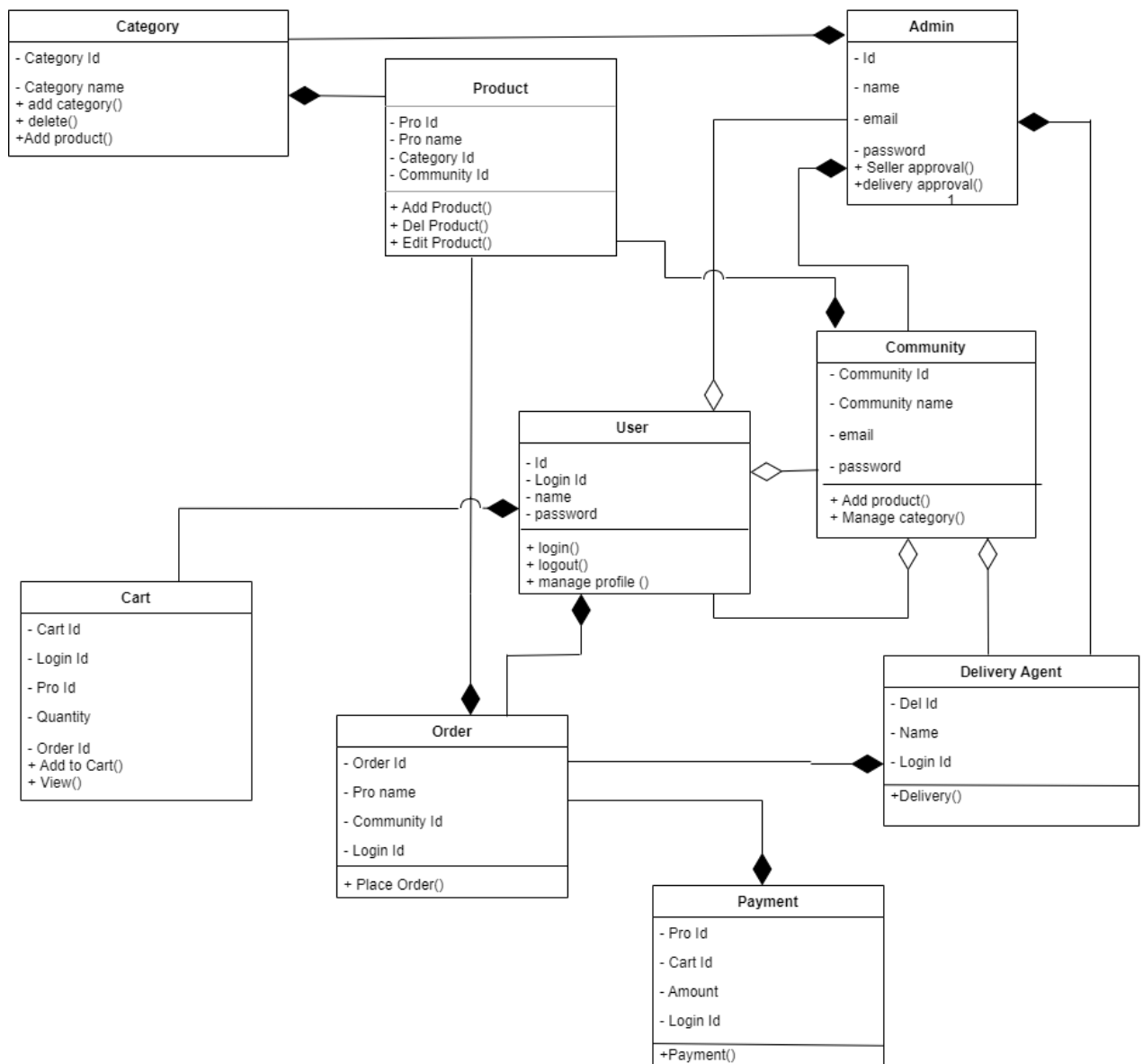


Fig 5: Class diagram for Online Gift Store

4.2.4 Object Diagram

An object diagram is a type of Unified Modeling Language (UML) diagram that provides a snapshot of a specific system or a part of it at a particular moment in time. It visualizes the instances (objects) of classes and the relationships between them, offering a detailed view of the system's structure and the state of objects at a specific point in its execution. Here are the key components and concepts associated with object diagrams:

- **Object:** An object represents an instance of a class at a specific moment in time. It carries the values of the class's attributes and is depicted as a rectangle with the object name followed by a colon and the class name (e.g., "myObject: MyClass"). The object may have underlined attributes to show their values.
- **Class:** A class is a blueprint for creating objects. Objects in an object diagram are instances of classes.
- **Attributes:** Attributes represent the properties or data members of a class. They define the characteristics or state of objects.
- **Relationships:** Object diagrams can depict the relationships between objects, such as associations, aggregations, compositions, or dependencies. These relationships illustrate how objects are related to each other.
- **Links:** Links are lines connecting objects in the diagram to represent relationships between them. These links show how instances of different classes are connected.

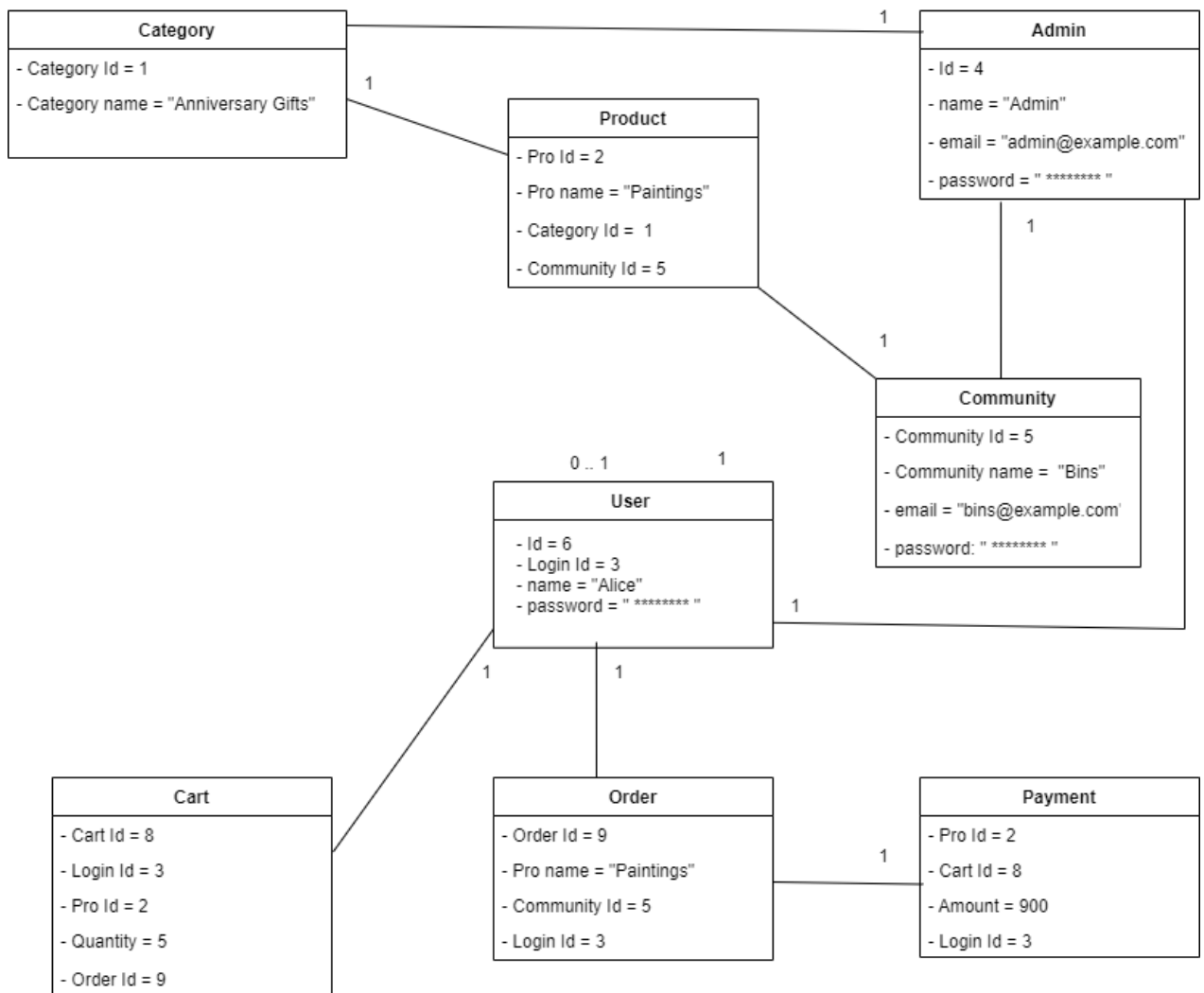


Fig 6: Object Diagram for Online Gift Store

4.2.5 Component Diagram

A component diagram is a type of Unified Modeling Language (UML) diagram used to depict the physical components and their relationships in a system or software application. It focuses on the high-level structure of the system, illustrating how different components interact and collaborate to provide the system's functionality. Here are the key components and concepts associated with component diagrams:

- Component:** A component is a modular, reusable, and replaceable part of a system or software application. Components can be software modules, libraries, executable files, hardware devices, or any other physical or logical unit that contributes to the system's functionality. Components are represented as rectangles with the component name inside.

- **Interface:** Interfaces define a contract specifying a set of methods or services that a component exposes to the outside world. They allow components to interact with one another through well-defined communication points. Interfaces are depicted as small rectangles with the interface name and a lollipop-like connector.
- **Port:** Ports are connection points on a component where interfaces can be attached. They represent how a component communicates with the external environment or with other components. Ports are typically small squares located on the sides of a component and are connected to interfaces.
- **Dependency:** A dependency is a relationship between two components that indicates that one component relies on the functionality provided by another component. Dependencies are often represented as dashed arrows.
- **Assembly Connector:** Assembly connectors illustrate the relationships between components when one component contains or uses another component. These connectors show how components are combined to form a larger system. Assembly connectors are depicted as solid arrows.
- **Provided and Required Interfaces:** Provided interfaces indicate the interfaces offered by a component, while required interfaces specify the interfaces that the component depends on or requires to function properly.

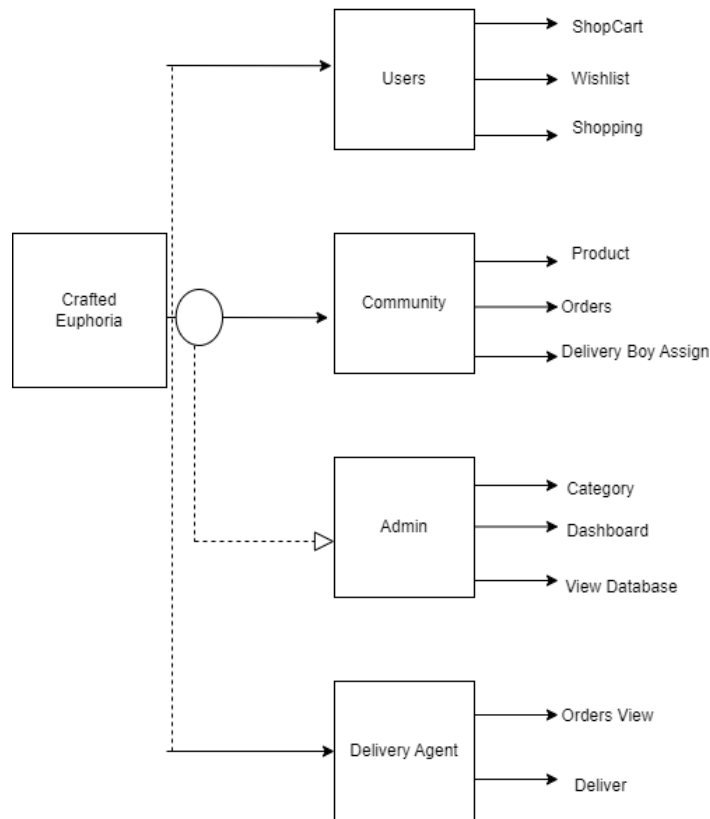


Fig 7: Component Diagram for Online Gift Store

4.2.8 Deployment Diagram

A deployment diagram is a type of Unified Modeling Language (UML) diagram used to visualize the physical deployment of software components and hardware nodes in a system. It provides a clear and detailed representation of how software components are distributed across various hardware elements such as servers, devices, and networks. Deployment diagrams are particularly useful for understanding the system's deployment architecture, including how software is hosted and how different components interact with each other. Here are the key components and concepts associated with deployment diagrams:

- **Nodes:** Nodes represent the hardware elements or computing devices in the system. These can include physical servers, virtual machines, workstations, routers, or other hardware components. Nodes are depicted as rectangles with the name of the hardware element or device.
- **Artifacts:** Artifacts are instances of software components that are deployed on nodes. They represent executable files, libraries, configuration files, and other

software elements that are distributed across the hardware. Artifacts are depicted as rectangles or ellipses and are typically connected to the nodes on which they are deployed.

- **Deployed Artifact:** A line with an arrow connecting an artifact to a node represents the deployment of that artifact on the node. It indicates which software components are running on specific hardware.
- **Relationships:** Deployment diagrams can show various relationships between nodes, such as associations, dependencies, and generalization relationships. These relationships can help define how nodes interact with each other.
- **Communication Path:** Communication paths, represented as lines with arrows, illustrate how nodes are connected and communicate with each other through networks or other means.
- **Stereo Types:** Deployment diagrams can use stereo types to provide additional information about nodes and artifacts. For example, you can use "<<web server>>" to indicate that a node is a web server.

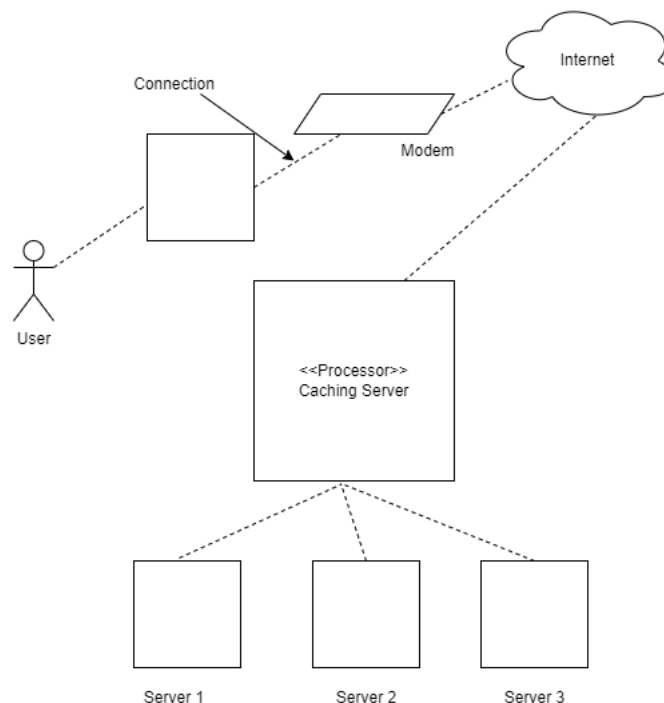


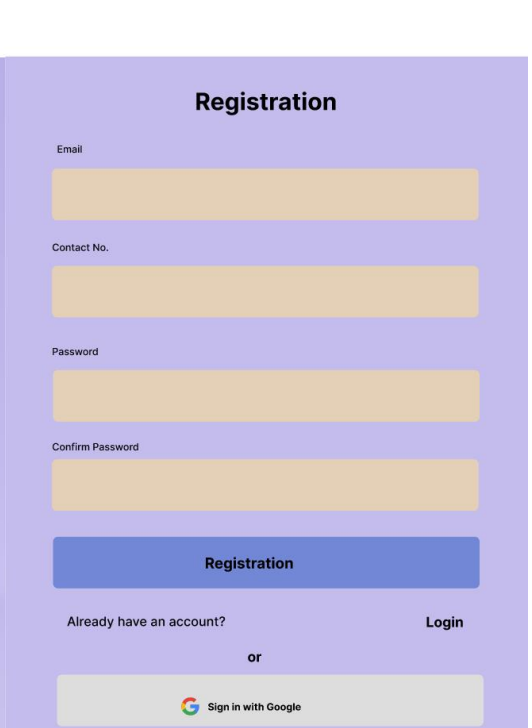
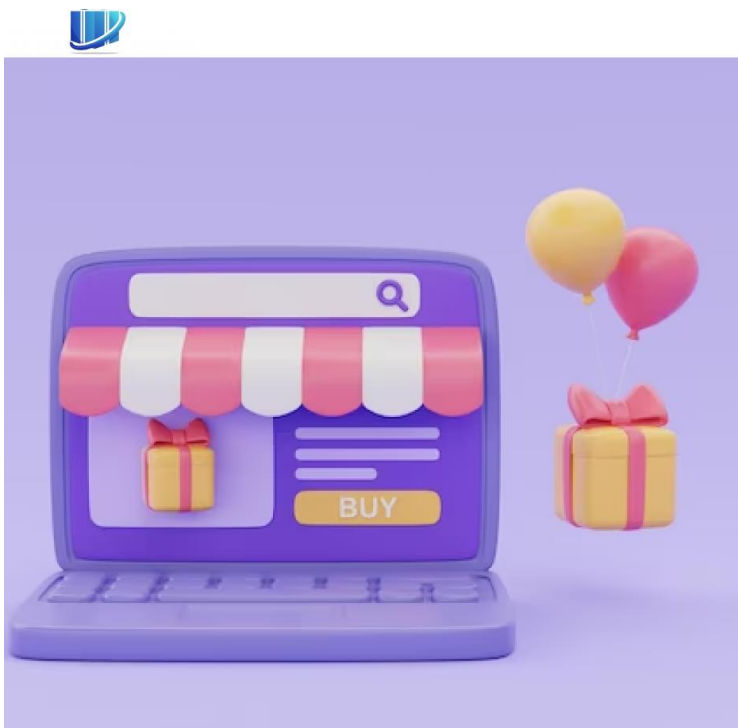
Fig 8: Deployment Diagram for Online Gift Store

4.2.9 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). Developers can use these diagrams to portray the dynamic behaviour of a particular use case and define the role of each object. To create a collaboration diagram, first identify the structural elements required to carry out the functionality of an interaction. Then build a model using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Registration Page



Registration

Email

Contact No.


Password

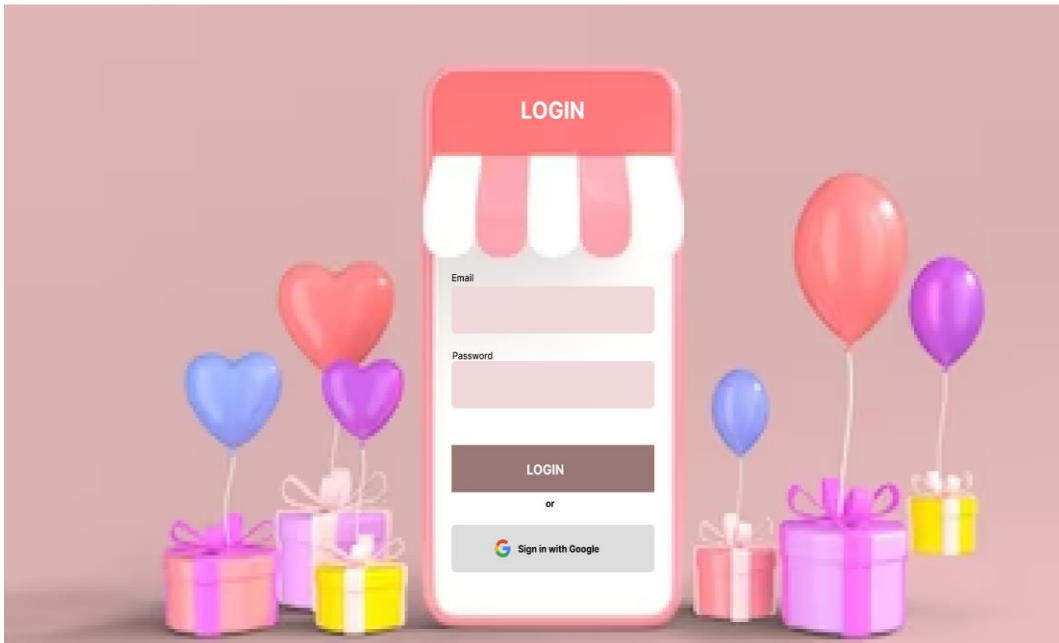
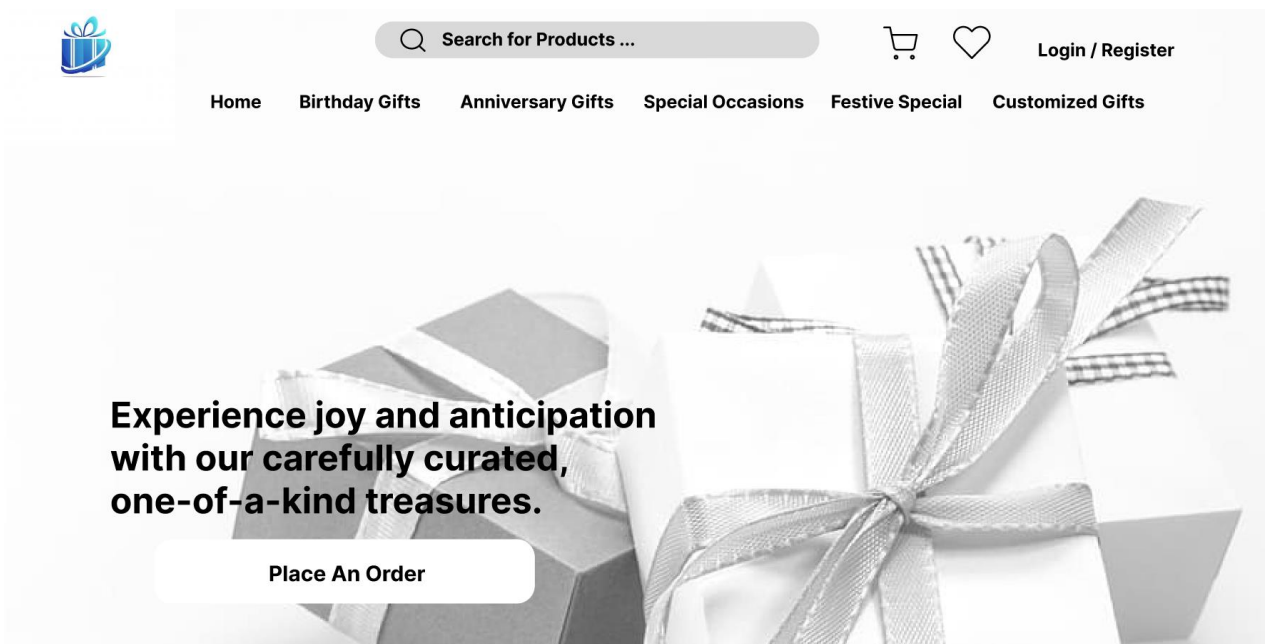
Confirm Password

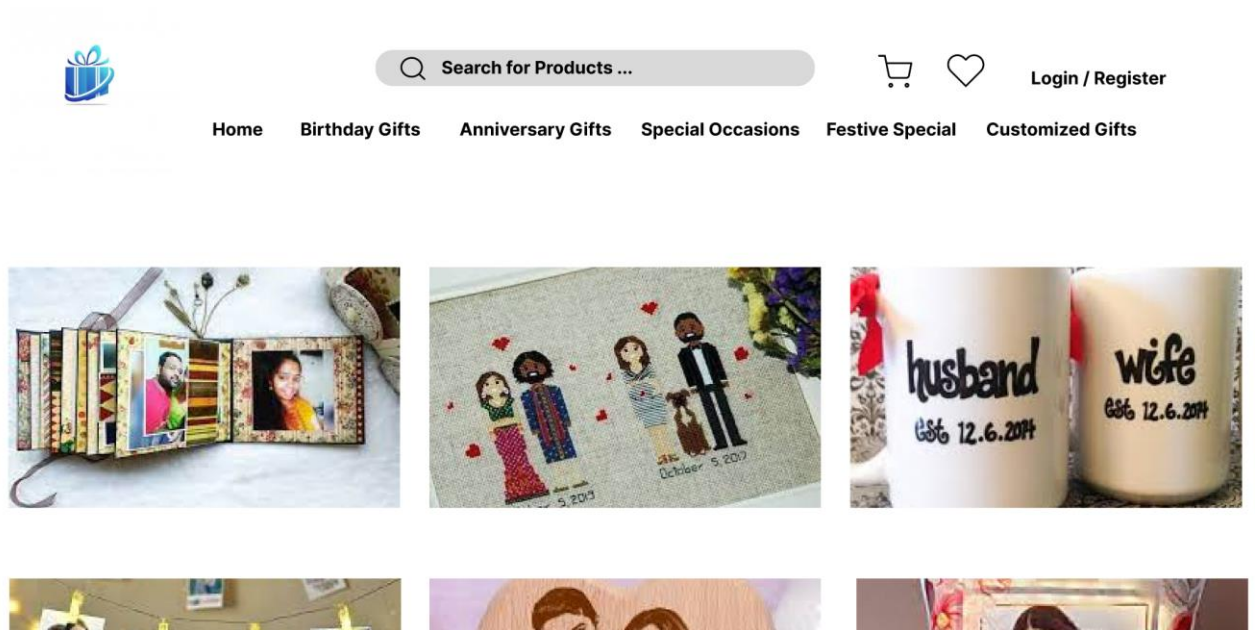
Registration

Already have an account? **Login**

or

 Sign in with Google

Form Name: Login Page**Form Name: Home Page**

Form Name: Product Page

4.4 DATABASE DESIGN

A database is an organized collection of information that's organized to enable easy accessibility, administration, and overhauls. The security of information could be an essential objective of any database. The database design process comprises of two stages. In the first stage, user requirements are gathered to create a database that meets those requirements as clearly as possible. This is known as information-level design and is carried out independently of any DBMS. In the second stage, the design is converted from an information-level design to a specific DBMS design that will be used to construct the system. This stage is known as physical-level design, where the characteristics of the specific DBMS are considered. Alongside system design, there is also database design, which aims to achieve two main goals: data integrity and data independence.

4.4.1 Relational Database Management System (RDBMS)

A Relational Database Management System (RDBMS) is a powerful tool for organizing and managing structured data. In the RDBMS model, data is stored in tables with rows and columns, and these tables can be related to one another through primary and foreign keys, enabling data integrity and efficient retrieval. RDBMS systems follow the ACID properties to ensure data consistency and reliability, even in the event of system failures. They use SQL as the standard language for data manipulation and querying.

RDBMS systems support features like indexes for optimizing query performance, normalization for data organization, and the ability to define triggers, stored procedures, and views for custom data handling and business logic. These systems find extensive use in applications where data integrity and structured organization are paramount, such as enterprise databases, e-commerce platforms, and content management systems, with popular RDBMS products like Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and SQLite providing robust solutions.

4.4.2 Normalization

Normalization is a database design process used to minimize data redundancy and improve data integrity in relational database management systems (RDBMS). It involves organizing data into structured tables to reduce the chances of anomalies during data manipulation and retrieval. The primary goals of normalization are to:

1. **Eliminate Data Redundancy:** By organizing data efficiently, normalization reduces duplication of data within a database. This minimizes storage requirements and ensures that data remains consistent.
2. **Ensure Data Integrity:** Normalization enforces rules and constraints that maintain data integrity. It helps prevent anomalies such as update anomalies, insertion anomalies, and deletion anomalies, which can occur when data is not properly structured.

Normalization typically involves breaking large tables into smaller, related tables, ensuring that each table has a clear, well-defined purpose. It is achieved through a series of normal forms (e.g., First Normal Form, Second Normal Form, Third Normal Form) that progressively improve the organization and structure of the database. The specific normal forms address issues like partial dependencies, transitive dependencies, and other data integrity concerns, ultimately resulting in a well-structured and efficient database schema.

4.4.3 Sanitization

Data sanitization is the process of removing any illegal characters or values from data. In web applications, sanitizing user input is a common task to prevent security vulnerabilities. PHP provides a built-in filter extension that can be used to sanitize and validate various types of external input such as email addresses, URLs, IP addresses, and more. These filters are designed to make data sanitization easier and faster. For example, the PHP filter extension has a function that can remove all characters except letters, digits, and certain special characters (!#\$%&'*+ -=?_`{}~@.[]), as specified by a flag.

Web applications often receive external input from various sources, including user input from forms, cookies, web services data, server variables, and database query results. It is important to sanitize all external input to ensure that it is safe and does not contain any malicious code or values.

4.4.4 Indexing

Indexing, in the context of databases and data storage, is the process of creating a data structure (an index) to optimize the retrieval of specific data from a larger dataset. Indexes are used to improve the speed and efficiency of data retrieval operations by providing a quick lookup mechanism for specific columns or fields in a database table. They work like an organized

reference to data, allowing the database management system (DBMS) to quickly locate the rows that match a given search criteria without the need to scan the entire dataset. Here are some key points about indexing:

1. **Data Retrieval Optimization:** The primary purpose of indexing is to speed up data retrieval operations, such as searching for records that meet certain criteria or filtering data based on specific columns. Without indexes, these operations might require scanning the entire dataset, which can be slow and resource-intensive for large datasets.
2. **Indexed Columns:** Indexes are created on specific columns of a database table. Typically, these columns are the ones frequently used in search conditions (e.g., in WHERE clauses of SQL queries). Indexes are not created on all columns, only on those where the benefit of faster data retrieval justifies the additional storage and maintenance overhead.
3. **Data Structure:** An index is a data structure that contains a sorted or hashed copy of a subset of the data from the indexed column, along with a pointer to the actual data row. The data in the index is organized to facilitate quick lookups.
4. **Types of Indexes:** There are various types of indexes, including B-tree (balanced tree) indexes, hash indexes, bitmap indexes, and more. Each type has its advantages and is suitable for specific use cases.
5. **Maintenance:** Indexes need to be maintained as data in the table changes. This means that when you add, update, or delete rows, the corresponding changes must also be reflected in the indexes. Therefore, there is a trade-off between the benefits of faster data retrieval and the overhead of index maintenance.
6. **Unique and Non-Unique Indexes:** Some indexes enforce uniqueness, meaning that the indexed column's values must be unique across all rows. Other indexes allow duplicate values in the indexed column.
7. **Clustered and Non-clustered Indexes:** In some database systems, you may come across the concept of clustered and non-clustered indexes. A clustered index determines the physical order of data in a table, while non-clustered indexes are separate data structures used for quicker data retrieval.
8. **Composite Indexes:** In cases where you frequently search based on multiple columns, you can create composite indexes that include multiple columns in a single index.

4.5 TABLE DESIGN

1.Tbl_login

Primary key: **user_id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Login_id	Int(10)	Primary Key	Login id
2.	Email	Varchar(20)	Null	Email
3.	Password	Varchar(20)	Not Null	Password
4.	Reg_id	Int(10)	Foreign Key	Registration id

2.Tbl_user_reg

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	User_id	Int(10)	Primary Key	User id
2.	Name	Varchar(20)	Not Null	Name
3.	Contact_no	Varchar(20)	Not Null	Contact_no
4.	Role_id 1	Int(10)	Not Null	Registration
5.	Role_id 2	Int(10)	Not Null	Registration
6.	Addressline_1	Varchar(20)	Not Null	Address
7.	Status	Int(5)	Not Null	Status

3.Tbl_category

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Category_id	Int(10)	Primary Key	Category id
2.	Category_Name	Varchar(20)	Not Null	Category Name
3.	Category_Description	Varchar(20)	Not Null	Category Description
4.	Status	Int(5)	Not Null	Status

4.Tbl_product

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Pro_id	Int(10)	Primary Key	Product id
2.	Pro_Name	Varchar(20)	Not Null	Product Name
3.	Category_id	Int(10)	Foreign Key	Category id
4.	Pro_description	Varchar(20)	Not Null	Product Description
5.	Community_id	Int(10)	Foreign Key	Community id
6.	Quantity	Int(10)	Null	Quantity
7.	Status	Int(5)	Not Null	Status

5.Tbl_order

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Order_id	Int(10)	Primary Key	Order id
2.	User_id	Int(10)	Foreign Key	User id
3.	Pro_id	Int(10)	Foreign Key	Category id
4.	Login_id	Int(10)	Foreign Key	Login id
5.	Quantity	Int(10)	Not Null	Quantity
6.	Addressline_1	Varchar(20)	Not Null	Address

6.Tbl_Community

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Community_id	Int(10)	Primary Key	Category id
2.	Community_Name	Varchar(20)	Not Null	Category Name
3.	Community_Description	Varchar(20)	Not Null	Category Description

7.Tbl_Payment

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1.	Payment_id	Int(10)	Primary Key	Payment id
2.	Order_id	Int(10)	Foreign Key	Order id
3.	Payment_amt	Varchar(20)	Not Null	Payment Amount
4.	Payment_Status	Varchar(20)	Not Null	Status of Payment
5.	Date	Date	Not Null	Date
6.	Community_id	Int(10)	Not Null	Community id

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing involves executing a software program in a controlled manner to determine if it behaves as intended, often using verification and validation methods. Validation involves evaluating a product to ensure it complies with specifications, while verification can involve reviews, analyses, inspections, and walkthroughs. Static analysis examines the software's source code to identify issues, while dynamic analysis examines its behavior during runtime to gather information like execution traces, timing profiles, and test coverage details. Testing involves a series of planned and systematic activities that start with individual modules and progress to the integration of the entire computer-based system.

The objectives of testing include identifying errors and bugs in the software, ensuring that the software functions according to its specifications, and verifying that it meets performance requirements. Testing can be performed to assess correctness, implementation efficiency, and computational complexity. A successful test is one that detects an undiscovered error, and a good test case has a high probability of uncovering such errors. Testing is crucial to achieving system testing objectives and can involve various techniques such as functional testing, performance testing, and security testing.

5.2 TEST PLAN

A test plan is a document that outlines the required steps to complete various testing methodologies. It provides guidance on the activities that need to be performed during testing. Software developers create computer programs, documentation, and associated data structures. They are responsible for testing each component of the program to ensure it meets the intended purpose. To address issues with self-evaluation, an independent test group (ITG) is often established. Testing objectives should be stated in quantifiable language, such as mean time to failure, cost to find and fix defects, remaining defect density or frequency of occurrence, and test work-hours per regression test.

The different levels of testing include:

- Unit testing
- Integration testing
- Data validation testing

- Output testing

5.2.1 Unit Testing

Unit testing is a software testing technique that focuses on verifying individual components or modules of the software design. The purpose of unit testing is to test the smallest unit of software design and ensure that it performs as intended. Unit testing is typically white-box focused, and multiple components can be tested simultaneously. The component-level design description is used as a guide during testing to identify critical control paths and potential faults within the module's perimeter.

During unit testing, the modular interface is tested to ensure that data enters and exits the software unit under test properly. The local data structure is inspected to ensure that data temporarily stored retains its integrity during each step of an algorithm's execution. Boundary conditions are tested to ensure that all statements in a module have been executed at least once, and all error handling paths are tested to ensure that the software can handle errors correctly. Before any other testing can take place, it is essential to test data flow over a module interface. If data cannot enter and exit the system properly, all other tests are irrelevant.

Another crucial duty during unit testing is the selective examination of execution pathways to anticipate potential errors and ensure that error handling paths are set up to reroute or halt work when an error occurs. Finally, boundary testing is conducted to ensure that the software operates correctly at its limits. In the Sell-Soft System, unit testing was carried out by treating each module as a distinct entity and subjecting them to a variety of test inputs. Unused code was eliminated, and it was confirmed that every module was functional and produced the desired outcome.

5.2.2 Integration Testing

Integration testing is a systematic approach that involves creating the program structure while simultaneously conducting tests to identify interface issues. The objective is to construct a program structure based on the design that uses unit-tested components. The entire program is then tested. Correcting errors in integration testing can be challenging due to the size of the overall program, which makes it difficult to isolate the causes of the

errors. As soon as one set of errors is fixed, new ones may arise, and the process may continue in an apparently endless cycle. Once unit testing is complete for all modules in the system, they are integrated to check for any interface inconsistencies. Any discrepancies in program structures are resolved, and a unique program structure is developed.

5.2.3 Validation Testing or System Testing

The final stage of the testing process involves testing the entire software system, including all forms, code, modules, and class modules. This is commonly referred to as system testing or black box testing. The focus of black box testing is on testing the functional requirements of the software. A software engineer can use this approach to create input conditions that will fully test each program requirement. The main types of errors targeted by black box testing include incorrect or missing functions, interface errors, errors in data structure or external data access, performance errors, initialization errors, and termination errors.

5.2.4 Output Testing or User Acceptance Testing

User acceptance testing is performed to ensure that the system meets the business requirements and user needs. It is important to involve the end users during the development process to ensure that the software aligns with their needs and expectations. During user acceptance testing, the input and output screen designs are tested with different types of test data. The preparation of test data is critical to ensure comprehensive testing of the system. Any errors identified during testing are addressed and corrected, and the corrections are noted for future reference.

5.2.5 Automation Testing

Automation testing is a software testing approach that employs specialized automated testing software tools to execute a suite of test cases. Its primary purpose is to verify that the software or equipment operates precisely as intended. Automation testing identifies defects, bugs, and other issues that may arise during product development.

While some types of testing, such as functional or regression testing, can be performed manually, there are numerous benefits to automating the process. Automation testing can be

executed at any time of day and uses scripted sequences to evaluate the software. The results are reported, and this information can be compared to previous test runs. Automation developers typically write code in programming languages such as C#, JavaScript, and Ruby.

5.2.6 Selenium Testing

Selenium is an open-source automated testing framework used to verify web applications across different browsers and platforms. Selenium allows for the creation of test scripts in various programming languages such as Java, C#, and Python. Jason Huggins, an engineer at Thought Works, developed Selenium in 2004 while working on a web application that required frequent testing. He created a JavaScript program called "JavaScriptTestRunner" to automate browser actions and improve testing efficiency. Selenium has since evolved and continues to be developed by a team of contributors.

In addition to Selenium, another popular tool used for automated testing is Cucumber. Cucumber is an open-source software testing framework that supports behavior-driven development (BDD). It allows for the creation of executable specifications in a human readable format called Gherkin. By using a common language, Cucumber facilitates effective communication and collaboration during the testing process. It promotes a shared understanding of the requirements and helps ensure that the developed software meets the intended business goals.

Cucumber can be integrated with Selenium to combine the benefits of both tools. Selenium is used for interacting with web browsers and automating browser actions, while Cucumber provides a structured framework for organizing and executing tests. This combination allows for the creation of end-to-end tests that verify the behavior of web applications across different browsers and platforms, using a business-readable and maintainable format.

Test Case 1: User login

Code

```
package proj1;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class Assignment3{

    WebDriver driver = null;
    @Given("browser is open")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette", "C:\\\\Users\\sradhangeorg
e\\\\eclipse-workspace\\\\proj1\\\\src\\\\test\\\\resources\\\\Driver\\\\geckodriver.exe");
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login page")
    public void user_is_on_login_page() throws Throwable{
        driver.navigate().to("http://127.0.0.1:8000/login_page");
        Thread.sleep(2000);
    }

    @When("user enters username and password")
    public void user_enters_username_and_password() throws Throwable {
        driver.findElement(By.id("email")).sendKeys("akku@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Sradhaann$54");
        Thread.sleep(3000);
    }

    @And("User click on login")
    public void User_click_on_login() {
        driver.findElement(By.id("submit")).click();
    }

    @Then("user is navigated to the home page")
    public void user_is_navigated_to_the_home_page() throws Throwable {
        driver.findElement(By.id("t")).isDisplayed();
        Thread.sleep(3000);
        driver.close();
        driver.quit();
    }

}
```

Screenshot

```

Oct 29, 2023 10:23:29 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Check login is successfull with valid credential # src/test/resources/Assignment/oracle.feature:2
Oct 29, 2023 10:23:31 PM org.openqa.selenium.remote.service.DriverFinder getPath
WARNING: Unable to obtain geckodriver using Selenium Manager: Unsuccessful command executed: [C:\Users\SRADHA~1\AppData\Local\Temp\selenium-m
error sending request for url (https://github.com/mozilla/geckodriver/releases/latest): error trying to connect: dns error: No such host is k
Build info: version: '4.8.3', revision: 'e5e76298c3'
System info: os.name: 'Windows 11', os.arch: 'amd64', os.version: '10.0', java.version: '17.0.2'
Driver info: driver.version: FirefoxDriver
Given browser is open                                # proj1.Assignment3.browser_is_open()
    java.lang.IllegalStateException: The path to the driver executable Unable to locate the geckodriver executable; for more information on
    at org.openqa.selenium.internal.Require$StateChecker.nonNull(Require.java:314)
    at org.openqa.selenium.remote.service.DriverFinder.getPath(DriverFinder.java:33)
    at org.openqa.selenium.firefox.FirefoxDriver.generateExecutor(FirefoxDriver.java:139)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:131)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:127)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:112)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:102)
    at proj1.Assignment3.browser_is_open(Assignment3.java:18)
    at *.browser is open(file:///C:/Users/sradhangeorge/eclipse-workspace/proj1/src/test/resources/Assignment/oracle.feature:3)

And user is on login page                                # proj1.Assignment3.user_is_on_login_page()
When user enters username and password                    # proj1.Assignment3.user_enters_username_and_password()
And User click on login                                  # proj1.Assignment3.User_click_on_login()
Then user is navigated to the home page                  # proj1.Assignment3.user_is_navigated_to_the_home_page()
  
```

Test Report

Test Case 1

Project Name: CRAFTEDEUPHORIA					
Login Test Case					
Test Case ID: Test_1			Test Designed By: Sradha Ann George		
Test Priority(Low/Medium/High): High			Test Designed Date: 29/09/2023		
Module Name: : Login Screen			Test Executed By : Ms. Nimmy Francis		
Test Title : User Login			Test Execution Date: 29/09/2023		
Description: Verify Login with valid email and password					
Pre-Condition :User has valid username and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/ Fail)
1	Navigation to Login Page Login button		Dashboard should be displayed	Login page displayed	Pass
2	Provide Valid Email	Email: akku@gmail.com	User should be able to Login	User Logged in and navigated to User Dashboard	Pass
3	Provide Valid Password	Password: Sradhaann\$54			
4	Click on Login button				
Post-Condition: User is validated with database and successfully login into account. The Account session details are logged in database					

Test Case 2: Add to Cart

Code

```
package proj1;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.support.ui.WebDriverWait; // Import WebDriverWait
import org.openqa.selenium.support.ui.ExpectedConditions; // Import
ExpectedConditions
import org.openqa.selenium.support.ui.WebDriverWait;
import org.openqa.selenium.support.ui.ExpectedConditions;
import java.time.Duration;

public class Category {
    WebDriver driver = null;

    @Given("browser is open1")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette",
"C:\\Users\\sradhangeorge\\eclipse-
workspace\\proj1\\src\\test\\resources\\Driver\\geckodriver.exe");

        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login page1")
    public void user_is_on_login_page() throws Throwable {
        driver.navigate().to("http://127.0.0.1:8000/login_page");
        Thread.sleep(2000);
    }

    @When("user enters username and password1")
    public void user_enters_username_and_password() throws Throwable {
        driver.findElement(By.id("email")).sendKeys("akku@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Sradhaann$54");
        Thread.sleep(3000);
    }

    @And("User click on login1")
    public void User_click_on_login() {
        driver.findElement(By.id("submit")).click();
    }

    @And("user selects Birthday Gifts from header")
    public void user_selects_birthday_gifts_from_header() {
        WebElement categoriesLink = driver.findElement(By.id("cart"));
        categoriesLink.click(); // Click on the "Categories" link
    }

    @Then("user is navigated to the Birthday Gifts page")
}
```

```

    public void user_is_navigated_to_birthday_gifts_page() throws Throwable {
        // Add assertions to verify that you are on the Birthday Gifts page.
        // For example, you can check the page title or other elements on the
page.
        // You can also perform additional actions specific to this page.
        Thread.sleep(3000);
    }
}

```

Screenshot

```

Oct 29, 2023 10:57:57 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Check login is successful with valid credential # src/test/resources/Assignment/Cat.feature:2
Oct 29, 2023 10:57:58 PM org.openqa.selenium.remote.service.DriverFinder getPath
WARNING: Unable to obtain geckodriver using Selenium Manager: Unsuccessful command executed: [C:\Users\SRADHA~1\AppData\Local\Temp\selenium-m
error sending request for url (https://github.com/mozilla/geckodriver/releases/latest): error trying to connect: dns error: No such host is k
Build info: version: '4.8.3', revision: 'e5e76298c3'
System info: os.name: 'Windows 11', os.arch: 'amd64', os.version: '10.0', java.version: '17.0.2'
Driver info: driver.version: FirefoxDriver
Given browser is open() # proj1.Category.browser_is_open()
    java.lang.IllegalStateException: The path to the driver executable Unable to locate the geckodriver executable; for more information on
        at org.openqa.selenium.internal.Require$StateChecker.nonNull(Require.java:314)
        at org.openqa.selenium.remote.service.DriverFinder.getPath(DriverFinder.java:33)
        at org.openqa.selenium.firefox.FirefoxDriver.generateExecutor(FirefoxDriver.java:139)
        at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:131)
        at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:127)
        at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:112)
        at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:102)
        at proj1.Category.browser_is_open(Category.java:25)
        at *.browser is open(file:///C:/Users/sradhangeorge/eclipse-workspace/proj1/src/test/resources/Assignment/Cat.feature:3)

And user is on login page() # proj1.Category.user_is_on_login_page()
When user enters username and password() # proj1.Category.user_enters_username_and_password()
And user click on login() # proj1.Category.user_click_on_login()
And user selects Birthday Gifts from header() # proj1.Category.user_selects_birthday_gifts_from_header()
Then user is navigated to the Birthday Gifts page() # proj1.Category.user_is_navigated_to_birthday_gifts_page()

```

Test report

Test Case 2					
Project Name: CRAFTEDEUPHORIA					
Add to Cart Test Case					
Test Case ID: Test_2			Test Designed By: Sradha Ann George		
Test Priority(Low/Medium/High): High			Test Designed Date: 29/09/2023		
Module Name: Cart Screen			Test Executed By : Ms. Nimmy Francis		
Test Title : Add to Cart			Test Execution Date: 29/09/2023		
Description: Products is added to the cart by user					
Pre-Condition :User has valid username and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/Fail)
1	Navigation to Login Page Login button		Dashboard should be displayed	Login page displayed	Pass
2	Provide Valid Email	Email: akku@gmail.	User	User Logged	Pass

		com	should be able to Login	in and navigated to User Dashboard	
3	Provide Valid Password	Password: Sradhaann\$54			
4	Click on Login button				
5	User is navigated to product list page and select the products and move to add to cart		User should redirect to cart page	Cart page with products should be displayed	Pass
6	Logout and navigate to login page		Login page should be displayed	Redirect to login page	Pass
Post-Condition: User successfully added an item to the cart					

Test Case 3: Search

Code

```
package proj1;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.openqa.selenium.WebElement;

public class Search{

    WebDriver driver = null;
    @Given("browser is open2")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette", "C:\\\\Users\\sradhangeorge\\eclipse-workspace\\proj1\\src\\test\\resources\\Driver\\geckodriver.exe");
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login page2")
    public void user_is_on_login_page() throws Throwable{
        driver.navigate().to("http://127.0.0.1:8000/login_page");
    }
}
```

```

Thread.sleep(2000);
}

@When("user enters username and password2")
public void user_enters_username_and_password() throws Throwable {
    driver.findElement(By.id("email")).sendKeys("akku@gmail.com");
    driver.findElement(By.id("password")).sendKeys("Sradhaann$54");
    Thread.sleep(3000);
}

@And("User click on login2")
public void User_click_on_login() {
    driver.findElement(By.id("submit")).click();
}

@Then("user is navigated to the home page2")
public void user_is_navigated_to_the_home_page() throws Throwable {

    // Now, you should be on the "All Property" page.
    // You can add additional assertions or actions here as needed.

    // Don't forget to close the WebDriver when done.
    WebElement cupassField = driver.findElement(By.id("productname"));

    cupassField.sendKeys("toys");

    WebElement saveButton = driver.findElement(By.id("search-button"));
    saveButton.click();
    Thread.sleep(3000);
    driver.close();
    driver.quit();
}
}

```

Screenshot

```

Oct 29, 2023 11:03:08 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Check login is successful with valid credential # src/test/resources/Assignment/find.feature:2
Oct 29, 2023 11:03:09 PM org.openqa.selenium.remote.service.DriverFinder getPath
WARNING: Unable to obtain geckodriver using Selenium Manager: Unsuccessful command executed: [C:\Users\SRADHA-1\AppData\Local\Temp\selenium-m
error sending request for url (https://github.com/mozilla/geckodriver/releases/latest): error trying to connect: dns error: No such host is k
Build info: version: '4.8.3', revision: 'e5e76298c3'
System info: os.name: 'Windows 11', os.arch: 'amd64', os.version: '10.0', java.version: '17.0.2'
Driver info: driver.version: FirefoxDriver
Given browser is open2                                # proj1.Search.browser_is_open()
  java.lang.IllegalStateException: The path to the driver executable Unable to locate the geckodriver executable; for more information on
    at org.openqa.selenium.internal.Require$StateChecker.nonNull(Require.java:314)
    at org.openqa.selenium.remote.service.DriverFinder.getPath(DriverFinder.java:33)
    at org.openqa.selenium.firefox.FirefoxDriver.generateExecutor(FirefoxDriver.java:139)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:131)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:127)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:112)
    at org.openqa.selenium.firefox.FirefoxDriver.<init>(FirefoxDriver.java:102)
    at proj1.Search.browser_is_open(Search.java:18)
    at *.browser is open2(file:///C:/Users/sradhangeorge/eclipse-workspace/proj1/src/test/resources/Assignment/find.feature:3)

And user is on login page2                                # proj1.Search.user_is_on_login_page()
When user enters username and password2                    # proj1.Search.user_enters_username_and_password()
And User click on login2                                    # proj1.Search.User_click_on_login()
Then user is navigated to the home page2                    # proj1.Search.user_is_navigated_to_the_home_page()

```

Test report

Test Case 3					
Project Name: CRAFTEDEUPHORIA					
Search Test Case					
Test Case ID: Test_3			Test Designed By: Sradha Ann George		
Test Priority(Low/Medium/High): High			Test Designed Date: 29/09/2023		
Module Name: Search Screen			Test Executed By : Ms. Nimmy Francis		
Test Title : Search Products			Test Execution Date: 29/09/2023		
Description: Search approved by Admin					
Pre-Condition :Admin has valid username and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/Fail)
1	Navigation to Login Page		Login Page should be displayed	Login page displayed	Pass
2	Provide Valid Email	Email: Admin	Admin should be able to Login	Admin Logged in and navigated to User Dashboard	Pass
3	Provide Valid Password	Password: admin			
4	Click on Login button				
5	Click on Product List		Product list page should be displayed	Admin navigated to product list page	Pass
6	Provide Products name	Product name: Toys			
7	Click on Submit button		Admin should be able to search products by name	Admin searched products by name	Pass
Post-Condition: Products searched by Admin					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

The implementation phase of a project is where the design is transformed into a functional system. It is a crucial stage in ensuring the success of the new system, as it requires gaining user confidence that the system will work effectively and accurately. User training and documentation are key concerns during this phase. Conversion may occur concurrently with user training or at a later stage. Implementation involves the conversion of a newly revised system design into an operational system.

During this stage, the user department bears the primary workload, experiences the most significant upheaval, and has the most substantial impact on the existing system. Poorly planned or controlled implementation can cause confusion and chaos. Whether the new system is entirely new, replaces an existing manual or automated system, or modifies an existing system, proper implementation is essential to meet the organization's needs. System implementation involves all activities required to convert from the old to the new system. The system can only be implemented after thorough testing is done and found to be working according to specifications. The implementation phase involves careful planning, investigating system and constraints, and designing methods to achieve changeover.

6.2 IMPLEMENTATION PROCEDURES

Software implementation is the process of installing the software in its actual environment and ensuring that it satisfies the intended use and operates as expected. In some organizations, the software development project may be commissioned by someone who will not be using the software themselves. During the initial stages, there may be doubts about the software, but it's important to ensure that resistance does not build up.

This can be achieved by:

- Ensuring that active users are aware of the benefits of the new system, building their confidence in the software.
- Providing proper guidance to the users so that they are comfortable using the application. Before viewing the system, users should know that the server program must be running on the server. Without the server object up and running, the intended process will not take place.

6.2.1 User Training

User training is designed to prepare the user for testing and converting the system. To achieve the objective and benefits expected from computer-based system, it is essential for the people who will be involved to be confident of their role in the new system. As system becomes more complex, the need for training is more important. By user training the user comes to know how to enter data, respond to error messages, interrogate the database, and call up routine that will produce reports and perform other necessary functions.

6.2.2 Training on the Application Software

After providing the necessary basic training on computer awareness, it is essential to provide training on the new application software to the user. This training should include the underlying philosophy of using the new system, such as the flow of screens, screen design, the type of help available on the screen, the types of errors that may occur while entering data, and the corresponding validation checks for each entry, and ways to correct the data entered. Additionally, the training should cover information specific to the user or group, which is necessary to use the system or part of the system effectively. It is important to note that this training may differ across different user groups and levels of hierarchy.

6.2.3 System Maintenance

The maintenance phase is a crucial aspect of the software development cycle, as it is the time when the software is use and performs its intended functions. Proper maintenance is essential to ensure that the system remains functional, reliable, and adaptable to changes in the system environment. Maintenance activities go beyond simply identifying and fixing errors or bugs in the system. It may involve updates to the software, modifications to its functionalities, and enhancements to its performance, among other things. In essence, software maintenance is an ongoing process that requires continuous monitoring, evaluation, and improvement of the system to meet changing user needs and requirements.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In conclusion, the development of the personalized online gift store represents a comprehensive and feature-rich platform designed to offer users a seamless and enjoyable shopping experience for all their gifting needs. This mini project encompasses various modules and features, ensuring a well-rounded and interactive environment for both users and administrators.

The User Module allows users to create accounts, browse through a vast catalog, view detailed product information, use a convenient search bar, add items to their cart, and efficiently complete their purchases. The addition of a secure payment gateway ensures that online transactions are safe and convenient.

The Admin Module empowers administrators to manage products, handle orders, and monitor inventory in real-time. It streamlines the process of ensuring that the gift store remains well-stocked and efficient. Additionally, integrating popular digital wallets makes payments hassle-free for users.

The Community Module fosters engagement and creativity by allowing users to add their own products and collections. Users can personalize their profiles, connect with others, and share their gift ideas, creating a sense of community within the platform.

This mini project's successful implementation hinges on a user-centric design, secure e-commerce functionality, and community engagement, making it a valuable resource for both gift shoppers and store administrators. Overall, the personalized online gift store project embodies the potential to create a vibrant, user-friendly, and thriving online platform for users seeking the perfect gifts for any occasion.

7.2 FUTURE SCOPE

The personalized online gift store project has numerous potential areas for growth and enhancement. These include developing a mobile application, implementing AI and machine learning for accurate product recommendations, allowing user reviews and ratings, expanding gift personalization options, expanding the store's global reach, introducing a gift registry system for special occasions, introducing sustainable and ethical products, offering gift cards and vouchers, enhancing social integration, implementing robust data analytics tools, investing in customer support and chatbots, offering gift wrapping and delivery services, collaborating with local artisans, exploring blockchain technology for supply chain transparency, and incorporating augmented reality for virtual gifting.

These developments will help the store adapt to technological advancements, evolving consumer preferences, and innovative ideas, ensuring its long-term success and growth in the competitive e-commerce market. By continuously adapting and expanding the platform, the personalized online gift store project can continue to thrive in the competitive e-commerce market.

.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- Pankaj Jalote, “Software engineering: a precise approach”
- Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009
- Ken Schwaber, Mike Beedle, Agile Software Development with Scrum, Pearson (2008)
- Roger S Pressman, “Software Engineering”
- IEEE Std 1016 Recommended Practice for Software Design Descriptions.

WEBSITES:

- <https://www.smarteyeapps.com>
- www.w3schools.com
- www.bootstrap.com
- <https://chat.openai.com/chat>

CHAPTER 9

APPENDIX

9.1 Sample Code

Login

```
{% load static %}
{% load socialaccount %}
{% block content %}
<html>
<head>
  <title>Login</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <style>
    .image-container {
      background-size: cover;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }

    .form-container {
      background-color: white;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
    }

    .image-container img {
      max-width: 100%;
      height: auto;
      display: block;
    }
  </style>
</head>
<body>
  <div class="container-fluid">
```

```

<div class="row">
  <div class="col-md-6 image-container">
    
  </div>
  <div class="col-md-6">
    <div class="container form-container mt-5">
      <h1>Login</h1>
      {% for messages in messages %}
      <h3 style="color: red">{{ messages }}</h3>
      {% endfor %}
      <form id="loginForm" action="#" method="POST">
        {% csrf_token %}
        <div class="form-group">
          <label for="email">Email:</label>
          <input class="form-control" id="email" name="email" >
          <small id="emailError" class="form-text text-danger"></small>
        </div>

        <div class="form-group">
          <label for="password">Password:</label>
          <input type="password" class="form-control" id="password" name="password"
required>
          <small id="passwordError" class="form-text text-danger"></small>
        </div>

        <button type="submit" class="btn btn-primary" id="submit">Login</button>
      </form>

      <p class="mt-3">Or</p>
      <a href="{% provider_login_url 'google'%}"?next="/" class="btn btn-primary btn-block mt-3" onclick="signInWithGoogle()">
        <span class="align-middle">
          <img src="https://cdn-icons-

```



```

png.flaticon.com/512/281/281764.png?w=740&t=st=1691744655~exp=1691745255~hmac=0599
0f95cf17e4c6378ea22d0d190f708342a8954403cb069955a97e2c0fd10c" alt="Google Icon"
width="24" height="24">
    </span>
    <span class="align-middle ml-2">Sign in with Google</span>
</a>
<div class="mt-3">
    <p>Don't have an account yet? <a href="{ % url 'register' % }">Sign up</a></p>
</div>
</div>
</div>
</div>
</div>
{ % endblock % }
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

Registration

```

<!DOCTYPE html>
<html>
<head>
    <title>Sign Up</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        .image-container {
            background-size: cover;
            min-height: 100vh;
            display: flex;

```

```
align-items: center;
justify-content: center;
}

.form-container {
background-color: white;
padding: 20px;
border-radius: 5px;
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
}

.image-container img {
max-width: 100%;
height: auto;
display: block;
}
</style>
</head>
<body>
<div class="container-fluid">
<div class="row">
<div class="col-md-6 image-container">

</div>
<div class="col-md-6">
<div class="container form-container mt-5">
<h1>Sign Up</h1>
{ % for messages in messages % }
<h3 style="color: red">{ { messages } }</h3>
{ % endfor % }
<form method="post" action="#" id="registrationForm" enctype="multipart/form-data">
{ % csrf_token % }

<div class="form-group">
```

```
<label for="username">Name:</label>
<input type="text" class="form-control" id="name" name="username" required>
<small id="usernameError" class="form-text text-danger"></small>
</div>

<div class="form-group">
  <label for="email">Email:</label>
  <input type="email" class="form-control" id="email" name="email" required>
  <small id="emailError" class="form-text text-danger"></small>
</div>

<div class="form-group">
  <label for="phoneNumber">Phone Number:</label>
  <input type="tel" class="form-control" id="phoneNumber" name="mobile" required>
  <small id="phoneNumberError" class="form-text text-danger"></small>
</div>

<div class="form-group">
  <label for="password">Password:</label>
  <input type="password" class="form-control" id="password" name="password"
required>
  <small id="passwordError" class="form-text text-danger"></small>
</div>

<div class="form-group">
  <label for="confirmPassword">Confirm Password:</label>
  <input type="password" class="form-control" id="confirmPassword"
name="confirmPassword" required>
  <small id="confirmPasswordError" class="form-text text-danger"></small>
</div>

<div>
  <p>Want to be a Seller</p>
  <input type="radio" id="customer" name="user_role" value="customer" checked>
  <label for="customer">No</label>
```

```

        <input type="radio" id="staff" name="user_role" value="staff">
        <label for="staff">Yes</label>
    </div>
    <div class="form-group" id="certificationImageField" style="display: none;">
        <label for="certification_image">Certification Image:</label>
        <input type="file" class="form-control" id="certification_image"
name="certification_image">
        <small id="certificationImageError" class="form-text text-danger"></small>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
    <div class="mt-3">
        <p>Already have an account? <a href="{ % url 'login_page' % }">Login</a></p>
    </div>
</form>
</div>
</div>
</div>
</div>
<!-- Add Bootstrap JS and jQuery scripts -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script>
    const sellerRadio = document.getElementById('staff');
    const certificationImageField = document.getElementById('certificationImageField');

    sellerRadio.addEventListener('change', function () {
        if (sellerRadio.checked) {
            certificationImageField.style.display = 'block';
        } else {
            certificationImageField.style.display = 'none';
        }
    })

```

```
});  
</script>
```

```
<script>
```

```
const registrationForm = document.getElementById('registrationForm');  
const nameInput = document.getElementById('name');  
const nameError = document.getElementById('usernameError');  
const emailInput = document.getElementById('email');  
const phoneNumberInput = document.getElementById('phoneNumber');  
const passwordInput = document.getElementById('password');  
const confirmPasswordInput = document.getElementById('confirmPassword');  
const certificationImageInput = document.getElementById('certification_image');  
const certificationImageError = document.getElementById('certificationImageError');
```

```
function validateName() {  
const nameValue = nameInput.value;
```

```
// Regular expression to check if the name contains only alphabets and starts with a capital letter  
const namePattern = /^[A-Z][a-zA-Z]*$/;
```

```
if (!namePattern.test(nameValue)) {  
    nameError.textContent = 'Name should start with a capital letter and contain only alphabets.';  
    return false;  
} else {  
    nameError.textContent = "";  
    return true;  
}  
}
```

```
function validateEmail() {
```

```
const emailValue = emailInput.value;
const emailError = document.getElementById('emailError');

if (!/^\S+@\S+\.\S+$/i.test(emailValue)) {
    emailError.textContent = 'Enter a valid email address.';
    return false;
} else {
    emailError.textContent = '';
    return true;
}
}

function validatePhoneNumber() {
    const phoneNumberValue = phoneNumberInput.value;
    const phoneNumberError = document.getElementById('phoneNumberError');

    if (!/^[6-9]\d{9}$/i.test(phoneNumberValue)) {
        phoneNumberError.textContent = 'Enter a valid phone number.';
        return false;
    } else {
        phoneNumberError.textContent = '';
        return true;
    }
}

function validatePassword() {
    const passwordValue = passwordInput.value;
    const passwordError = document.getElementById('passwordError');

    if (!/(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*\W){8,}/i.test(passwordValue)) {
        passwordError.textContent = 'Password should contain at least 8 characters, one uppercase letter, one lowercase letter, one number, and one special character.';
        return false;
    } else {
        passwordError.textContent = '';
    }
}
```

```
    return true;
  }
}
```

```
function validateConfirmPassword() {
  const confirmPasswordValue = confirmPasswordInput.value;
  const confirmPasswordError = document.getElementById('confirmPasswordError');

  if (confirmPasswordValue !== passwordInput.value) {
    confirmPasswordError.textContent = 'Passwords do not match.';
    return false;
  } else {
    confirmPasswordError.textContent = "";
    return true;
  }
}
```

```
function validateCertificationImage() {
  const certificationImageValue = certificationImageInput.value;
  const allowedExtensions = ['.jpg', '.jpeg', '.png'];
  const extension = certificationImageValue.substring(certificationImageValue.lastIndexOf('.') + 1).toLowerCase();

  if (!allowedExtensions.includes(`.${extension}`)) {
    certificationImageError.textContent = 'File format must be JPEG, PNG, or JPG.';
    return false;
  } else {
    certificationImageError.textContent = "";
    return true;
  }
}
```

```
nameInput.addEventListener('blur', validateName);
emailInput.addEventListener('keyup', validateEmail);
```

```
phoneNumberInput.addEventListener('keyup', validatePhoneNumber);
passwordInput.addEventListener('keyup', validatePassword);
confirmPasswordInput.addEventListener('keyup', validateConfirmPassword);
certificationImageInput.addEventListener('change', validateCertificationImage);

// Validate on form submit
registrationForm.addEventListener('submit', function(event) {
  // const isUsernameValid = validateUsername();
  const isEmailValid = validateEmail();
  const isPhoneNumberValid = validatePhoneNumber();
  const isPasswordValid = validatePassword();
  const isConfirmPasswordValid = validateConfirmPassword();

  if (!isEmailValid || !isPasswordValid || !isConfirmPasswordValid) {
    event.preventDefault();
  }
});

document.querySelector('a[href="#"]').addEventListener('click', function(event) {
  event.preventDefault();
});
</script>
</body>
</html>
```

Product List page

```
{ % load static % }
<!Doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title> CraftedEuphoria</title>
  <link rel="shortcut icon" href="{ % static 'assets/images/fav.png' % }" type="image/x-
icon">
```



```
<link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700;800;900&a
mp;display=swap" rel="stylesheet">
<link rel="shortcut icon" href="{ % static 'assets/images/fav.jpg' % }">
<link rel="stylesheet" href="{ % static 'assets/css/bootstrap.min.css' % }">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.7.2/font/bootstrap-icons.css">
<link rel="stylesheet" type="text/css" href="{ % static 'assets/css/style.css' % }"/>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">

<style>
.product-box {
    height: 100%; /* Set a fixed height for the box */
}
.product-image {
    width: 100%; /* Set a fixed width for the image */
    height: 200px; /* Set a fixed height for the image */
    object-fit: cover; /* Maintain aspect ratio and cover the entire space */
}
</style>
</head>
<body>
<header class="container-fluid bg-white">
<div class="header-top bg-gray border-bottom">
<div class="container">
<div class="row">
<div class="col-md-8">
<ul class="d-inline-flex pt-0 pt-md-2 fs-6">
</ul>
</div>
<div class="col-md-4 d-flex align-items-end">
<ul class="ms-auto d-inline-flex">
{ % if user.is_authenticated % }
```

```

        <li class="p-2"><a target="_blank" href="{ % url 'logout' % }"><button
class="btn px-4 btn-danger">Logout</button></a></li>

        <li class="p-2"><a target="_blank" href="{ % url 'userprofile' % }"><button
class="btn px-4 btn-danger">User Profile</button></a></li>

        { % else % }

        <li class="p-2"><a target="_blank" href="{ % url 'login_page' % }"><button
class="btn px-4 btn-danger">Login</button></a></li>

        <li class="p-2"><a target="_blank" href="{ % url 'register' % }"><button
class="btn px-4 btn-outline-danger">Sign Up</button></a></li>

        { % endif % }

    </ul>

</div>

</div>

</div>

</div>

<header>

    <div class="logo-container p-2">
        <div class="container">
            <div class="row">
                <div class="col-md-3 col-9 pt-1 pb-2">

                </div>

                <div class="col-md-6 d-none d-md-block pt-2">

<div class="row">
    <div class="col-md-12">
        <form id="search-form" method="GET" action="{ % url 'search_product' % }">
            <div class="input-group">
                <input type="text" class="form-control" name="productname" id="productname"

```

```

placeholder="Search by Product Name" oninput="toggleSearchButton()">
    <div class="input-group-append">
        <button class="btn btn-primary" id="search-button" type="submit" disabled><i
class="fa fa-search"></i> </button>
    </div>
</div>
</div>
</form>
</div>
</div>
<script>
function toggleSearchButton() {
    const productNameInput = document.getElementById("productname");
    const searchButton = document.getElementById("search-button");

    if (productNameInput.value.trim() === "") {
        searchButton.disabled = true;
    } else {
        searchButton.disabled = false;
    }
}
</script>

</div>
<div class="col-md-3 col-3 pt-1 text-end">
    <a href="{% url 'cart' %}">
        <button type="button" class="btn btn-light shadow-md border position-
relative">
            <i class="bi fs-4 bi-basket"></i>
        </button>
    </a>
    <button type="button" class="btn d-none d-md-inline-block ms-3 btn-light
shadow-md border position-relative">
        <i class="bi fs-4 bi-heart"></i>
    </button>
</div>

```

```

        </div>
    </div>
</div>
<div class="menu-bar bg-danger container-fluid border-top">
    <div class="container">
        <h6 class="d-md-none text-white p-3 mb-0 fw-bold">Menu
            <a class="text-white" data-bs-target="#menu" data-bs-toggle="collapse" aria-
expanded="false" aria-controls="menu"><i class="bi cp bi-list float-end fs-1 dmji"></i></a>
        </h6>
        <ul id="menu" class="navcol fw-bold d-none d-md-inline-flex">
            <li class="p-21 px-4">
                <a class="text-white" href="">Categories<i class="bi pt-2 bi-chevron-
down"></i></a>
                <div class="inner-div">
                    <ul class="">
                        <li><a href="{ % url 'products_list' category_id=8 % }">Birthday
Gifts</a></li>
                        <li><a href="{ % url 'products_list' category_id=4 % }">Anniversary
Gifts</a></li>
                        <li><a href="{ % url 'products_list' category_id=7 % }">Special
Occasion</a></li>
                        <li><a href="{ % url 'products_list' category_id=3 % }">Festive
Special Gifts</a></li>
                        <li><a href="{ % url 'products_list' category_id=5 % }">Customized
Gifts</a></li>
                    </ul>
                </div>
            </li>
            <li class="p-21 px-4 active"><a class="text-white" href="{ % url 'index'
% }">Home </a></li>
            <li class="p-21 px-4"><a class="text-white" href="{ % url 'about' % }">About
Us </a></li>
        </ul>
    </div>

```

```

        </div>
    </div>
</header>
</header>
<div class="row mt-5">
    {% for product in products %}
    <div class="col-lg-3 col-md-4 mb-4">
        <div class="product-box bg-white p-2 shadow-md">
            <div class="text-center">
                <a href="">
                    
                </a>
            </div>
            <div class="detail p-2">
                <h4 class="mb-1 fs-5 fw-bold">{{ product.name }}</h4>
                <h4 class="mb-1 fs-5">{{ product.description }}</h4>
                <b class="fs-4 text-danger">${{ product.price }}</b>

                <ul class="mt-0 vgt h">
                    <li class="fs-8">

                        </li>
                        <li class="float-end gvi">
                            <i class="bi text-danger bi-heart-fill"></i>
                        </li>
                    </ul>
                <div class="row pt-2">
                    <div class="col-md-6">
                        <a href="{% url 'product_detail' product.id %}">
                            <button class="btn mb-2 fw-bold w-100 btn-danger">Buy Now</button>
                        </a>
                    </div>
                    <div class="col-md-6">

```

```

        <!-- Add to Cart Form -->
        <form method="post">
            { % csrf_token % }
            <!-- Add a hidden input for the product ID -->
            <input type="hidden" name="product_id" value="{{ product.id }}">

            <!-- Product Name (Hidden Input) -->
            <input type="hidden" name="product_name" value="{{ product.name }}">

            <!-- Product Price (Hidden Input) -->
            <input type="hidden" name="product_price" value="{{ product.price }}">

            <!-- Quantity Input -->
            <input type="number" name="quantity" value="1" min="1">

            <!-- Add to Cart Button -->
            <button type="submit" class="btn mb-2 fw-bold w-100 btn-outline-
danger">Add to Cart</button>
        </form>
    </div>
</div>
</div>
</div>
</div>
</div>
    { % endfor % }
</div>
</body>
</html>

```

Add to Cart

```

    { % load static % }
    <!DOCTYPE html>
    <html lang="en">

```

```
<head>
  <meta charset="UTF-8">
  <title>Cart</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f0f0;
      margin: 0;
      padding: 0;
    }

    .cart-container {
      background-color: #fff;
      margin: 20px auto;
      padding: 20px;
      width: 80%;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
    }

    h1 {
      background-color: #333;
      color: #fff;
      padding: 20px;
      margin: 0;
      text-align: center;
    }

    ul {
      list-style: none;
      padding: 0;
    }

    li {
```

```
background-color: #f9f9f9;
margin: 10px 0;
padding: 10px;
border: 1px solid #ddd;
border-radius: 5px;
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
display: flex;
justify-content: space-between;
align-items: center;
}
```

```
.product-image {
width: 100px;
height: 100px;
margin-right: 10px;
}
```

```
.quantity {
display: flex;
align-items: center;
}
```

```
.quantity button {
background-color: #333;
color: #fff;
border: none;
padding: 5px 10px;
margin: 0 5px;
border-radius: 5px;
cursor: pointer;
}
```

```
a {
display: inline-block;
```



```

        background-color: #333;
        color: #fff;
        text-decoration: none;
        padding: 10px 20px;
        margin-top: 20px;
        border-radius: 5px;
        transition: background-color 0.3s ease;
    }

    a:hover {
        background-color: #555;
    }
</style>
</head>
<body>
<!-- cart.html -->
<a href="{% url 'index' %}">Continue Shopping</a>
<div class="cart-container">
    <h1>Cart</h1>
    <ul>
        {% if cart_items %}
            {% for item in cart_items %}
                {% if item.status == 1 %}
                    <li>
                        <div>
                            
                            {{ item.product.name }} - Price: ${{ item.product.price }}
                        </div>
                        <div class="quantity">
                            <!-- {% for product_id, item in cart.items %}
                                <p>{{ item.name }} - ${{ item.price }} (Quantity: {{ item.quantity }})
                                    <a href="{% url 'remove_from_cart' product_id %}">Remove</a></p>
                                {% endfor %} -->

```

```
<button onclick="decrementQuantity('{{ item.id }}', {{ item.product.price }}) ">-
</button>

<span id="quantity_{{ item.id }}">{{ item.quantity }}</span>

<button onclick="incrementQuantity('{{ item.id }}', {{ item.product.price
}})">+</button>

<a href="{% url 'remove_from_cart' item.product.id %}">Remove</a>
</div>
</li>
{% endif %}
{% endfor %}
{% else %}
<li>
<p>Your cart is empty.</p>
</li>
{% endif %}
</ul>
<div id="total-price">Total Price: ${{ total_price }}</div>
<a href="{% url 'billing' user_id=request.user.id %}">Checkout</a>
</div>

<script>

var totalPrice = {{ total_price }};

function incrementQuantity(productId, price) {
    var quantityElement = document.getElementById('quantity_' + productId);
    var currentQuantity = parseInt(quantityElement.textContent);
    currentQuantity++;
    quantityElement.textContent = currentQuantity;
    updateTotalPrice(price);
}
```

```
function decrementQuantity(productId, price) {
    var quantityElement = document.getElementById('quantity_' + productId);
    var currentQuantity = parseInt(quantityElement.textContent);
    if (currentQuantity > 1) {
        currentQuantity--;
        quantityElement.textContent = currentQuantity;
        updateTotalPrice(-price);
    }
}

function updateTotalPrice(priceChange) {
    totalPrice += priceChange;
    var totalElement = document.getElementById('total-price');
    totalElement.textContent = 'Total Price: $' + totalPrice.toFixed(2);
}

function remove_from_cart(productId, price) {
    var quantityElement = document.getElementById('quantity_' + productId);
    var currentQuantity = parseInt(quantityElement.textContent);
    var listItem = quantityElement.parentElement.parentElement;
    listItem.remove();
    updateTotalPrice(-price * currentQuantity);

    var cartList = document.querySelector('ul');
    if (cartList.children.length === 0) {
        cartList.innerHTML = '<li><p>Your cart is empty.</p></li>';
    }
}

</script>

</body>
</html>
```

Payment

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Payment Details</title>
  <style>
    /* Reset some default styles */
    body, h1, form {
      margin: 0;
      padding: 0;
    }

    body {
      font-family: Arial, Helvetica, sans-serif;
      background-color: #f2f2f2;
    }

    h1 {
      text-align: center;
      margin-top: 20px;
      color: #333;
    }

    form {
      background-color: #fff;
      max-width: 400px;
      margin: 0 auto;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
```

```
}

label {
    font-weight: bold;
}

input[type="text"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}

button[type="submit"] {
    display: block;
    width: 100%;
    padding: 10px;
    background-color: #007BFF;
    color: #fff;
    border: none;
    border-radius: 5px;
    font-size: 18px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button[type="submit"]:hover {
    background-color: #0056b3;
}

.container {
    position: relative;
    padding: 20px; /* Add padding around the container as needed */
}
```

```
}
```

```
.cart-link-box {  
  position: absolute;  
  top: 10px;  
  left: 10px;  
  background-color: black; /* Background color for the box */  
  color: white; /* Text color */  
  padding: 10px 20px; /* Padding inside the box */  
  border: 1px solid #333; /* Border for the box */  
  border-radius: 5px; /* Rounded corners */  
}
```

```
</style>  
</head>  
<body>  
<div class="container">  
  <div class="cart-link-box">  
    <a href="{ % url 'cart' % }" style="color: white; text-decoration: none;">Back to Cart</a>  
  </div>  
  <h1>Payment Details</h1>  
  <form action="{ % url 'billing' user_id=request.user.id % }" method="POST">  
    { % csrf_token % }  
    <!-- Billing Information -->  
    <h2>Billing Address</h2>  
    <label for="billing_name">Name:</label>  
    <input type="text" id="billing_name" name="name" placeholder=" Name" value="{ {  
user.userprofile.name } }" required>  
    <br>  
  
    <label for="billing_address">Address:</label>  
    <input type="text" id="billing_address" name="address" placeholder=" Address" value=""  
required>  
    <br>
```

```
<label for="billing_city">Town:</label>
<input type="text" id="billing_city" name="town" placeholder=" Town" value="" required>
<br>

<label for="billing_zip">ZIP Code:</label>
<input type="text" id="billing_zip" name="zip_code" placeholder=" ZIP Code" value=""
required>
<br>

<!-- Amount -->
<h2>Amount</h2>
<!-- Show the amount here (e.g., dynamically from your server-side code) -->
<p>{{ total_price }}</p>

<!-- Submit Button -->
<button type="submit">Submit Payment</button>
</form>
</body>
</html>
```

9.1 Screen Shots

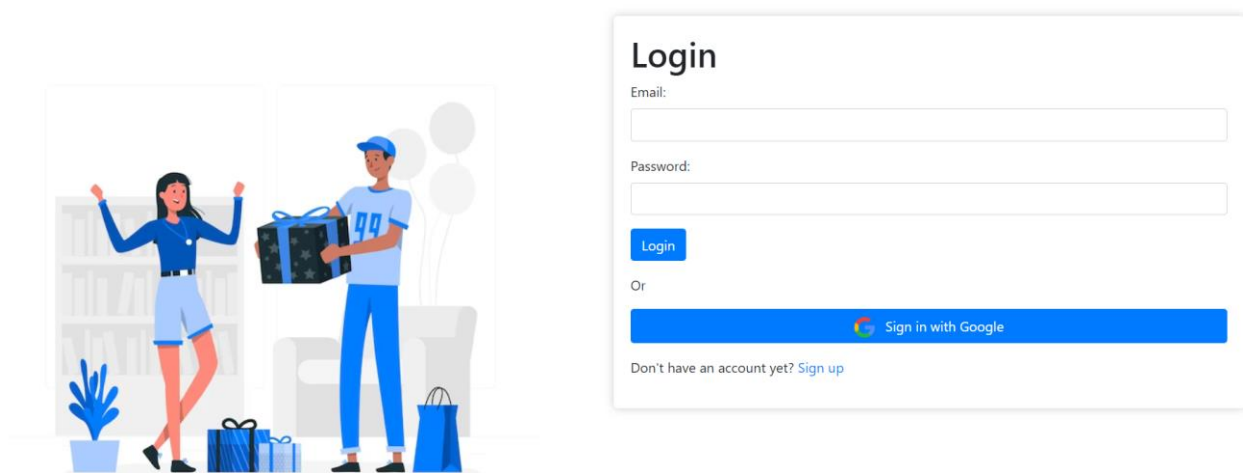



Fig 1: Login Page



Sign Up

Name:

Email:

Phone Number:

Password:

Confirm Password:

Want to be a Seller
☒ No ☐ Yes

Certification Image:
 No file chosen

Already have an account? [Login](#)

Fig 2: Registration Page

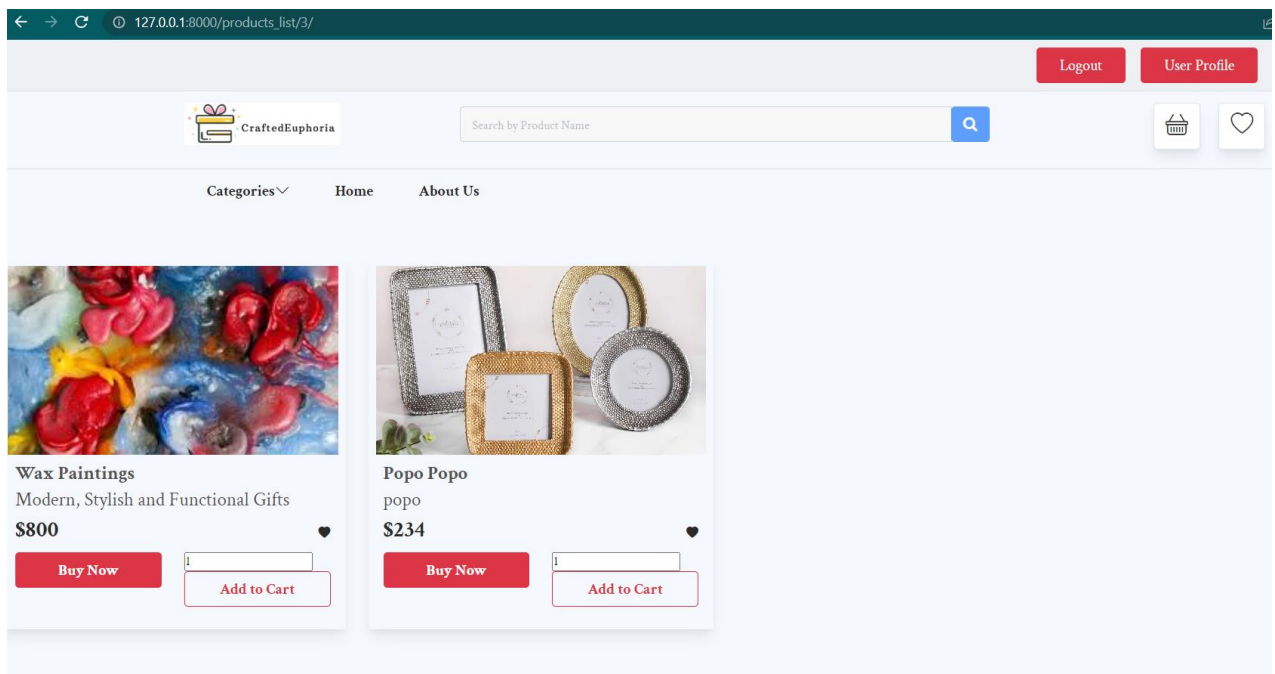


Fig 3: Product List Page

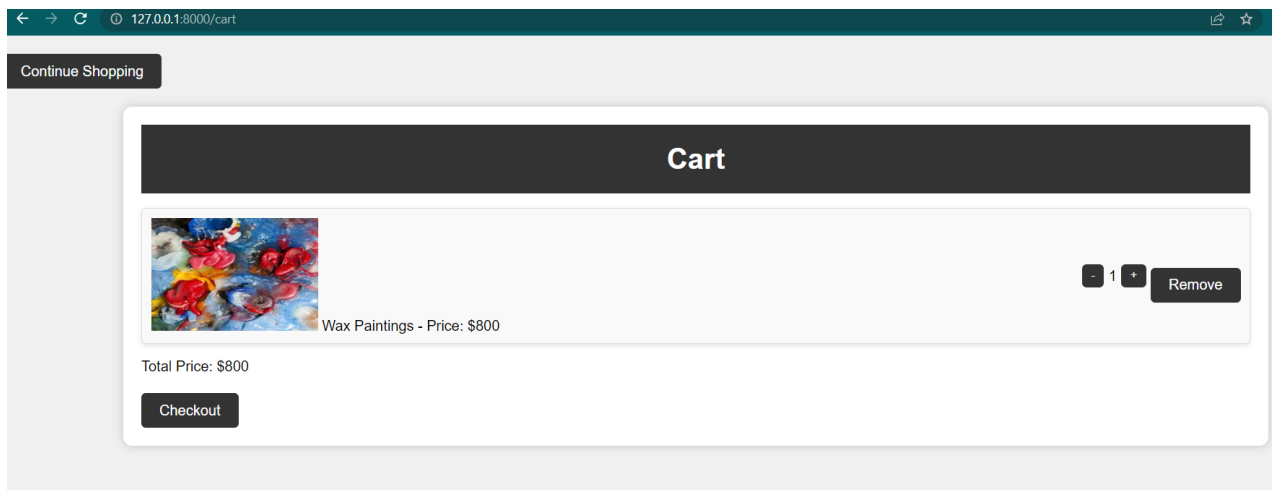


Fig 4: Add to Cart Page

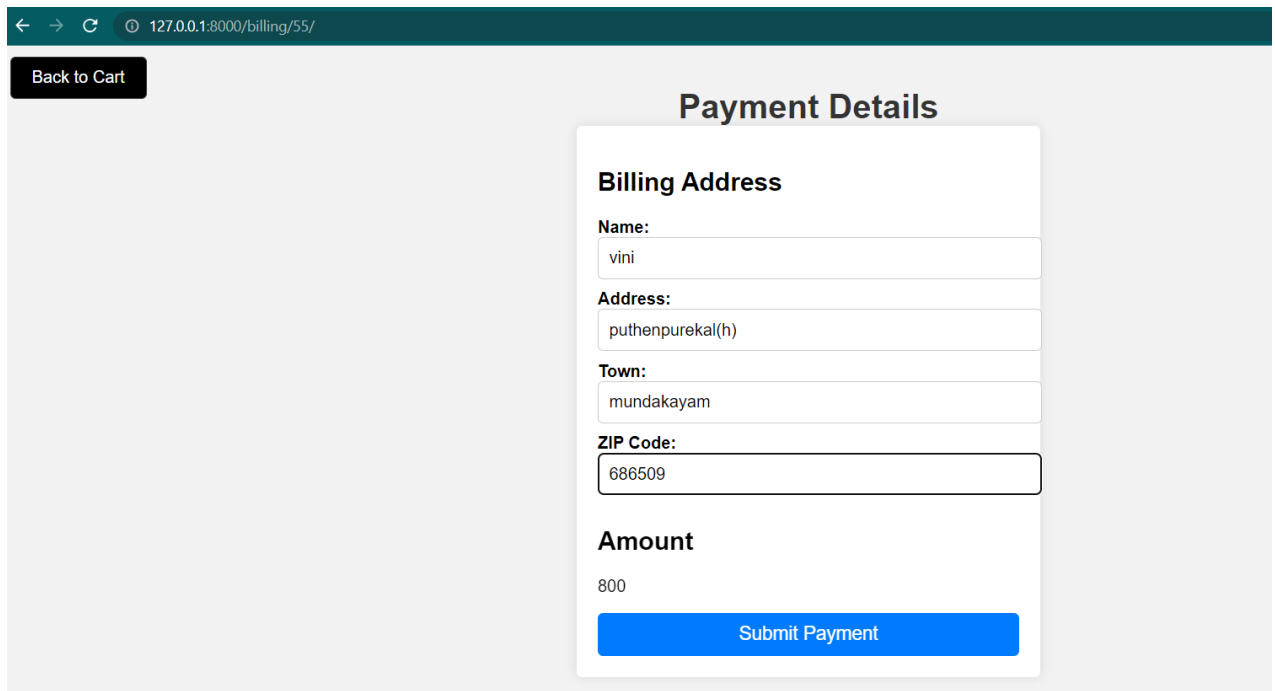


Fig 5: Payment Details Page

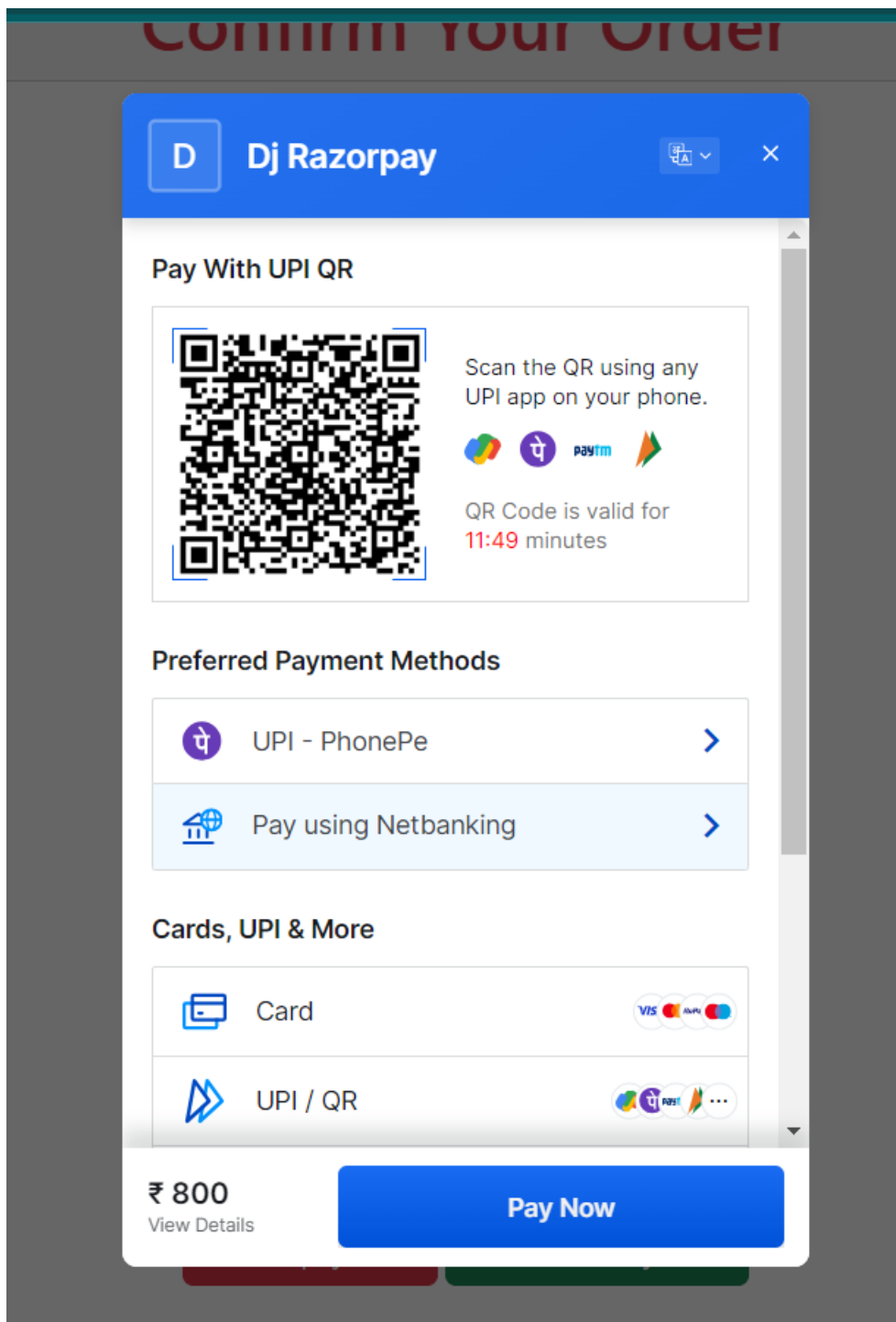


Fig 6: Payment Details Page

