MerryKid
Online Toy Shop Management System

Project Report

Submitted by

LINNU JOSE

Reg. No.: AJC22MCA-2058

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS

(MCA TWO YEAR)

(Accredited by NBA)

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. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2022-2024

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**MERRYKID"** is the bona fide work of **LINNU JOSE (Regno: AJC22MCA-2058)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Ms. Navyamol K T	**Ms. Meera Rose Mathew**
Internal Guide	**Coordinator**

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department **External Examiner**

// CERTIFICATE ON PLAGIARISM CHECK

DECLARATION

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

Date: 17/04/2024	**LINNU JOSE**
KANJIRAPPALLY	**Reg: AJC22MCA-2058**

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. Lillykutty 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 his 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. Navyamol K T** 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.

LINNU JOSE

ABSTRACT

.

The "Online Toy Shop Management System" project is a website designed to make buying toys online easy and enjoyable. It has a user-friendly interface, offering a curated selection of toys and children's products. The project is divided into four modules: user, sellers, admin and delivery agent. Unregistered users can browse the website, and there are login features for admin, sellers, and users. Admins manage the platform, while sellers add products. The main goal is to create versatile software for any toy company, using Python Django for the backend and HTML, CSS, and JavaScript for the frontend.

CONTENT

List of Abbreviations

- UML - Unified Modelling Language
- ORM - Object-Relational Mapping
- MVT - Model-View-Template
- MVC - Model-View-Controller
- RDBMS - Relational Database Management System
- 1NF - First Normal Form
- 2NF - Second Normal Form
- 3NF - Third Normal Form
- IDE - Integrated Development Environment
- HTML - HyperText Markup Language
- JS - JavaScript
- CSS - Cascading Style Sheets
- AJAX - Asynchronous JavaScript and XML
- JSON - JavaScript Object Notation
- API - Application Programming Interface
- UI - User Interface
- HTTP - Hypertext Transfer Protocol
- URL - Uniform Resource Locator
- SQL - Structured Query Language
- CRUD - Create, Read, Update, Delete

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The 'Online Toy Shop' project is a user-friendly website crafted to make toy shopping easy and enjoyable. Users can explore various categories of toys and access information about the toy shop. The platform is divided into four modules: user, sellers, admin, and delivery agents. Unregistered users have the option to browse the website, while admins oversee and manage the platform details. Sellers play a crucial role by adding their products to the catalog, and delivery agents ensure prompt and efficient delivery of purchased items. Login functionalities are incorporated for admin, sellers, delivery agents, and users, providing a secure and personalized experience. Admins have the authority to oversee and manage the overall platform, ensuring smooth operations. Sellers can efficiently add, delete, and edit their products within the website. The platform aims to redefine the digital toy shopping experience and provide advanced features for user engagement.

1.2 PROJECT SPECIFICATION

The proposed Online Toy Shop Management System is a comprehensive e-commerce platform tailored for the toy industry. This system revolutionizes the online toy shopping experience by offering a wide array of toys and children's products through a user-friendly interface. Users can effortlessly navigate the virtual catalog, adding their desired toys to the shopping cart through effective filtering options. The system introduces four key actors:

Admin:
The admin plays a pivotal role in overseeing the entire system. Admin privileges include managing the website's overall functionality, overseeing product listings, handling orders, and ensuring a seamless user experience.

Seller:
Registered sellers contribute to the platform by adding new products to the website. This enables a diverse and extensive catalog, enriching the options available for customers.

Customer:
Customers interact with the system through a user interface, where they can view their profiles, explore the catalog, and purchase toys based on different categories. The system supports online payments for a convenient transaction process. Additionally, customers can provide valuable feedback by writing reviews and rating the products, contributing to a dynamic and interactive

shopping environment.

Delivery Agent:

The delivery agent module ensures timely delivery of purchased items to customers' doorsteps. Delivery agents manage order fulfillment, tracking, and logistics, ensuring a smooth and efficient delivery process, enhancing the overall customer experience.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

In the context of the Online Toy Shop Management System, the process of system analysis involves a comprehensive approach to finding and analyzing data, diagnosing potential issues, and applying data for the enhancement of the system. This phase requires substantial communication between users and developers to troubleshoot effectively. System analysis is an integral part of the system development process, involving thorough research, efficient checking, and evaluation. Acting as investigators, system analysts crosscheck the functionality of the current toy shop system. To understand potential problems, they identify important variables, evaluate each element, and synthesize findings to propose optimal solutions. Utilizing methodologies like questionnaires and interviews, the toy shop system is extensively studied. The information gathered is carefully reviewed to draw meaningful conclusions and enhance awareness of the system's functioning. The current toy shop system undergoes meticulous evaluation to identify and address any existing problems. Proposed solutions are presented, and the most suitable one is chosen through analytical comparisons with the current system. Users are then given the opportunity to approve or reject the proposed suggestions. The preliminary study in the toy shop project involves interpreting and gathering essential facts, laying the foundation for subsequent studies and improvements to the system.

2.2 EXISTING SYSTEM

The current approach in the Online Toy Shop Management System involves data scattered across the system infrastructure, often requiring the use of numerous paper forms. Unfortunately, these forms frequently contain inaccuracies or deviate from management guidelines. There's a risk of records disappearing during computations, necessitating a stringent auditing system to prevent the loss of vital data. The online toy marketplace encompasses various services categorized under different sections, resulting in a lack of coordination in processing and limiting users from easily finding and booking relevant services. Similar to challenges faced in the beauty product industry, online toy shopping poses difficulties for customers who cannot physically assess or try out products before making a purchase.

2.2.1 NATURAL SYSTEM STUDIED

In the existing system of online toy shopping, customers are primarily engaged with individual toy sellers or other channels to make toy purchases. This process is often time-consuming and lacks a streamlined approach. The proposed Online Toy Shop Management System aims to address these drawbacks by offering a more efficient and user-friendly solution. This system empowers users to effortlessly browse and select from a diverse range of toys and accessories, providing a seamless and convenient shopping experience. By centralizing the toy purchasing process and enhancing accessibility, the proposed system seeks to overcome the limitations of the current approach, offering users the flexibility to explore and choose toys based on their specific preferences and requirements.

2.2.2 DESIGNED SYSTEM STUDIED

In the proposed online toy shop system, data management is seamlessly conducted through an online framework, ensuring the utmost safety, security, and ease of handling. This platform introduces an online booking system for toy-themed events and parties, along with a convenient avenue for purchasing a diverse array of toys. By allowing customers to schedule their toy-related appointments online, the system efficiently minimizes time wastage. Moreover, the platform incorporates exclusive discounts on various toy products, enabling customers to acquire items at affordable rates. To enhance the customer experience, a 'try-before-buy' option is provided, allowing users to explore selected toys before making a purchase decision. Following the successful completion of the payment process, users gain access to a comprehensive order tracking feature. Administrative functionalities within the system encompass the management of services, bookings, toy products, inventory, and customer interactions. This integrated approach aims to optimize the online toy shopping experience, ensuring it is secure, efficient, and enjoyable for both customers and administrators.

2.3 DRAWBACKS OF EXISTING SYSTEM

- The traditional toy shop offers a less interactive user environment.
- Customers are unable to physically try on toys before making a purchase.
- The usage of less user-friendly or traditional interfaces makes navigation challenging.
- There's an absence of proper online management for toy inventory.
- Customers may experience wastage of time due to conventional shopping processes.
- The process often requires significant human effort for various aspects of toy retail.
- Toy booking schedules are nonexistent in this traditional toy shop setup.

2.4 PROPOSED SYSTEM

The "Online Toy Stop," is an exciting web-based platform designed to transform the way people shop for toys online. It's packed with features to make the experience easy and enjoyable. The project has different parts. The Admin Section is for the people managing the site, with secure logins and a dashboard showing important information. It helps manage products, users, and orders efficiently, making packaging smoother. The Users Section is for customers. It lets them create accounts, log in securely, manage their shopping carts.Users can also update their profiles, write reviews, and easily make payments.

There's a special section for Toy Brands, showcasing different brands, their products, and rankings. This gives users a curated selection of popular toy brands, making their shopping experience even better. The project also includes a Packing Management Section to handle products accurately before shipping. By putting all these parts together, the "Online Toy Store" aims to be a complete platform, serving both the people running the site and the customers. The goal is to take online toy shopping to a whole new level.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- Reduced Expenses
- Purchase can be done any time
- Detailed product information
- Efficient search options
- Efficient Order Management

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A feasibility study is a preliminary examination of a prospective project or end to determine its merits and viability. A feasibility study aims to provide an objective assessment of the technical, economic, financial, legal, and environmental elements of a proposed project. The information can then be used by decisionmakers to decide whether to proceed with the project or not. The findings of the feasibility study can also be used to develop a practical project plan and budget. It cannot be simple to determine whether or not a proposed project is worthwhile pursuing without a feasibility study. The document provides the feasibility of the project that is being designed and lists. Various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibility. The following are its features:

3.1.1 Economical Feasibility

The economic feasibility analysis is a crucial process in determining the worth of a new project in terms of cost and time investment. It involves a thorough analysis of all factors that can influence the success of the initiative. The proposed system, Online Toy shop, has undergone cost-benefit analysis and is found to be feasible and economical within the pre-assumed cost of this project. Cost and benefit analyses are required to support the developing system. criteria to make sure that focus is on the project that will yield the best results and return the earliest. The price that would be involved in developing a new system is one of the variables. Some significant financial queries raised during the initial investigation include the following:

- The costs conduct a full system investigation?
  The proposed system is developed as part of project work, there is no manual cost to spend for the proposed system.
- The cost of the hardware and software?
  Also all the resources are already available.

3.1.2 Technical Feasibility

The system needs to be assessed first from a technical standpoint. The outline design of the system requirement in terms of input, output, programs, and procedures must serve as the foundation for the assessment of this feasibility. After determining an outline investigation must continue to identify the necessary equipment kind. Once the system has been designed, there are several ways to run it. Technical issues raised during the investigation are:

1. Is the project feasible within the limits of current technology?

Satisfied

2. Can the technology be easily applied to current problems?

   Satisfied

3. Does the technology have the capacity to handle the solution?

   Satisfied

3.1.3 Behavioral Feasibility

To ensure the success of this system, we carefully addressed two key questions:

- Is there sufficient user support?

  Satisfied

- Will the system cause any harm?

  No

Thorough evaluation of these questions was conducted to guarantee the system's positive impact upon implementation. The feasibility study also meticulously considered all behavioral factors, ensuring that the project is behaviorally feasible. Overall, we anticipate that the proposed system will successfully achieve its objectives with a high level of success.

3.1.4 Feasibility Study Questionnaire

1. Project Overview

The Online Toy Shop Management System is a comprehensive e-commerce platform designed to provide a convenient and enjoyable online shopping experience for customers seeking a wide range of toys and children's products. This system aims to bridge the gap between traditional toy stores and modern online shopping, offering a diverse catalog, secure transactions, and user-friendly features.

2. To what extend the system is proposed for?

The proposed Online Toy Shop Management System is designed to be a comprehensive and user-centric platform, spanning from an intuitive shopping experience for users of all ages to robust security measures, extensive product listings, and empowerment for independent sellers. It aims to streamline order processing, foster a dynamic feedback loop, and provide administrative control. The system's extent encompasses enhancing customer satisfaction, seller growth, and overall efficiency in the online toy shopping process, bridging the gap between traditional toy stores and modern e-commerce.

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

Customers: These are the primary users of the system who visit the platform to browse, search, and purchase toys and children's products. Customers interact with the system to explore the product catalog, add items to their shopping carts, make payments, and track their orders.

Independent Toy Sellers: sellers utilize the platform to register, list their products, manage their profiles, and engage with customers. They are an essential part of the system as they contribute to the diversity of products available.

Administrators: Administrators are responsible for overseeing and managing the entire system. They manage user accounts, product listings, order processing, and overall system operations. They also use the system for monitoring and reporting purposes.

Delivery Agents: Delivery agents interact with the system to manage delivery schedules, routes, and order deliveries. They are responsible for ensuring that products are safely and timely delivered to customers.

4. List the Modules included in your System?

1. Admin
2. Customer
3. Seller
4. Delivery Agent

5. Identify the users in your project?

Online Toy Shop Management System, users can be identified based on their roles and responsibilities within the platform.

I. Customers:

visit the platform to browse, select, and purchase toys and children's products for personal use or as gifts. Customers create accounts to personalize their shopping experience, track orders, and provide feedback.

2. Independent Toy Sellers:

They register as sellers, create seller profiles, manage product listings, fulfill orders, and engage with customers.

3. Admin:

Administrators are responsible for managing and overseeing the entire Online Toy Shop Management System. They have the highest level of access and control, managing user accounts, product listings, order processing, and overall system operations.

6. Who owns the system?

Administrator

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

The Online Toy Shop Management System is directly related to the e-commerce operations of a firm or organization that specializes in the retail of toys and children's products. This comprehensive digital platform is tailored to meet the needs of such a firm or organization, facilitating a diverse product catalog, secure transactions, and an improved online shopping experience for customers in the toy industry.

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

- Using different web resources
- Personal Experience

Questionnaire to collect details about the project?

1. **What is the primary goal of the Online Toy Shop Management System?**

   The primary goal of the Online Toy Shop Management System is to provide a comprehensive and user-centric e-commerce platform that offers a seamless and secure online shopping experience for customers seeking a wide range of toys and children's products.
2. **How does the system plan to bridge the gap between traditional toy stores and modern online shopping?**

   The system aims to bridge this gap by offering a diverse catalog, secure transactions, and user-friendly features. It provides a convenient and enjoyable online shopping experience while empowering independent toy sellers to showcase their products.

3. **What are the key features and functionalities that the system intends to offer to customers and administrators?**

   Some key features include user authentication and security, an extensive product catalog with categories, a shopping cart, multiple payment options, product reviews, and secure order processing and tracking.

4. **What are the core objectives of the project concerning user interface and user experience design?**

   The core objectives include creating an intuitive user interface, ensuring a seamless shopping experience, and facilitating user satisfaction for users of all ages.

5. **How does the system plan to empower independent toy sellers to showcase their products effectively?**

   Independent toy sellers can manage their product listings, interact with customers, and process orders within the system. Specific details on seller empowerment would be helpful.

6. **What technologies and tools are planned for the development of the Online Toy Shop Management System?**

   The technologies and tools planned for development include Django as the backend framework, HTML/CSS for the frontend, Python, JavaScript, a database management system (e.g., PostgreSQL or MySQL), version control tools like Git, and security measures such as encryption (HTTPS).

7. **How does the system plan to gather feedback and reviews from customers, and how will this feedback be used**?

   The system includes a feedback and review mechanism, but additional information on how customer feedback will be collected and utilized for improvements is needed.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	-	intel core i3
RAM	-	8 GB
Hard Disk	-	1 TB

3.2.2 Software Specification

Front End	-	HTML, JS, CSS, jQuery, Ajax, Bootstrap
Back End	-	Python-Django
Database	-	Sqlite
Client on PC	-	Windows 11
Technologies used	-	JS, HTML5, AJAX, jQuery, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 PYTHON DJANGO

Django is a popular and powerful open-source web framework written in Python, designed to facilitate rapid development and maintainable web applications. It follows the Model-View-Template (MVT) architectural pattern, which is similar to the Model-View-Controller (MVC) pattern. Django provides a structured and efficient way to build web applications, offering several key components and features.

At its core, Django includes a robust Object-Relational Mapping (ORM) system that simplifies database interactions, allowing developers to work with Python objects instead of raw SQL queries. It also includes a URL dispatcher for mapping URLs to view functions, an automatic admin interface for managing application data, and a templating engine for creating dynamic and reusable user interfaces.

Django places a strong emphasis on security, with built-in features to protect against common web vulnerabilities. It offers authentication and authorization systems, middleware support for global request and response processing, and compatibility with various databases.

The framework's scalability, extensibility, and a vibrant community of developers make it a popular choice for building web applications, from simple websites to complex, high-traffic platforms. Django's extensive documentation and ecosystem of third-party packages further enhance its appeal for web developers.

3.3.2 SQLITE

SQLite is a self-contained and serverless relational database management system (RDBMS) known for its simplicity and efficiency. It stores the entire database in a single file, eliminating the need for a separate server process, which simplifies deployment. This makes SQLite an ideal choice for embedded systems, mobile applications, and small to medium-sized projects. Developers interact with the database directly through function calls, making it an embedded database well-suited for various applications, including mobile apps, desktop software, and web applications.

Despite its lightweight nature, SQLite is ACID-compliant, ensuring data integrity with support for transactions. It offers a wide range of data types, and its compatibility with multiple programming languages, including Python, C/C++, and Java, makes it accessible to a diverse developer community. SQLite is open-source, free, and known for its speed and efficiency, particularly for read-heavy workloads. While not intended for extremely high-concurrency or large-scale applications, SQLite excels in scenarios where simplicity, portability, and low resource consumption are key requirements.

It is commonly used as a local data store, cache, or embedded database within larger applications, contributing to its versatility and widespread adoption in software development.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to effective system. The term "design" is defined as "the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization". It may be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance and accuracy levels. The design phase is a transition from a user oriented document to a document to the programmers or database personnel. System design goes through two phases of development: Logical and Physical Design.

4.2 UML DIAGRAM

UML, or Unified Modeling Language, is a standardized language used for specifying, visualizing, constructing, and documenting software system artifacts. Developed by the Object Management Group (OMG), the UML 1.0 specification draft was proposed in January 1997. UML stands out from common programming languages like C++, Java, or COBOL. it is a visual modeling language utilized for creating software blueprints. UML serves as a versatile and general-purpose visual modeling language, applicable not only to software systems but also to non-software systems, such as manufacturing unit process flows. While UML itself is not a programming language, it can be leveraged with tools to generate code in various programming languages based on UML diagrams. UML is closely linked with object-oriented analysis and design and has become an OMG standard after standardization.

The heart of UML lies in its visual diagrams, where elements and relationships come together to represent a complete system. The visual impact of UML diagrams is crucial, and various elements contribute to its completeness. UML encompasses nine key diagrams, each serving a specific purpose:

- Class Diagram
- Object Diagram
- Use Case Diagram
- Sequence Diagram

- Activity Diagram
- State Chart Diagram
- Deployment Diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A use case diagram is a visual representation illustrating how users and external entities interact with the internal components of a system. Its primary purpose is to identify, outline, and organize the functional needs of a system from the perspective of its users. The Unified Modeling Language (UML), a standard language for modeling real-world objects and systems, is commonly employed in constructing use case diagrams. Use cases serve various system objectives, including defining basic requirements, validating hardware designs, testing and debugging programs, creating online help references, or fulfilling customer support duties. Examples of use cases in the context of product sales include customer support, product purchasing, catalog updating, and payment processing.

A use case diagram consists of system boundaries, actors, use cases, and their interconnections. The system boundary defines the system's limits concerning its environment. Actors are entities defined by the roles they play, representing individuals or systems interacting with the system. Use cases specify the precise actions or behaviors that actors perform within or around the system. The diagram visually displays connections between actors and use cases, along with the use cases themselves.

When creating a use case diagram, it's crucial to adhere to specific guidelines for efficiency and effectiveness:

1. Choose descriptive names for use cases that accurately reflect their functionalities.
2. Assign appropriate names to actors for clear identification of their roles.
3. Clearly depict relationships and dependencies in the diagram.
4. Avoid including every possible relationship, focusing on essential requirements.
5. Use notes when necessary to clarify important points.

Following these guidelines ensures the creation of a clear and concise use case diagram that accurately represents the functional requirements of the system.

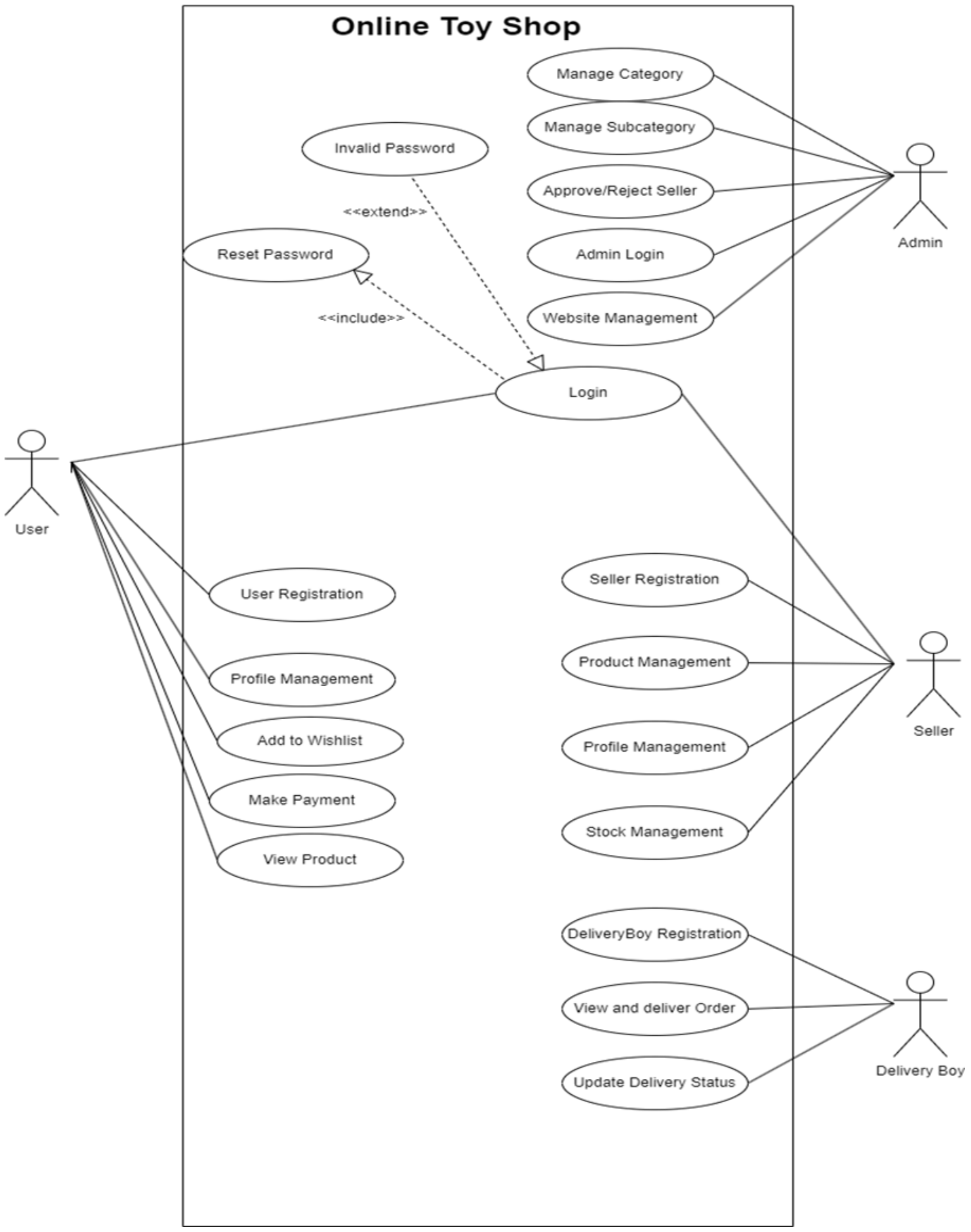


Fig. 4.2.1 Use case diagram for Online Toy Shop

4.2.2 SEQUENCE DIAGRAM

A sequence diagram, a type of interaction diagram, illustrates the chronological order of interactions among various system components. It provides a visual representation of how different entities communicate with each other through a series of messages. Also known as event scenarios or event scenario diagrams, sequence diagrams are commonly used in software engineering to describe and comprehend the requirements of both new and existing systems. They facilitate the visualization of object control relationships and aid in detecting systemic issues.

Sequence Diagram Notations:

i. Actors: In UML, actors represent roles that interact with the system and its objects. Actors often exist outside the system being portrayed in the UML diagram and can play various roles, including external entities or human users. Actors are symbolized by stick figures in UML diagrams, and a sequence diagram may have more than one actor depending on the modeled situation.

ii. Lifelines: Lifelines in a sequence diagram are vertical dashed lines representing the lifespan of an object participating in the interaction. Each lifeline corresponds to an individual participant, labeled with their name. The lifeline visually shows the timeline of events for the participant, extending from activation to deactivation points.

iii. Messages: Messages are fundamental to sequence diagrams, depicting interactions and communications between objects or components in a system. They include synchronous and asynchronous messages, create and delete messages, self-messages, reply messages, found messages, and lost messages. Guards are used to model conditions and restrictions on message flow.

iv. Guards: Guards in UML represent conditions that restrict the flow of messages when specific conditions are met. They are crucial for informing software developers about constraints or limitations associated with a system or a particular process.

Uses of Sequence Diagrams:

- Modeling and visualizing the logic of complex functions, operations, or procedures.
- Detailing UML use case diagrams.

- Understanding the detailed functionality of current or future systems.
- Visualizing how messages and tasks move between objects or components in a system.

Overall, sequence diagrams are useful for representing the flow of interactions between objects in a system, aiding both businesspeople and software engineers in better understanding and communicating system requirements and behavior.

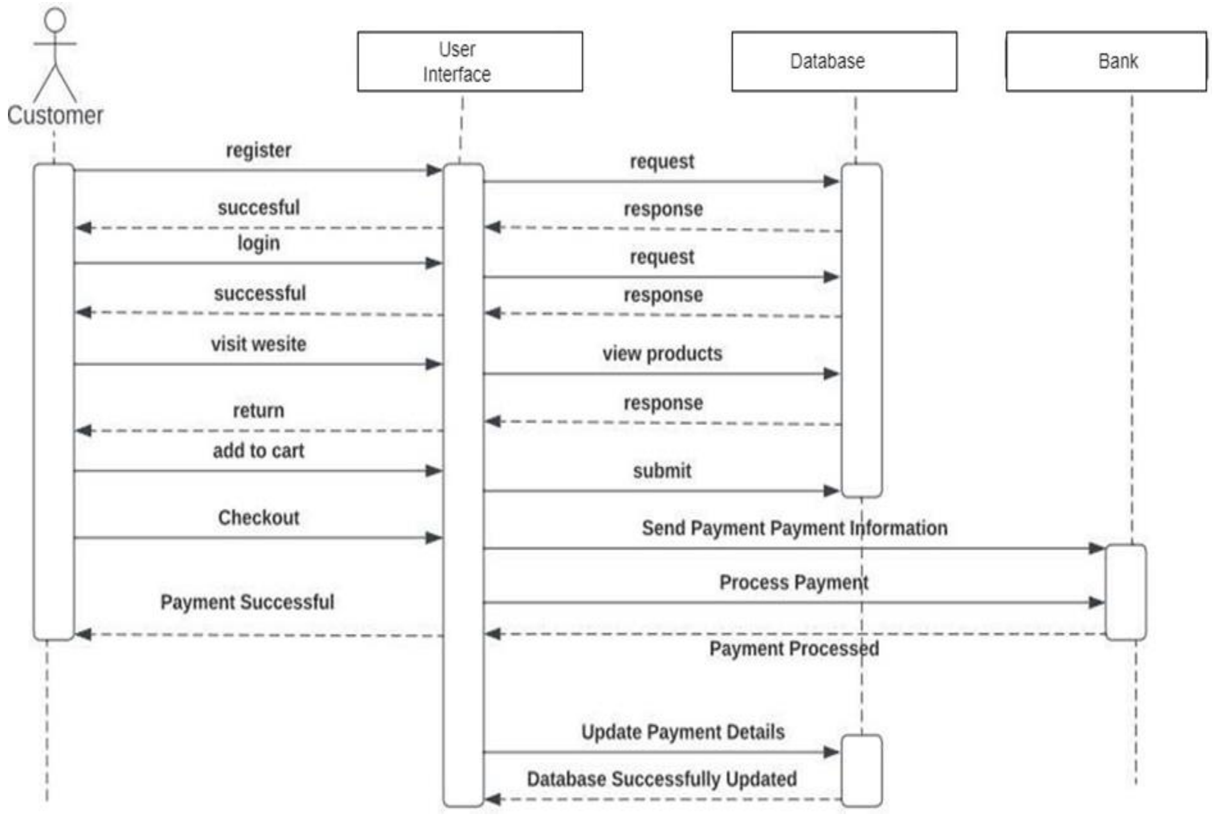


Fig. 4.2.2 Sequence diagram for Online Toy Shop

4.2.3 State Chart Diagram

A state diagram, often created using the Unified Modeling Language (UML), is a visual representation that illustrates the various states an object can exist in and the transitions between those states. This diagram is alternatively known as a state machine diagram or state chart diagram. The State Chart Diagram, a behavioral diagram in UML, captures the behavior of a system or object across different points in time. Key elements in a State Chart Diagram include:

- Initial State: Represented by a solid black circle, this state signifies the starting point of the system or object.

- State: Describes the current condition of the system or object at a specific moment, denoted by a rectangle with rounded corners.

- Transition: Depicts the movement of the system or object from one state to another, represented by an arrow.

- Event and Action: An event acts as a trigger causing a transition, and an action represents the behavior or effect of the transition.

- Signal: A message or trigger resulting from an event that is sent to a state, leading to a transition.

- Final State: The diagram concludes with a Final State element, depicted by a solid black circle with a dot inside. It indicates the completion of the system or object's behavior.

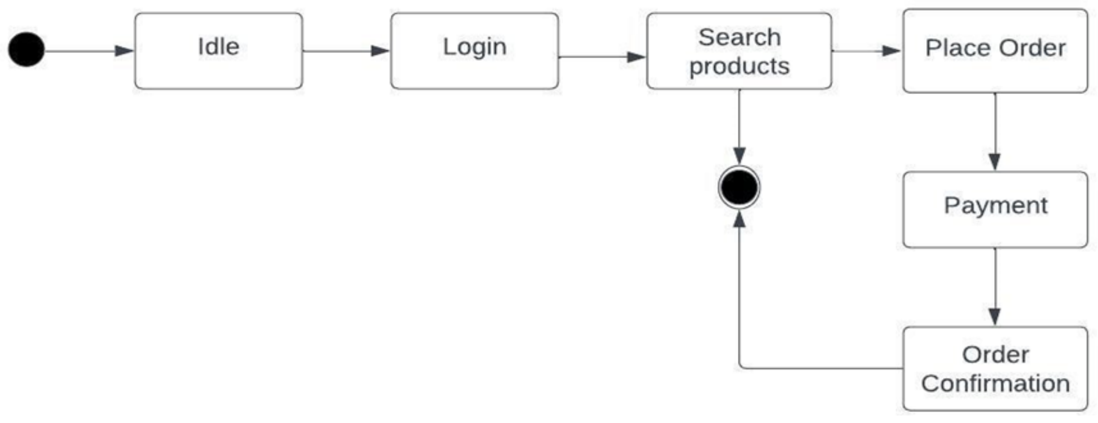


Fig.4.2.3 State Chart Diagram for Online Toy Shop

4.2.4 Activity Diagram

An activity diagram serves as a visual representation of a workflow, illustrating how one activity leads to another within a system. Activities, referred to as system operations, are linked in a control flow, where the flow can be parallel, concurrent, or branched. Activity diagrams employ various functions such as branching and joining to manage different types of flow control. These diagrams fall under the category of behavior diagrams and provide insights into the dynamic behavior of a system.

Key Components of an Activity Diagram:

i. Initial Node: Denoted by a black circle, it marks the starting point of the activity diagram.

ii. Activity: Represents a task or action performed by the system or entity, depicted by a rectangle with rounded corners.

iii. Control Flow: Represents the sequence of activities or actions, visualized by an arrow.

iv. Decision Node: A point of decision or branching in the activity flow, identified by a diamond shape.

v. Merge Node: Merges multiple branches of the activity flow into a single flow, indicated by a diamond shape with a plus sign.

vi. Fork Node: Splits the activity flow into multiple parallel flows, symbolized by a solid black circle with multiple arrows.

vii. Join Node: Joins multiple parallel flows back into a single flow, represented by a solid black circle with arrows pointing towards it.

viii. Final Node: Marks the endpoint of the activity diagram, recognized by a black circle with a dot inside.

ix. Object Flow: Represents the flow of objects or data between activities, shown by a dashed arrow.

Activity diagrams are valuable for clarifying complex processes, identifying potential issues, and communicating process flows to stakeholders and project team members.

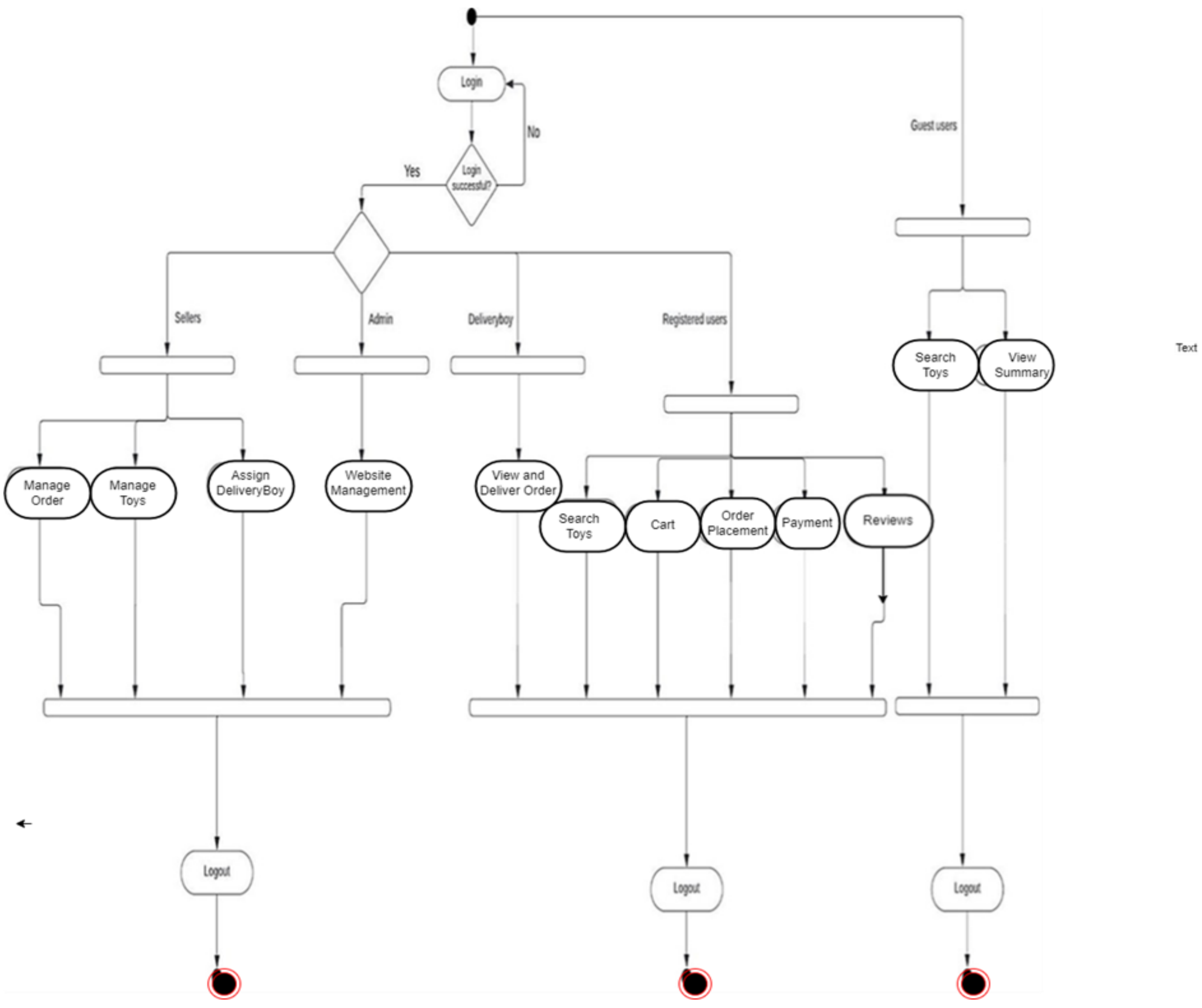


Fig.4.2.4 Activity Diagram for Online Toy Shop

4.2.5 Class Diagram

Static diagrams, like class diagrams, provide a representation of the static view of an application. Class diagrams are utilized in both creating executable software programs and visually depicting, describing, and managing various system components. These diagrams outline the attributes, capabilities, and constraints of classes within a system. Class diagrams hold a prominent role in the design of object-oriented systems, as they can be readily translated into entity languages.

In a class diagram, a visual representation of a collection of classes, interfaces, associations, collaborations, relationships, and constraints is presented. Class diagrams are often referred to as structural diagrams, as they focus on the structure and static aspects of a system.

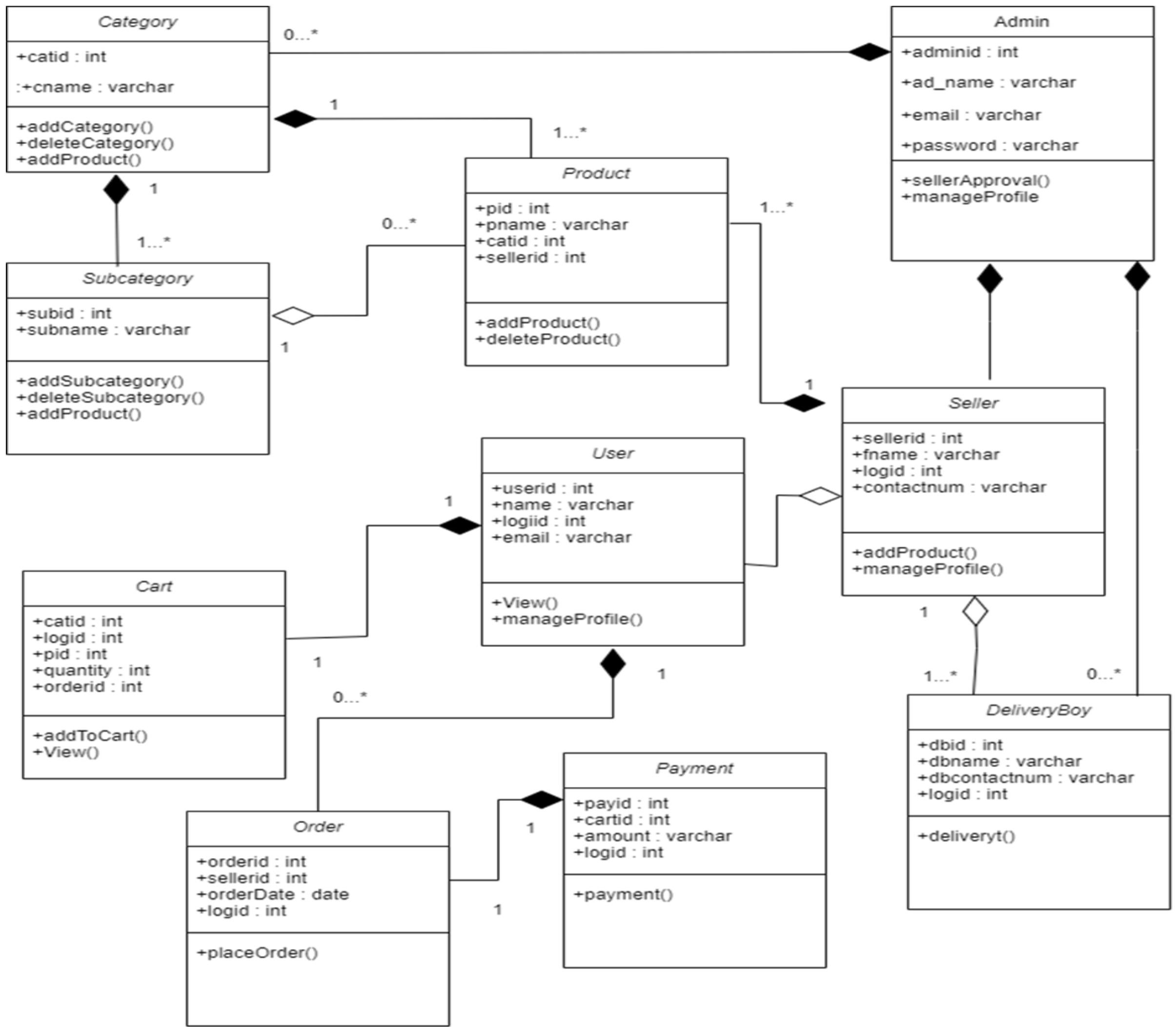


Fig.4.2.5 Class Diagram for Online Toy Shop

4.2.6 Object Diagram

Class diagrams and object diagrams are integral components of the Unified Modeling Language (UML), each serving distinct yet interconnected purposes in system modeling. The relationship between these two diagrams is symbiotic, with class diagrams providing the foundational structure of a system and object diagrams offering concrete instances derived from those structures. Object diagrams essentially capture a specific moment or scenario within a system, showcasing instances of classes and their relationships. They act as visual snapshots, offering a detailed and tangible representation of the static view of a system. While class diagrams establish the blueprint and core concepts, object diagrams bring these concepts to life, allowing stakeholders to visualize real-world scenarios and understand the relationships among instantiated objects within an object-oriented system. Together, class diagrams and object diagrams provide a holistic perspective, combining the abstract and the concrete to enhance comprehension and communication in system design and development.

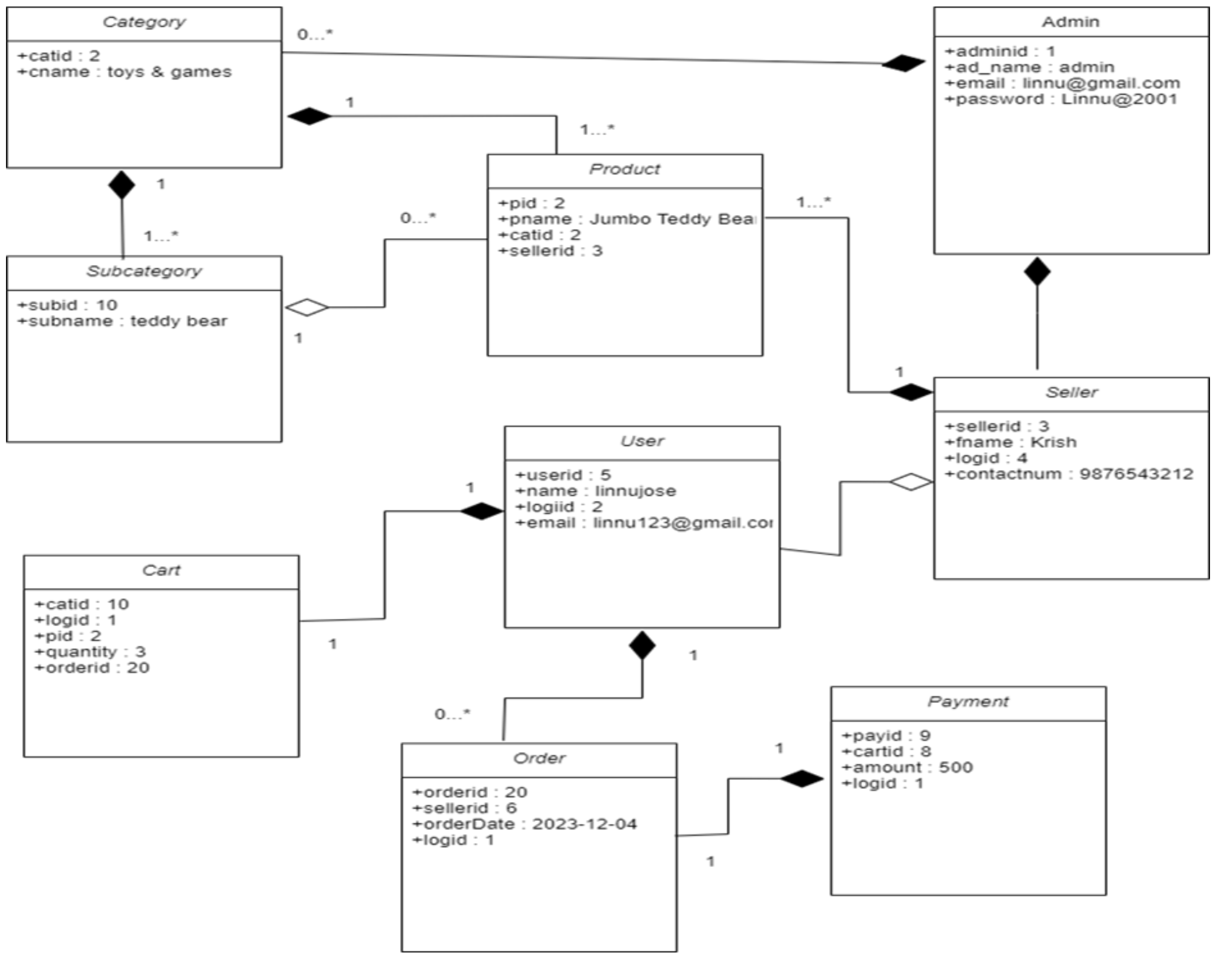


Fig.4.2.6 Object Diagram for Online Toy Shop

4.2.7 Component Diagram

A component diagram in UML illustrates how various components are interconnected to create larger components or software systems. It is an effective tool for representing the structure of complex systems with multiple components. By using component diagrams, developers can easily visualize the internal structure of a software system and understand how different components work together to accomplish a specific task.

Its key components include:

- Component: A modular and encapsulated unit of functionality in a system that offers interfaces to interact with other components. It is represented as a rectangle with the component name inside.
- Interface: A contract between a component and its environment or other components, specifying a set of methods that can be used by other components. It is represented as a circle with the interface name inside.
- Port: A point of interaction between a component and its environment or other components.It is represented as a small square on the boundary of a component.
- Connector: A link between two components that enables communication or data exchange.It is represented as a line with optional adornments and labels.
- Dependency: A relationship between two components where one component depends on another for its implementation or functionality. It is represented as a dashed line with an arrowhead pointing from the dependent component to the independent component.
- Association: A relationship between two components that represents a connection or link. It is represented as a line connecting two components with optional directionality, multiplicity, and role names.
- Provided/Required Interface: A provided interface is an interface that a component offers to other components, while a required interface is an interface that a component needs fromother components to function properly. These are represented by lollipops and half-circles respectively.

Component diagrams are useful for modeling the architecture of a software system, and can help identify potential issues and improvements in the design. They can also be used to communicate the structure and behavior of a system to stakeholders, such as developers and project managers.

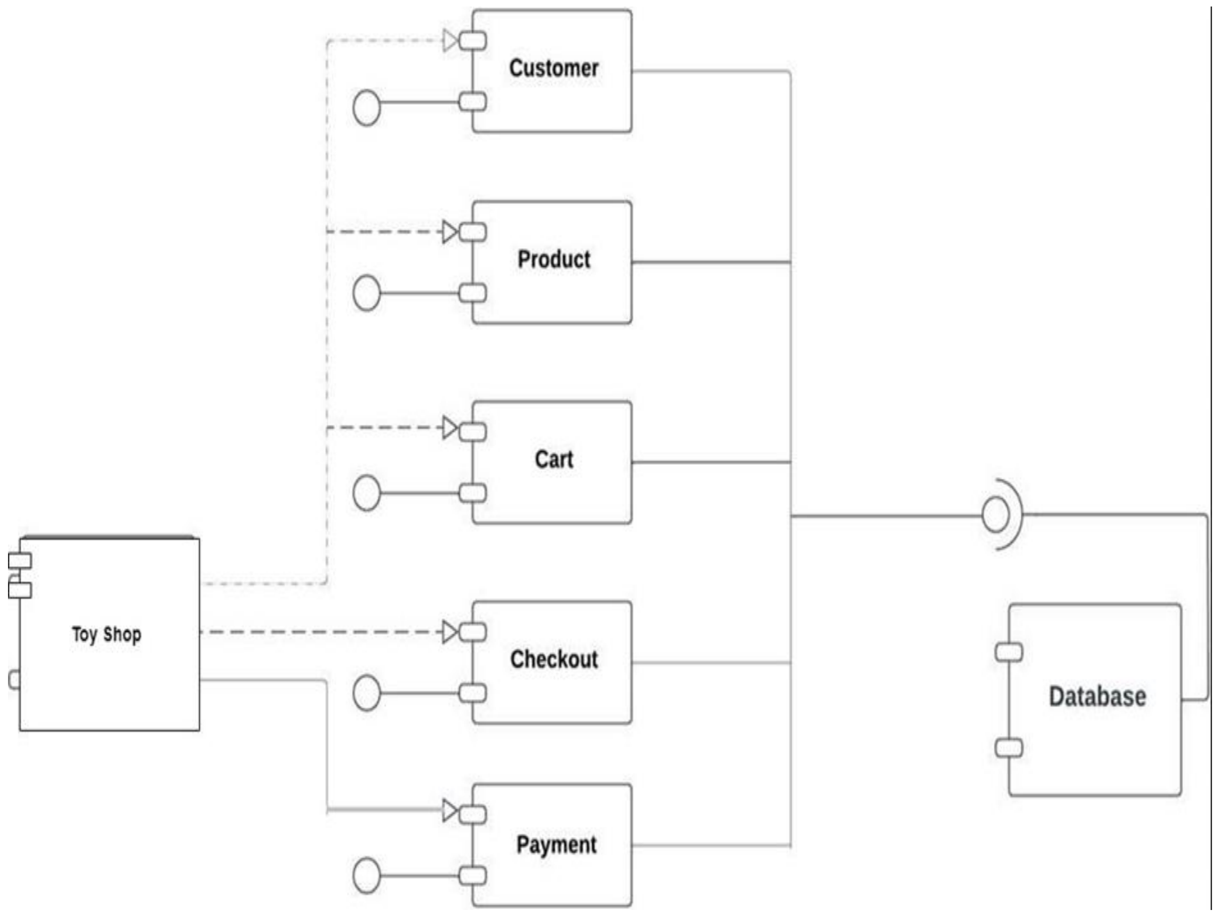


Fig.4.2.7 Component Diagram for Online Toy Shop

4.2.8 Deployment Diagram

A deployment diagram, a type of UML diagram, illustrates the execution architecture of a system, which consists of nodes such as hardware or software execution environments and the middleware connecting them. Deployment diagrams are frequently used to show a system's actual hardware and software. You can understand how the hardware will actually provide the system by utilising it. Deployment diagrams help represent the physical structure of a system, in contrast to other UML diagram types that primarily show the logical components of a system.

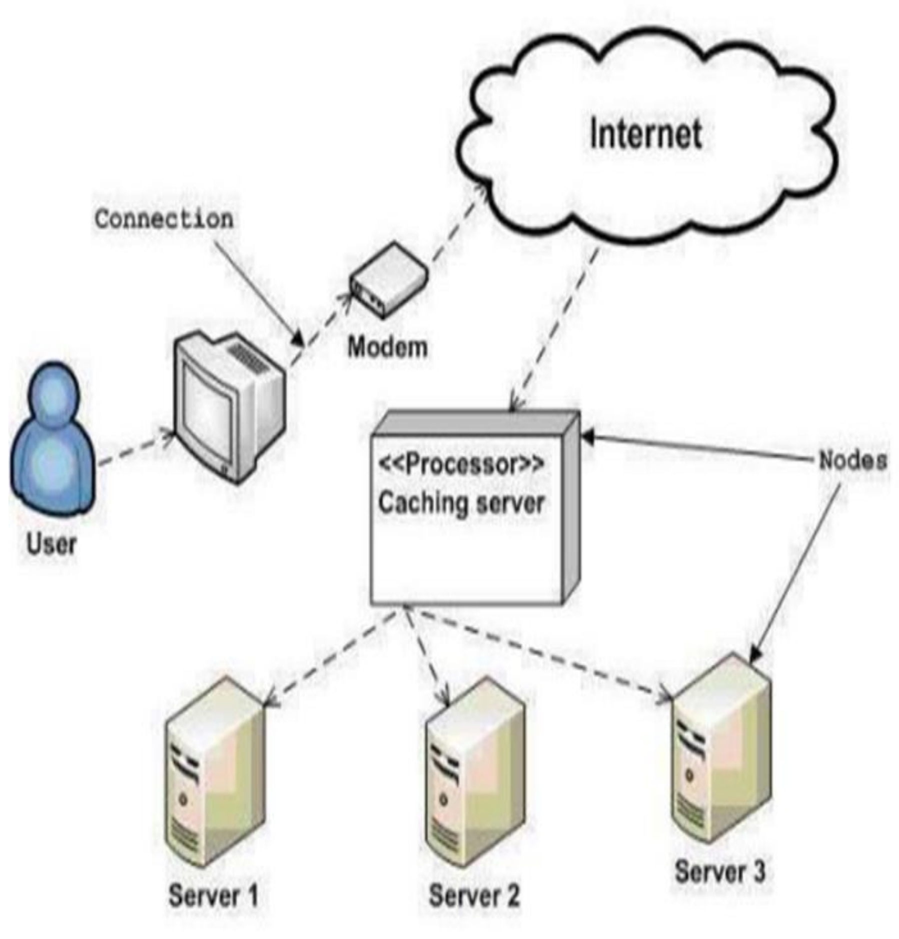
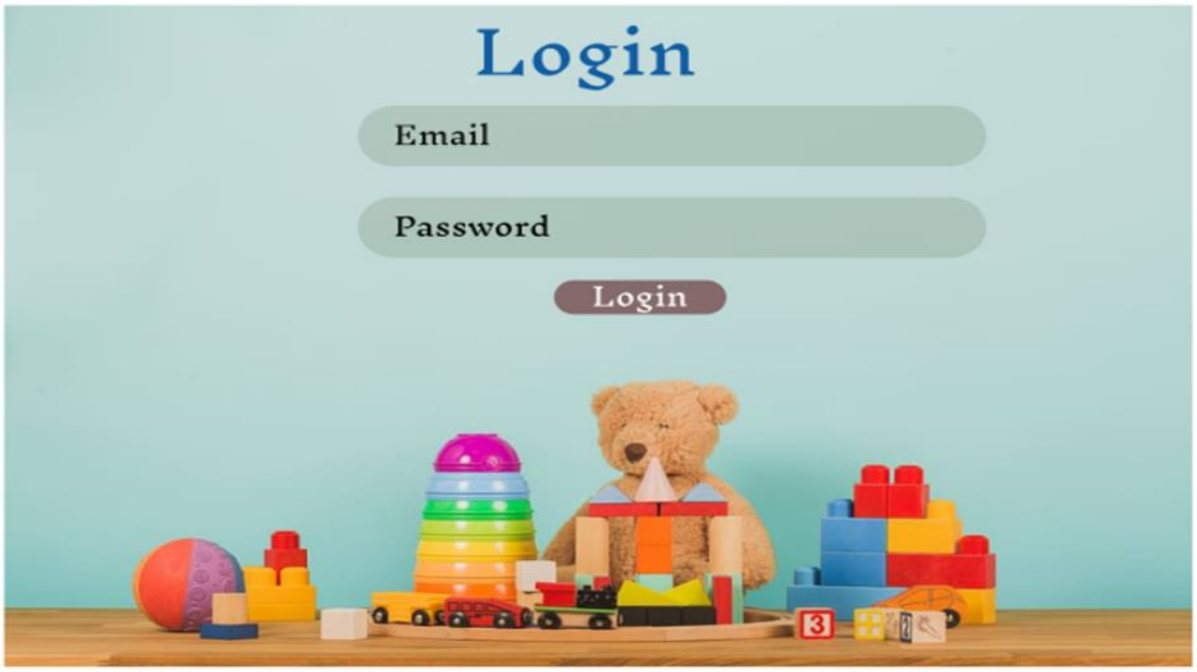


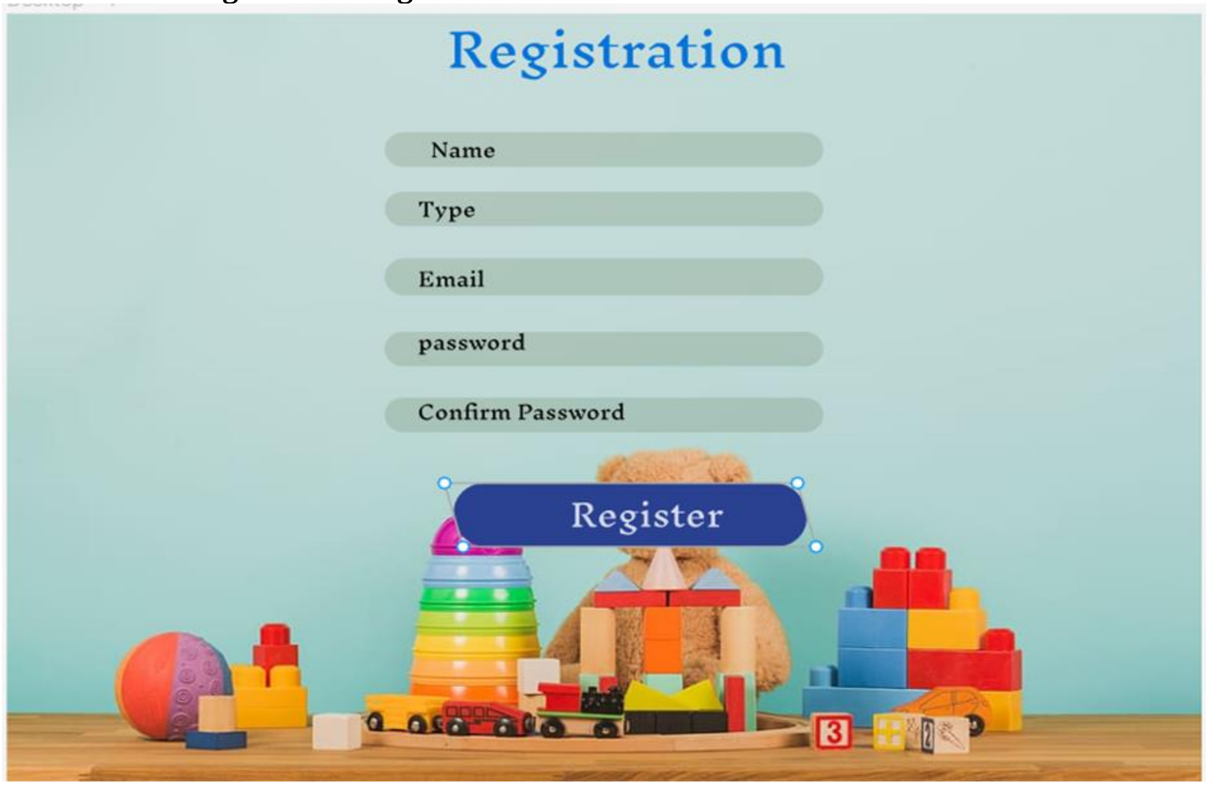
Fig.4.2.8 Deployment Diagram for Online Toy Shop

4.3 USER INTERFACE DESIGN USING FIGMA

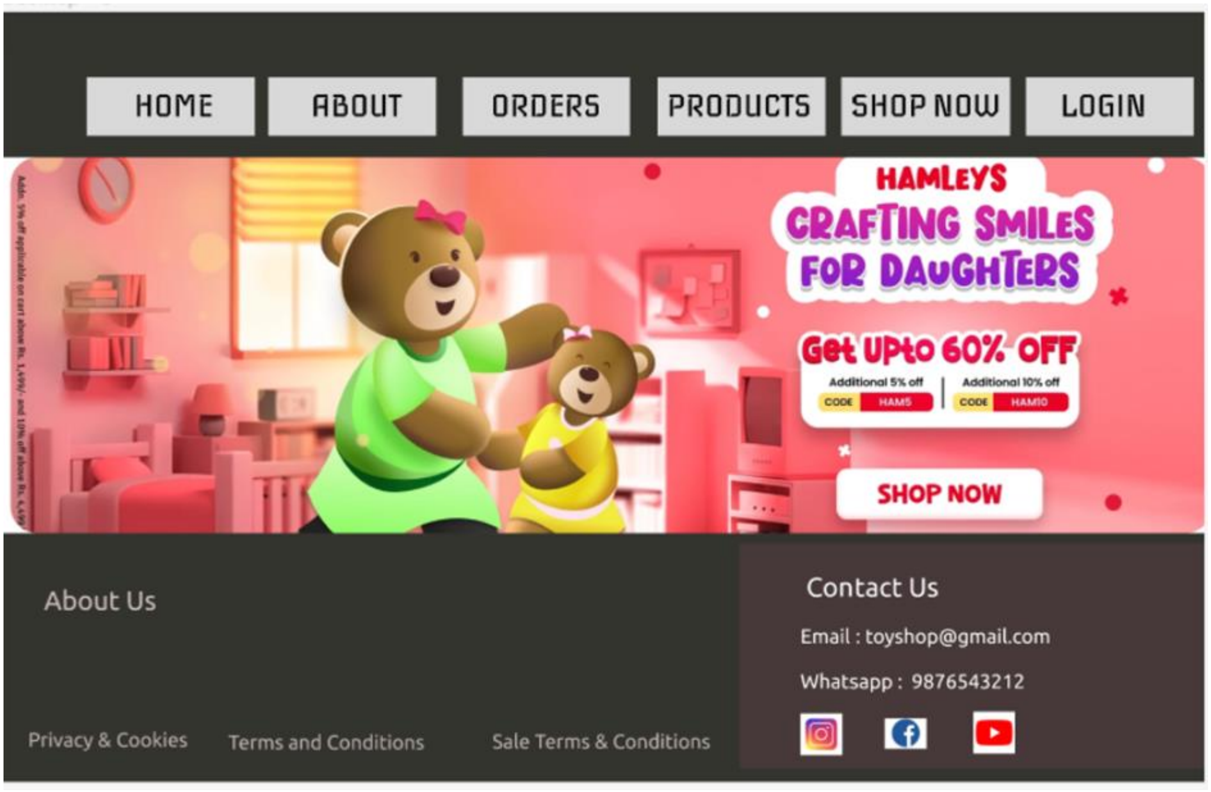
Form Name: Login Page



Form Name: Registration Page



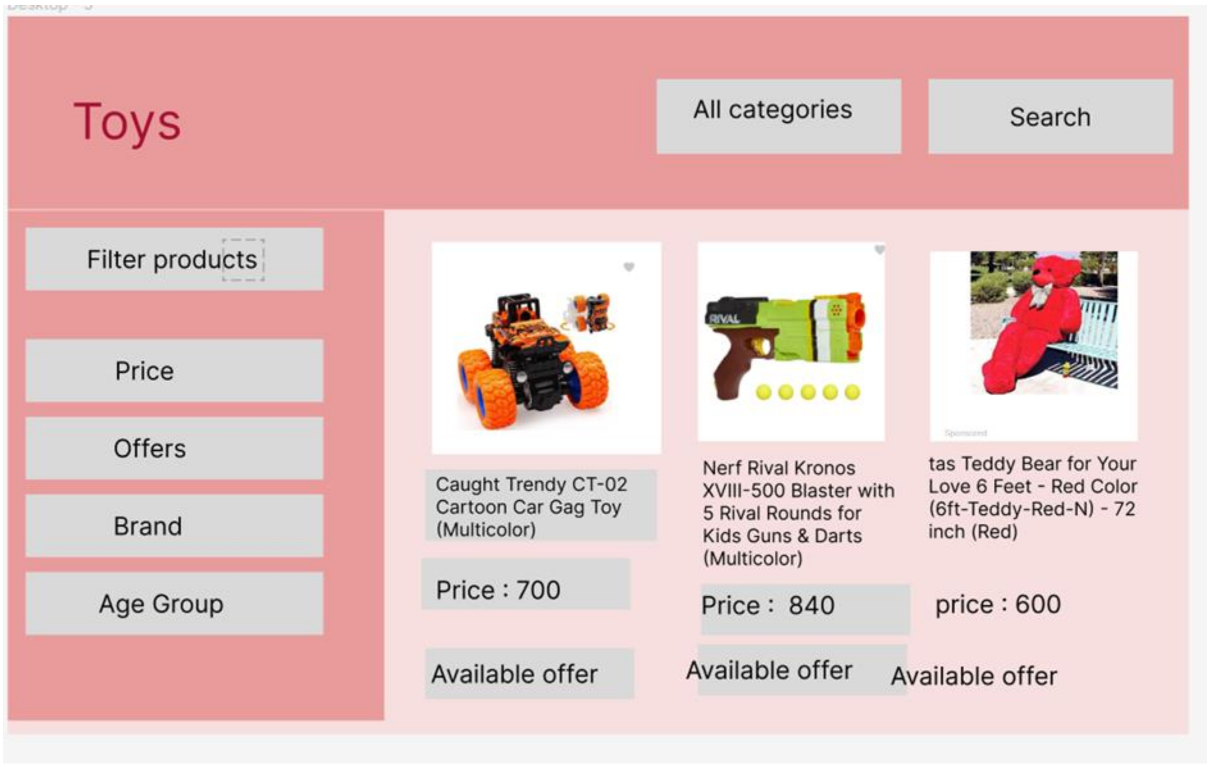
Form Name: Home Page



Email : toyshop@gmail.com

Whatsapp : 9876543212

Form Name: Category Listing Page



Caught Trendy CT-02 Cartoon Car Gag Toy (Multicolor)

Price : 700

Available offer

Nerf Rival Kronos XVIII-500 Blaster with 5 Rival Rounds for Kids Guns & Darts (Multicolor)

Price : 840

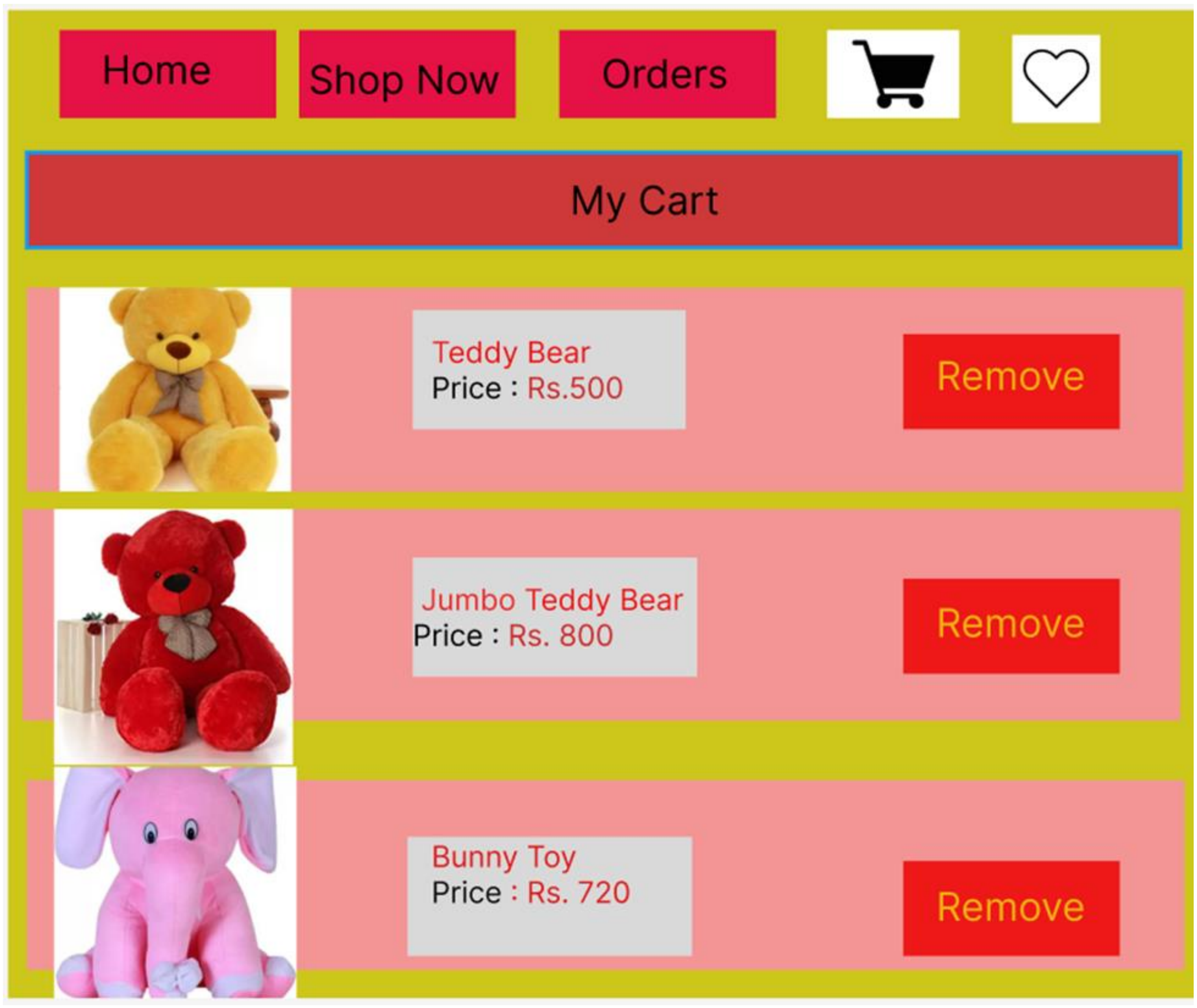
Available offer

tas Teddy Bear for Your Love 6 Feet - Red Color (6ft-Teddy-Red-N) - 72 inch (Red)

price : 600

Available offer

Form Name: Cart Page

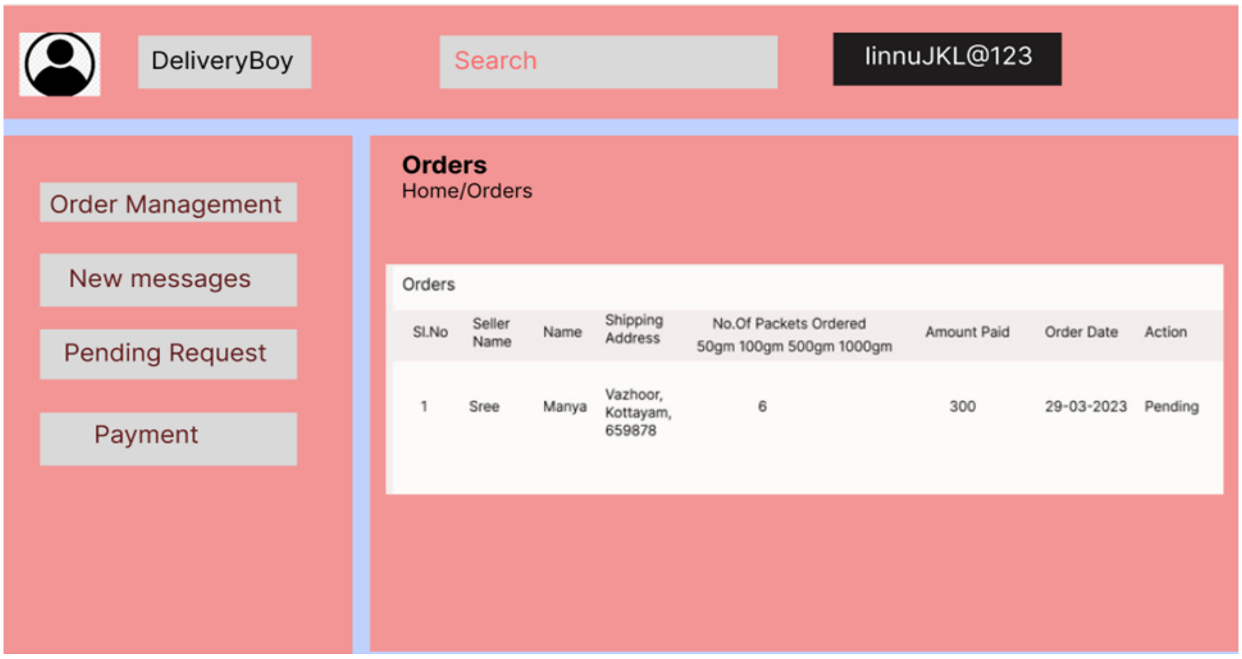


Teddy Bear
Price : Rs.500

Jumbo Teddy Bear
Price : Rs. 800

Bunny Toy
Price : Rs. 720

Form Name: Delivery Boy Home Page



Orders
Home/Orders

Orders

Sl.No	Seller Name	Name	Shipping Address	No.Of Packets Ordered 50gm 100gm 500gm 1000gm	Amount Paid	Order Date	Action
1	Sree	Manya	Vazhoor, Kottayam, 659878	6	300	29-03-2023	Pending

4.3 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 a 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 popular type of database that organizes data into tables to facilitate relationships with other stored data sets. Tables can contain vast amounts of data, ranging from hundreds to millions of rows, each of which are referred to as records. In formal relational model language, a row is called a tuple, a column heading is an attribute, and the table is a relation. A relational database consists of multiple tables, each with itsown name. Each row in a table represents a set of related values.

In a relational database, relationships are already established between tables to ensure the integrity of both referential and entity relationships. A domain D is a group of atomic values, and a common way to define a domain is by choosing a data type from which the domain's data values are derived. It is helpful to give the domain a name to make it easier to understand the values it contains. Each value in a relation is atomic and cannot be further divided.

In a relational database, table relationships are established using keys, with primary key and foreign key being the two most important ones. Entity integrity and referential integrity relationships can be established with these keys. Entity integrity ensures that no primary key can have null values, while referential integrity ensures that each distinct foreign key value must have a matching primary key value in the same domain. Additionally, there are other types of keys such as super keys and candidate keys.

4.4.2 Normalization

Data are grouped together in the simplest way so that later changes can be made with minimum impact on data structures. Normalization is formal process of data structures in manners that eliminates redundancy and promotes integrity. Normalization is a technique of separating redundant fields and breaking up a large table into a smaller one. It is also used to avoid insertion, deletion, and updating anomalies. Normal form in data modelling use two concepts, keys and relationships. A key uniquely identifies a row in a table.There are two types of keys, primary key and foreign key. A primary key is an element or a combination of elements in a table whose purpose is to identify records from the same table. A foreign key is a column in a table that uniquely identifies record from a different table. All the tables have been normalized up to the third normal form.

As the name implies, it denotes putting things in the normal form. The application developer via normalization tries to achieve a sensible organization of data into proper tables and columns and where names can be easily correlated to the data by the user. Normalization eliminates repeating groups at data and thereby avoids data redundancy which proves to be a great burden on the computer resources. These include:

- ✓ Normalize the data.
- ✓ Choose proper names for the tables and columns.
- ✓ Choose the proper name for the data.

First Normal Form

The First Normal Form (1NF) requires that each attribute in a table must contain only atomic or indivisible values. It prohibits the use of nested relations or relations within relations as attribute values within tuples. To satisfy 1NF, data must be moved into separate tables where the data is of similar type in each table, and each table should have a primary key or foreign key as required by the project. This process eliminates repeating groups of data and creates new relations for each non-atomic attribute or nested relation. A relation is considered to be in 1NF only if it satisfies the constraints that contain the primary key only.

Second Normal Form

Second normal form (2NF) is a rule in database normalization that states that non-key attributes should not be functionally dependent on only the part of the primary key in a relation that has a composite primary key. In other words, each non-key attribute should depend on the entire primary key, not just a part of it. To achieve this, we need to decompose the table and create new relationships for each subkey along with their dependent attributes. It is important to maintain the relationship with the original primary key and all attributes that are fully functionally dependent on it. A relation is said to be in 2NF only if it satisfies all the 1NF conditions for the primary key and every non-primary key attribute of the relation is fully dependent only on the primary key.

Third Normal Form

Third normal form (3NF) requires that a relation have no non-key attribute that is functionally determined by another non-key attribute or set of non-key attributes. This means that there should be no transitive dependency on the primary key. To achieve 3NF, we decompose the relation and set up a new relation that includes non-key attributes that functionally determine other non-key attributes. This helps eliminate any dependencies that don't just rely on the primary key. A relation is considered a relation in 3NF if it satisfies the conditions of 2NF and, moreover, the non-key attributes of the relation are not dependent on any other non-key attribute.

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. Python Django 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 Python Django 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

In SQLite, an index serves as a database structure that optimizes the speed of table operations. Indexes can be created on one or more columns to expedite swift lookups and efficient ordering of records. When establishing an index in SQLite, it's essential to identify the columns frequently used in SQL queries and create one or more indexes on those specific columns. In practical terms, indexes in SQLite are akin to a separate table storing a primary key or index field along with pointers to corresponding records in the main table. These indexes remain hidden from users and are exclusively utilized by the SQLite database engine for swift record retrieval. The creation of indexes is accomplished using the CREATE INDEX statement.

In SQLite, the presence of indexes impacts the duration of INSERT and UPDATE statements as the database must adjust index values during these operations. Conversely, SELECT statements benefit from increased speed on tables with indexes because the index facilitates the rapid location of records. The strategic creation of indexes in SQLite thus involves a trade-off between enhanced query performance and potential overhead during insertion and updating operations.

4.5 TABLE DESIGN

1. Table Name : Tbl_Users

Primary key: **UserId**

No	Field name	Datatype (Size)	Key Constraints	Description of the field
1	UserId	int	Primary Key	Registered user id
2	Username	Varchar(30)	Not Null	User’s username
3	Name	Varchar(50)	Not Null	User’s name
4	Password	Varchar(50)	Not Null	user’s password
5	Email	Varchar(50)	Not Null	User’s email

2. Table Name : Tbl_Toy

Primary key: **pid**

Foreign key: **subcatid** references table **Tbl_SubCategory, brandId** references table **Tbl_Brand, sellerId** references table **Tbl_Seller, ageid** references table **Tbl_Age**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	pid	Int(10)	Primary key	product id
2.	pname	Int(10)	Not Null	Product name
3.	brandId	Int(10)	Foreign Key	Id for Brand Table
4.	subcatid	Int(10)	Foreign Key	Id for Sub Category Table
5	ageid	Int(10)	Foreign Key	Id for Age Table
6	sellerId	Int(10)	Foreign Key	Id for SellerTable
7.	description	Varchar(30)	Not Null	Product description
8.	price	Varchar(30)	Not Null	price
9.	material	Varchar(30)	Not Null	Materials
10	gender	Varchar(30)	Not Null	Gender

3. Table Name : Tbl_Category

Primary key: **cat_id**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of thefield
1.	cat_id	Int(10)	Primary key	Category id
2.	cat_name	Varchar(20)	Not Null	Category name

4. Table Name : Tbl_SubCategory

Primary key: sub**catid**

Foreign key: **cat_id** references table **Tbl_Category**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of thefield
1.	subcatid	Int(10)	primary key	Sub category id
2.	cat_id	Int(10)	Foreign Key	Id for Category Table
3.	Subcatname	Varchar(30)	Not Null	Sub category name

5. Table Name : Tbl_Age

Primary key: **ageid**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of thefield
1.	ageid	Int(10)	primary key	Age id
2	age_description	Varchar(20)	Not Null	Age_description

6. Table Name : Tbl_Brand

Primary key: **brand_Id**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	brand_Id	Int(10)	Primary key	Brand id
2.	brand_name	Int(10)	Not null	Brand name
3.	description	Varchar(20)	Not Null	Description

7. Table Name : Tbl_Cart

Primary key: **cart_Id**

Foreign key: **UserId** references table **Tbl_Users, pid** references table **Tbl_Toy**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	cart_Id	Int(10)	Primary key	Cart id
2.	UserId	Int(10)	Foreign Key	Id for User table
3.	pid	Int(10)	Foreign Key	Id for Toy table
4.	quantity	Varchar(30)	Not Null	Number of items ordered

8. Table Name : Tbl_Seller

Primary key: **sellerId**

Foreign key: **pid** references table **Tbl_Toy**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	sellerId	Int(10)	Primary key	Seller id
2.	pid	Int(10)	Foreign Key	Id for Toy table
3.	phone	Varchar(20)	Not Null	Phone no
4.	email	Varchar(30)	Not Null	Email id

9. Table Name : Tbl_Order

Primary key: **orderId**

Foreign key: **pid** references table **Tbl_Toy, UserId** references table **Tbl_Users**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	orderId	Int(10)	Primary key	Order id
2	pid	Int(10)	Foreign Key	Id for Toy table
3	UserId	Int(10)	Foreign Key	Id for User table
4	orderDate	date	Not Null	Order date
5	Total_amount	Varchar(30)	Not Null	Total amount
6	Order_status	Varchar(30)	Not Null	Order status

10. Table Name : Tbl_OrderItem

Primary key: **orderit_Id**

Foreign key: **pid** references table **Tbl_Toy**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	orderit_Id	Int(10)	Primary key	Order item id
2.	pid	Int(10)	Foreign Key	Id for Toy table
3.	quantity	Varchar(20)	Not Null	Quantity
4.	Sold_out	Varchar(30)	Not Null	Sold quantity
5.	date	date	Not Null	Date

11. Table Name : Tbl_Stock

Primary key: **stock_Id**

Foreign key: **pid** references table **Tbl_Toy**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	stock_Id	Int(10)	Primary key	Stock id
2.	pid	Int(10)	Foreign Key	Id for Toy table
3.	quantity	Varchar(20)	Not Null	Stock quantity
4.	Sold_out	Varchar(30)	Not Null	Sold quantity
5.	date	date	Not Null	Date

12. Table Name : Tbl_Payment

Primary key: **pay_id**

Foreign key: **orderId** references table **Tbl_Order**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	pay_id	Int(10)	Primary key	Payment id
2.	orderId	Int(10)	Foreign Key	Id for order table
3.	date	date	Not Null	Date
4.	Amount	int	Not Null	Amount
5.	Pay_status	date	Not Null	Payment status
6.	Pay_method	Varchar(20)	Not null	Payment method

13. Table Name : Tbl_Reviews

Primary key: **review_id**

Foreign key: **pid** references table **Tbl_Toy, UserId** references table **Tbl_Users**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	review_id	Int(10)	Primary key	Reviews id
2.	UserId	Int(10)	Foreign Key	Id for User table
3.	pid	Int(10)	Foreign Key	Id for product table
4.	rating	Int(10)	Not Null	rating
5.	comments	varchar	Not Null	comments

14. Table Name : Tbl_DeliveryBoy

Primary key: **dbId**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	dbId	Int(10)	Primary key	Delivery Boy id
2.	name	Varchar(10)	Not Null	Delivery Boy name
3.	phone	Varchar(20)	Not Null	Phone no
4.	email	Varchar(20)	Not Null	Email id

15. Table Name : Tbl_address

Primary key: **ship_id**

Foreign Key: **UserId** references table **Tbl_Users**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	Ship_id	Int(10)	Primary key	Shipment id
2.	UserId	Int(10)	Foreign Key	Id for User table
3.	shipname	Varchar(20)	Not Null	Name to which shipment to be done
4.	ShippingAddress	Varchar(20)	Not Null	Shipment Address
5.	shippingCity	Varchar(20)	Not Null	Shipment City
6.	shippingPincode	Varchar(20)	Not Null	Shipment pin code
7.	shipphone	Varchar(20)	Not Null	Shipment Phone Number

16. Table Name : Tbl_assign

Primary key: **dbaid**

Foreign Key: **dbId** references table **Tbl_ DeliveryBoy, orderId** references table **Tbl_Order**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	dbaid	Int(10)	Primary key	Delivery Boy Assign id
2.	dbId	Int(10)	Foreign Key	Id for Delivery Boy Table
3.	orderId	Int(10)	Foreign Key	Id for Order table
4.	deliver_status	Varchar(20)	Not Null	Status
5.	delivered_time	Varchar(20)	Not Null	Delivered Time

17. Table Name : Tbl_rating

Primary key: **rateid**

Foreign Key: **pid** references table **Tbl_Toy**

No:	Fieldname	Datatype(Size)	Key Constraints	Description of the field
1.	rateid	Int(10)	Primary key	Rate id
2.	pid	Int(10)	Foreign Key	Id for Toy Table
3.	status	Varchar(10)	Not Null	Status

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 tofind and fix defects, remaining defect density or frequency of occurrence, and test work-hours perregression 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 softwaredesign 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 softwareunit under test properly. The local data structure is inspected to ensure that data temporarily storedretains 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 errorsand 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. Any issues with the internal logic of the modules were fixed, and each module was tested and run separately after coding. 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 as a whole, 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 requirementsof the software. A software engineer can use this approach to create input conditions that will fullytest 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 inputand 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 orequipment 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 anytime of day and uses scripted sequences to evaluate the software. The results are reported, and thisinformation can be compared to previous test runs. Automation developers typically write code inprogramming 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. Cucumberis 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.One of the advantages of using Cucumber is its ability to bridge the gap between business stakeholders and technical teams. 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 usedfor interacting with web browsers and automating browser actions, while Cucumber provides a structured framework for organizing and executing tests. This combination allows for the creationof end-to-end tests that verify the behavior of web applications across different browsers and platforms, using a business-readable and maintainable format.

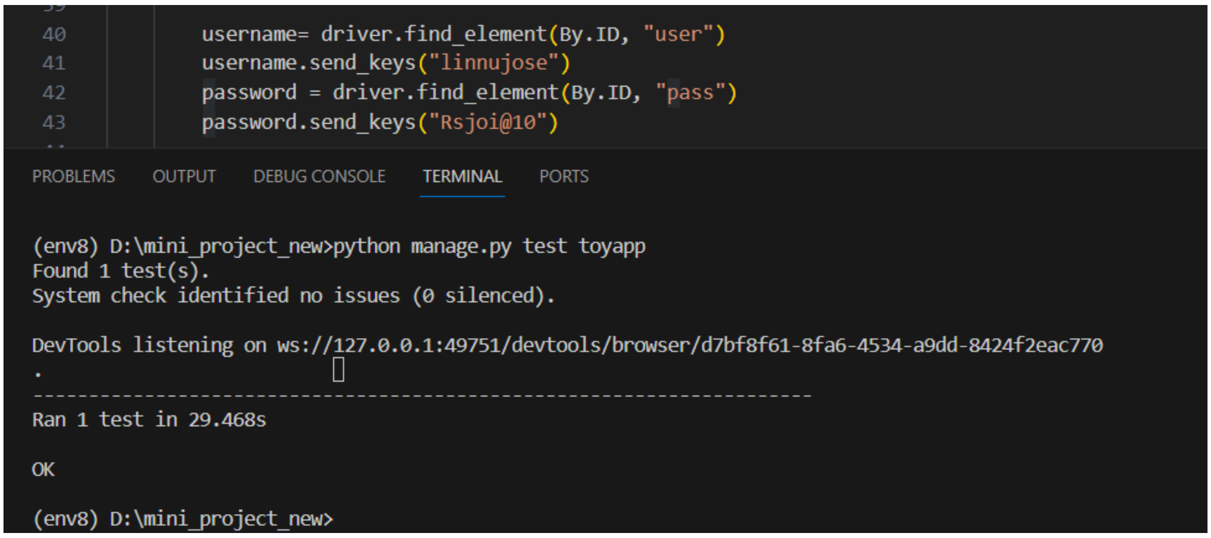
Test Case 1 : Customer Login

Code

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC


class Logintest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000/'

    def tearDown(self):
        self.driver.quit()

    def test_complete_shopping_flow(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(1)

        # Login
        submit1 = driver.find_element(By.ID, "login")
        submit1.click()
        time.sleep(2)
```

```
username= driver.find_element(By.ID, "user")
username.send_keys("linnujose")
password = driver.find_element(By.ID, "pass")
password.send_keys("Rsjoi@10")

submit1 = driver.find_element(By.ID, "sub")
submit1.click()
time.sleep(2)

logout = driver.find_element(By.ID, "log")
logout.click()
time.sleep(2)
```

Screenshot



```
username= driver.find_element(By.ID, "user")
username.send_keys("linnujose")
password = driver.find_element(By.ID, "pass")
password.send_keys("Rsjoi@10")
```

```
(env8) D:\mini_project_new>python manage.py test toyapp
Found 1 test(s).
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:49751/devtools/browser/d7bf8f61-8fa6-4534-a9dd-8424f2eac770
.
----------------------------------------------------------------------
Ran 1 test in 29.468s

OK

(env8) D:\mini_project_new>
```

Test Report

Test Case 1

Project Name: MerryKid					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Linnu Jose**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 3/12/2023**		
Module Name: Login Module			**Test Executed By : Ms. Navyamol K T**		
Test Title: Customer Login			**Test Execution Date: 4/12/2023**		
Description: **Verify login with valid email andpassword**					
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		Dashboard should be displayed	Login page displayed	Pass
2	ProvideValid Username	Username: linnujose	Customer should be able to Login	Customer Logged in and navigatedto Admin Dashboard with records	Pass
3	Provide Valid Password	Password : Rsjoi@10			
4	Click on Login Button				
5	Provide Invalid username or password	Username: Linnu Password: Ryy123	User should not be able inLogin	Message forenter valid email id or password displayed	Pass
6	Provide null username or password	Username :null Password: null			
7	Click on Login In button				
Post-Condition: User is validated with database and login to the website.The account session details are logged in database					

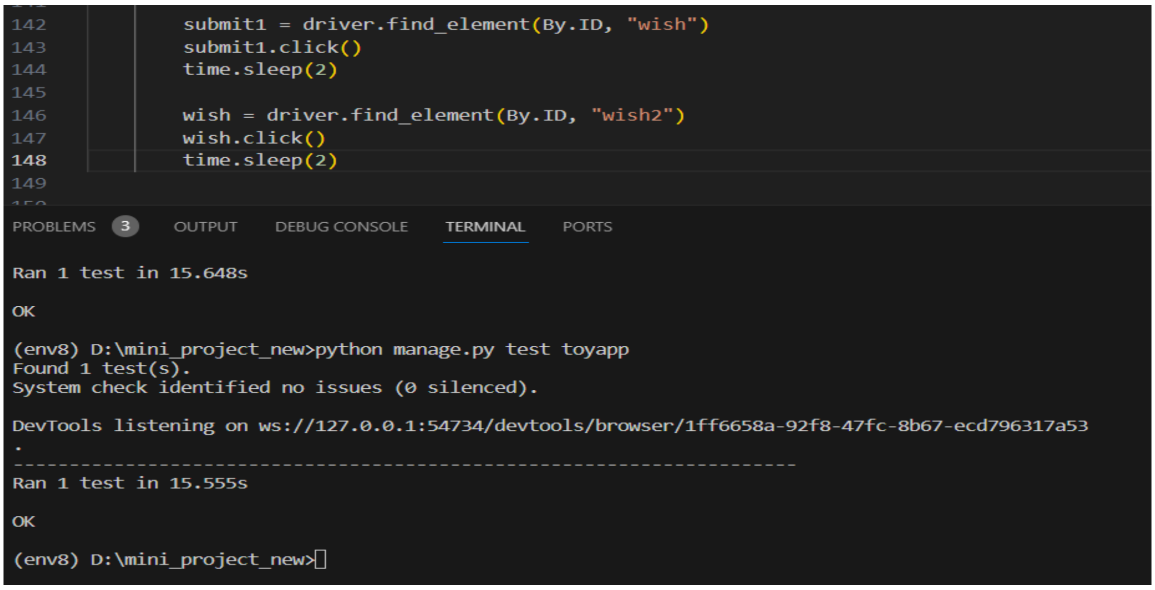
Test Case 2: Add to Wishlist

Code

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
class Logintest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000/product_list/'

    def tearDown(self):
        self.driver.quit()

    def test_complete_shopping_flow(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(1)

        submit1 = driver.find_element(By.ID, "wish")
        submit1.click()
        time.sleep(2)

        wish = driver.find_element(By.ID, "wish2")
        wish.click()
```

time.sleep(2)

Screenshot



```
        submit1 = driver.find_element(By.ID, "wish")
        submit1.click()
        time.sleep(2)

        wish = driver.find_element(By.ID, "wish2")
        wish.click()
        time.sleep(2)
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Ran 1 test in 15.648s

OK

(env8) D:\mini_project_new>python manage.py test toyapp
Found 1 test(s).
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:54734/devtools/browser/1ff6658a-92f8-47fc-8b67-ecd796317a53
.
----------------------------------------------------------------------
Ran 1 test in 15.555s

OK

(env8) D:\mini_project_new>
```

Test report

Test Case 2

Project Name: MerryKid	
Add to Wishlist Test Case	
Test Case ID: Test_2	**Test Designed By: Linnu Jose**
Test Priority(Low/Medium/High): High	**Test Designed Date:** 3/12/2023
Module Name: Add to Wishlist	**Test Executed By : Ms. Navyamol K T**
Test Title : User Add to Wishlist	**Test Execution Date: 4/12/2023**
Description: The product is added to wish by theuser	
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		Dashboard should be displayed	Login page displayed	Pass
2	ProvideValid Username	Username: linnujose	Customer should be able to Login	Customer Logged in and navigatedto Admin Dashboard with records	Pass
3	Provide Valid Password	Rsjoi@10			
4	Click on Login Button				
5	The User navigates to the manure page and clicks on add to wishlist button of a product		User should redirect to manure page and should be able to add an item to the wishlist	User navigated to manure page and added an item to wishlist	Pass

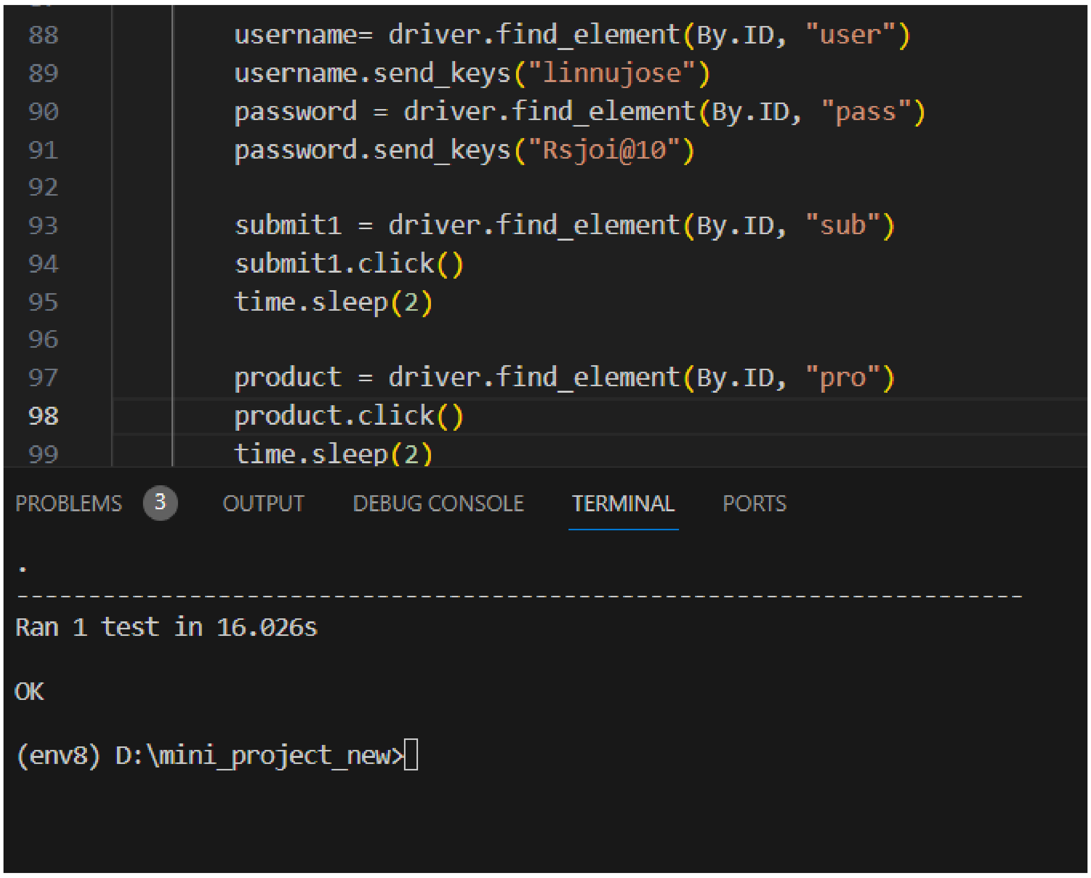
Post-Condition: User successfully added an item to the wishlist

Test Case 3: Add to cart

Code

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class Logintest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000/login/'

    def tearDown(self):
        self.driver.quit()

    def test_complete_shopping_flow(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(1)

        # Login
        # submit1 = driver.find_element(By.ID, "login")
        # submit1.click()
        # time.sleep(2)
        username= driver.find_element(By.ID, "user")
        username.send_keys("linnujose")
        password = driver.find_element(By.ID, "pass")
        password.send_keys("Rsjoi@10")

        submit1 = driver.find_element(By.ID, "sub")
        submit1.click()
        time.sleep(2)

        product = driver.find_element(By.ID, "pro")
        product.click()
        time.sleep(2)
```

Screenshot



```
        username= driver.find_element(By.ID, "user")
        username.send_keys("linnujose")
        password = driver.find_element(By.ID, "pass")
        password.send_keys("Rsjoi@10")

        submit1 = driver.find_element(By.ID, "sub")
        submit1.click()
        time.sleep(2)

        product = driver.find_element(By.ID, "pro")
        product.click()
        time.sleep(2)
```

```
.
----------------------------------------------------------------------
Ran 1 test in 16.026s

OK

(env8) D:\mini_project_new>
```

Test report

Test Case 3

Project Name: MerryKid					
Add to cart Test Case					
Test Case ID: Test_3			**Test Designed By: Linnu Jose**		
Test Priority(Low/Medium/High): High			**Test Designed Date**: 3/12/2023		
Module Name: **Add to cart**			**Test Executed By : Ms. Navyamol K T**		
Test Title : User Add to Cart			**Test Execution Date:** 4/12/2023		
Description: The product is added to cart by theuser					
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		Dashboard should be displayed	Login page displayed	Pass
2	ProvideValid Username	Username: linnujose	Customer should be able to Login	Customer Logged in and navigatedto Admin Dashboard with records	Pass
3	Provide Valid Password	Password: Rsjoi@10			
4	Click on Login Button				
5	The User navigates to the manure page and clicks on add to cart button of a product		User should redirect to manure page and should be able to add an item to the cart	User navigatedto manure pageand added an item to cart	Pass
Post-Condition: User successfully added an item to the cart					

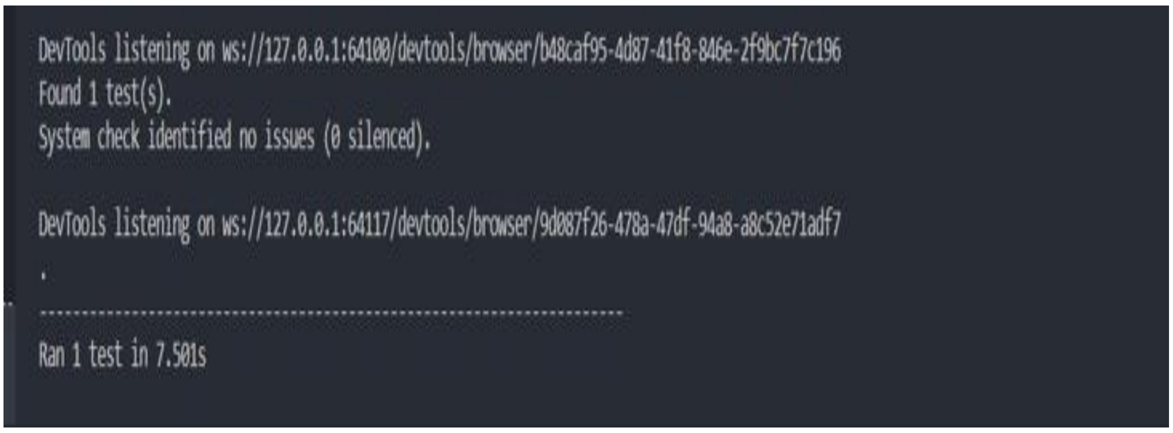
Test Case 4: Product Filtering test

Code

```
#test4
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Start the WebDriver (you can use other browsers as well) driver
= webdriver.Chrome()

try:
    # Open the HTML page
    url = 'http://127.0.0.1:8000/serums_products/'  # Replace with the actual file path or
URL
    driver.get(url)

    # Wait for the product list to be present on the page product_list
    = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CLASS_NAME, 'product-list'))
    )

    # Get the list of products
    products = product_list.find_elements(By.CLASS_NAME, 'product')

    # Perform assertions or interactions based on your requirements assert
    len(products) > 0, "No products found on the page."

    # Example: Click on the first product and check if it redirects to the product detail
page


    first_product = products[0]
    product_link   =   first_product.find_element(By.TAG_NAME,   'a')
    product_link.click()
```

```
# Wait for the product detail page to load
WebDriverWait(driver, 10).until(
    EC.title_contains('Product Detail')
)

# Perform more assertions or interactions on the product detail page if needed finally:
# Close the browser window
driver.quit()
```

Screenshot



```
DevTools listening on ws://127.0.0.1:64100/devtools/browser/b48caf95-4d87-41f8-846e-2f9bc7f7c196
Found 1 test(s).
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:64117/devtools/browser/9d087f26-478a-47df-94a8-a8c52e71adf7
.
----------------------------------------------------------------------
Ran 1 test in 7.501s
```

Test report:

Test Case 4

Project Name: MerryKid					
Product Filtering Test Case					
Test Case ID: Test_4			**Test Designed By: Linnu Jose**		
Test Priority(Low/Medium/High): High			**Test Designed Date: 3/12/2023**		
Module Name: **Product Filtering**			**Test Executed By : Ms. Navyamol K T**		
Test Title : Product Filtering			**Test Execution Date: 4/12/2023**		
Description: Product Filtering test					
Pre-Condition : User can filter products					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigation to Login Page		Dashboard should be displayed	Login page displayed	Pass
2	ProvideValid Username	Username: linnujose	Customer should be able to Login	Customer Logged in and navigatedto Admin Dashboard with records	Pass
3	Provide Valid Password	Rsjoi@10			
4	Click on Login Button				
5	Product Filtering test		Logined user should filter products	Logined user can filter products	Pass
Post-Condition: User is validated with database and login to the website. User can filter products based on category					

CHAPTER 6
IMPLEMENTATION

6.1INTRODUCTION

Implementation is the stage of the project where the theoretical design is turned into a working system. It can be considered to be the most crucial stage in achieving a successful new system gaining the users confidence that the new system will work and will be effective and accurate. It is primarily concerned with user training and documentation. Conversion usually takes place about the same time the user is being trained or later. Implementation simply means convening a new system design into operation, which is the process of converting a new revised system design into an operational one. At this stage the main work load, the greatest upheaval and the major impact on the existing system shifts to the user department. If the implementation is not carefully planned or controlled, it can create chaos and confusion. Implementation includes all those activities that take place to convert from the existing system to the new system. The new system may be a totally new, replacing an existing manual or automated system or it may be a modification to an existing system. Proper implementation is essential to provide a reliable system to meet organization requirements. The process of putting the developed system in actual use is called system implementation. This includes all those activities that take place to convert from the old system to the new system. The system can be implemented only after through testing is done and if it is found to be working according to the specifications. The system personnel check the feasibility of the system. The more complex the system being implemented, the more involved will be the system analysis and design effort required to implement the three main aspects: education and training, system testing and changeover. The implementation state involves the following tasks: Careful planning. Investigation of system and constraints. Design of methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

Implementation of software refers to the final installation of the package in its real environment, to the satisfaction of the intended uses and the operation of the system. In many organizations someone who will not be operating it, will commission the software development project. In the initial stage people doubt about the software but we have to ensure that the resistance does not build up, as one has to make sure that: The active user must be aware of the benefits of using the new system. Their confidence in the software is built up. Proper guidance is imparted to the user so that he is comfortable in using the application. Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not up running on the server, the actual process won't 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 the user will have to be trained on the new application software. This will give the underlying philosophy of the use of the new system such as the screen flow, screen design type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the ways to correct the date entered. It should thencover information needed by the specific user/ group to use the system or part of the system while imparting the training of the program on the application. This training may be different across different user groups and across different levels of hierarchy.

6.2.3 System Maintenance

Maintenance is the enigma of system development. The maintenance phase of the software cycle is the time in which a software product performs useful work. After a system is successfully implemented, it should be maintained in a proper manner. System maintenance is an important aspect in the software development life cycle. The need for system maintenance is for it to make adaptable to the changes in the system environment. Software maintenance is of course, far more than "Finding Mistakes".

6.2.4 Hosting

Hosting refers to the process of providing infrastructure, services, and resources necessary for storing and serving content, applications, or websites over the internet. It involves storing website files, databases, and other resources on a server that is connected to the internet, making them accessible to users who request them via web browsers or other client applications.

Hosting services are typically provided by hosting providers, who maintain server hardware, networking infrastructure, and data centers to ensure the availability and performance of hosted content. These services may include shared hosting, where multiple websites share resources on a single server, virtual private servers (VPS), which offer dedicated resources within a virtualized environment, dedicated hosting, where a single server is exclusively allocated to one client, and cloud hosting, which provides scalable and flexible hosting solutions using cloud infrastructure.

Hosting providers often offer additional services such as domain registration, website builders, email hosting, security features, and technical support to help users manage and optimize their hosted content. The goal of hosting is to ensure that websites and applications are accessible to users with minimal downtime, optimal performance, and robust security measures in place.

6.2.4.1 AMAZON ELASTIC COMPUTE CLOUD (EC2)

Amazon Elastic Compute Cloud (EC2) is a highly scalable cloud computing service provided by Amazon Web Services (AWS). It enables users to rent virtual servers, known as instances, on which they can run their applications and services. EC2 offers a wide range of instance types optimized for various workloads, providing flexibility in terms of computing power, memory, storage, and networking resources. Users have full control over their instances, allowing them to customize configurations, choose operating systems, and scale resources up or down as needed. EC2 is a fundamental building block of cloud infrastructure, empowering businesses to deploy and manage applications in a reliable, secure, and cost-effective manner.

Procedure for hosting a website on 000Webhost:

- **Step 1:** Create an AWS Account: Begin by registering for an AWS account if you haven't already done so. Once logged in to the AWS Management Console, proceed to the EC2 dashboard.

- **Step 2:** Launch an EC2 Instance: Click on the "Launch Instance" button to initiate a new EC2 instance. Choose an appropriate Amazon Machine Image (AMI) and select an instance type based on your project's requirements, such as the amount of CPU, memory, and storage needed.

- **Step 3:** Configure Instance Settings: Customize instance parameters including the number of instances, networking configurations, and storage preferences. Configure storage volumes to securely store website files and data.

- **Step 4:** Set Up Security Group: Establish a new security group or use an existing one to define firewall rules. Ensure that ports 80 (HTTP) and 443 (HTTPS) are open for web traffic, and port 22 (SSH) for remote access.

- **Step 5:** Launch Instance: Review the instance setup and start the EC2 instance launch process. Optionally, choose or generate a key pair for SSH access.

- **Step 6:** Connect to the Instance: Once the instance is launched, establish an SSH connection to it using the public IP address or DNS name of the instance, along with the key pair generated in the previous step.

- **Step 7:** Install Web Server Software: Install the necessary web server software such as Apache, Nginx, or another server of your choice on the EC2 instance. Configure the server to serve your website files.

- **Step 8:** Upload Website Files: Transfer your website files to the EC2 instance using secure file transfer protocols like SCP or SFTP, or by cloning your website's repository from a version control system like Git.

- **Step 9:** Configure Domain Name: If you own a domain name, configure the DNS settings to point to the public IP address or DNS name of your EC2 instance. This allows users to access your website using your domain name.

- **Step 10:** Configure SSL/TLS Certificate (Optional): Enable HTTPS for your website by setting up an SSL/TLS certificate using AWS Certificate Manager or a third-party provider. This enhances the security of data transmitted between the website and its visitors.

- **Step 11:** Test the Website: Access a web browser and navigate to your website's URL (either the public IP address or domain name). Verify that the website is accessible and functions correctly.

- **Step 12:** Monitor and Maintain: Utilize AWS monitoring tools such as CloudWatch to monitor the performance, security, and uptime of your EC2 instance. Regularly update your instance and web server software to ensure security patches are applied and to optimize performance.

Hosted Website: Amazon EC2

Hosted Link: http://15.206.211.171:8000/

Screenshot

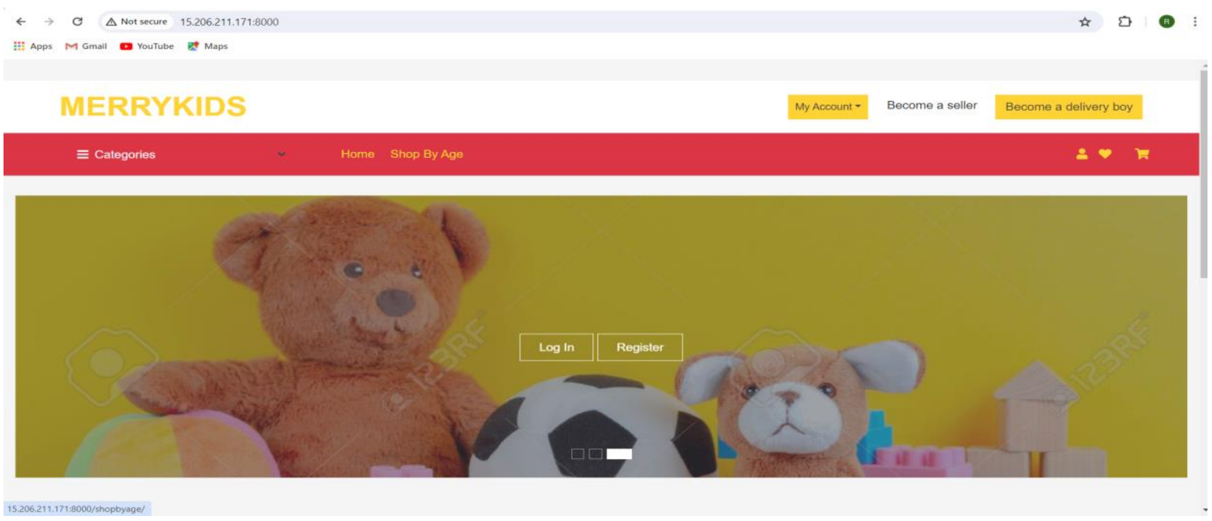


Fig.1

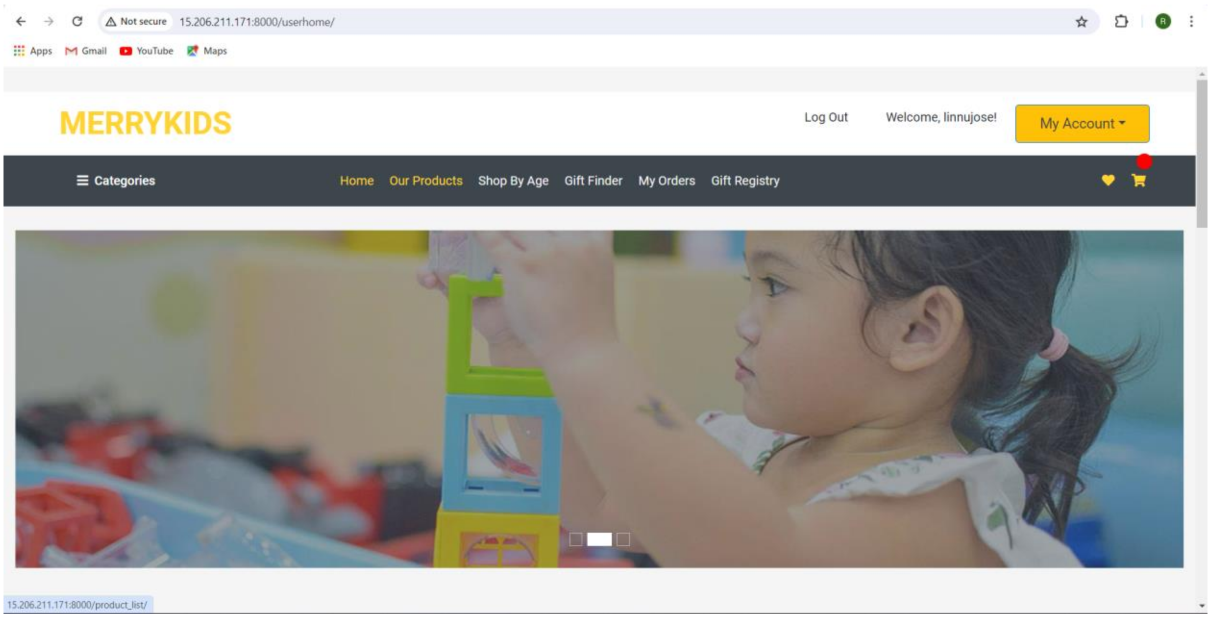

Not secure 15.206.211.171:8000/userhome/
Apps Gmail YouTube Maps
MERRYKIDS
Log Out
Welcome, linnujose!
My Account
Categories
Home
Our Products
Shop By Age
Gift Finder
My Orders
Gift Registry
15.206.211.171:8000/product_list/

Fig.2

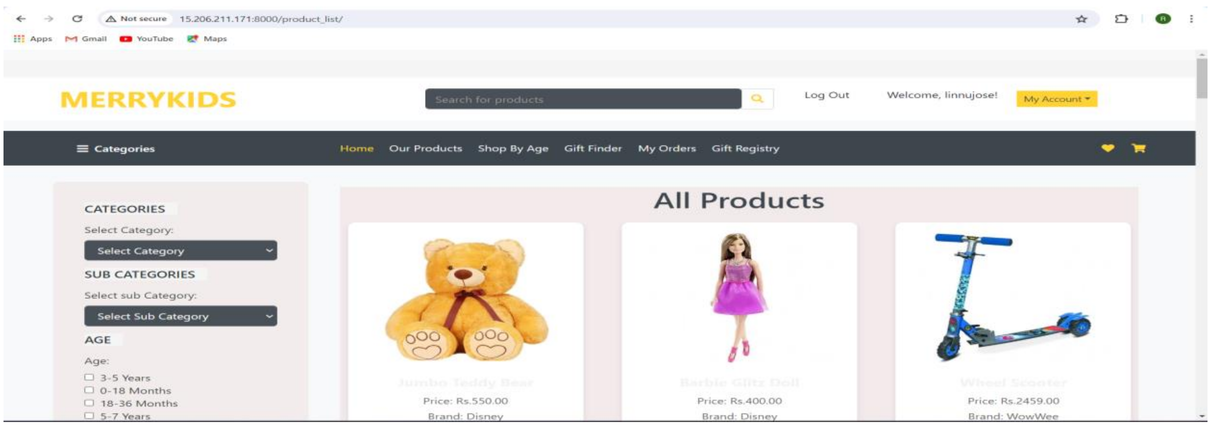

Not secure 15.206.211.171:8000/product_list/
Apps Gmail YouTube Maps
MERRYKIDS
Search for products
Log Out
Welcome, linnujose!
My Account
Categories
Home
Our Products
Shop By Age
Gift Finder
My Orders
Gift Registry
CATEGORIES
Select Category:
Select Category
SUB CATEGORIES
Select sub Category:
Select Sub Category
AGE
Age:
3-5 Years
0-18 Months
18-36 Months
5-7 Years
All Products
Jumbo Teddy Bear
Price: Rs.550.00
Brand: Disney
Barbie Glitz Doll
Price: Rs.400.00
Brand: Disney
Wheel Scooter
Price: Rs.2459.00
Brand: WowWee

Fig.3

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The analysis of the "MerryKid" system reveals that it is a well-designed and efficient platform for online toy shopping and inventory management. It addresses the need for a user-friendly online toy store, providing a comprehensive set of features for customers and store administrators. The key features of the system, including toy browsing, searching, and user account management, contribute to its effectiveness. The "MerryKid" project focuses on creating a user-friendly interface with features like a visually appealing homepage, easy toy browsing and searching, and detailed toy pages. Customers can personalize their accounts, manage preferences, and access their shopping carts for toy purchases. From the admin perspective, the system offers a comprehensive dashboard for managing toy, customer accounts, and site settings. Admins have the authority to add, edit, or remove toys, manage customer accounts, and customize site settings to enhance the user experience. The "MerryKid" system aims to provide a seamless and efficient experience for both customers and administrators, leveraging technology to create a comprehensive online toy shopping platform.

7.2 FUTURE SCOPE

MerryKid, the Online Toyshop management system has significant potential for growth and improvement in the dynamic world of toy. With the increasing trend of online toy shopping, MerryKid can consider expanding its product range by including a diverse selection of toys. This expansion could focus on providing detailed product information, customer reviews, and personalized recommendations to enhance the overall shopping experience. To promote community engagement, the platform might introduce features like toy forums, virtual toy consultations, and user-generated content, allowing customers to share toy tips, experiences, and recommendations. Utilizing emerging technologies such as augmented reality for virtual play previews and personalized toy algorithms for tailored product suggestions can further enhance the customer experience. However, addressing challenges related to accurate representation of toy colors, ensuring a secure and user-friendly interface, and staying updated on evolving toy trends will be crucial for the sustained success of the MerryKid.

.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- PankajJalote, “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.firstcry.com/
- www.w3schools.com
- www.bootstrap.com
- https://chat.openai.com/chat
- www.jquery.com

CHAPTER 9

APPENDIX

9.1 Sample Code

Login

```
<!-- <html>
<h1>login</h1>
</html> -->
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="UTF-8">
    <title> Login Form </title>
    <link rel="stylesheet" href="{% static '/css/login.css' ' %}">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.2/css/all.min.css"/>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">


    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
      @import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700&display
=swap');
      *{
        margin: 0;
        padding: 0;
        box-sizing: border-box;
        font-family: 'Poppins',sans-serif;
      }
      html, body{
        display: grid;
        height: 100vh;
        width: 100%;
        place-items: center;
        background: linear-gradient(to right, #a09c9e 0%, #baacb3 100%);



        /* background-image: url('/static/img/image3.jpg');
        background-size: cover;
        background-repeat: no-repeat; */

      }
      ::selection{
        background: #ff80bf;

      }
      .container{
        background: #fff;
        /* max-width: 350px; */
        max-width: 450px;
        width: 100%;
        padding: 25px 30px;
        border-radius: 5px;
        box-shadow: 0 10px 10px rgba(0, 0, 0, 0.15);
      }
      .container form .title{
        font-size: 30px;
        font-weight: 600;
        margin: 20px 0 10px 0;
        position: relative;
      }
      .container form .title:before{
        content: '';
        position: absolute;
        height: 4px;
        width: 33px;
        left: 0px;
        bottom: 3px;
        border-radius: 5px;
        background: linear-gradient(to right, #99004d 0%, #ff0080 100%);
      }
      .container form .input-box{
        width: 100%;
        height: 45px;
        margin-top: 25px;
        position: relative;
      }
      .container form .input-box input{
        width: 100%;
        height: 100%;
        outline: none;
        font-size: 16px;
        border: none;
      }
      .container form .underline::before{
        content: '';
        position: absolute;
        height: 2px;
        width: 100%;
        background: #ccc;
        left: 0;
        bottom: 0;
      }
      .container form .underline::after{
        content: '';
        position: absolute;
        height: 2px;
        width: 100%;
        background: linear-gradient(to right, #99004d 0%, #ff0080 100%);
        left: 0;
        bottom: 0;
        transform: scaleX(0);
        transform-origin: left;
        transition: all 0.3s ease;
      }
      /* .container form .input-box input:focus ~ .underline::after,
      .container form .input-box input:valid ~ .underline::after{
        transform: scaleX(1);
        transform-origin: left;
      } */
      .container form .button{
        margin: 40px 0 20px 0;
      }
      .container .input-box input[type="submit"]{
        background: linear-gradient(to right, #4016e9 0%, #182fc6 100%);
        font-size: 17px;
        color: #fff;
        border-radius: 5px;
        cursor: pointer;
        transition: all 0.3s ease;
      }
      .container .input-box input[type="submit"]:hover{
        letter-spacing: 1px;
        background: linear-gradient(to left, #1f15db 0%, #2023e2 100%);
      }
      .container .option{
        font-size: 14px;
        text-align: center;
      }
      .container .forgotpass a,
      .container .twitter a{
        display: block;
        height: 45px;
        width: 100%;
        font-size: 15px;
        text-decoration: none;
        padding-left: 20px;
        line-height: 45px;
        color: #fff;
        border-radius: 5px;
        transition: all 0.3s ease;
      }

      .container .forgotpass i,
      .container .twitter i{
        padding-right: 12px;
        font-size: 20px;
      }
      .container .twitter a{
        background: linear-gradient(to right,  #00acee 0%, #1abeff 100%);
        margin: 20px 0 15px 0;
      }
      .container .twitter a:hover{
        background: linear-gradient(to left,  #00acee 0%, #1abeff 100%);
        margin: 20px 0 15px 0;
      }
      .container .forgotpass a{
        background: linear-gradient( to right,  #3b5998 0%, #476bb8 100%);
        margin: 20px 0 50px 0;
      }
      .container .forgotpass a:hover{
        background: linear-gradient( to left,  #3b5998 0%, #476bb8 100%);
        margin: 20px 0 50px 0;
      }



      /* password */
```

```
  .container form .input-box {
    position: relative;
}

.container form .input-box i {
    position: absolute;
    top: 50%;
    left: 10px;
    transform: translateY(-50%);
    color: #121111; /* Adjust the color of the icon */
}

.container form .input-box input {
    width: calc(100% - 30px); /* Adjust the width of the input box */
    padding-left: 30px; /* Ensure the text doesn't overlap with the icon */
    height: 100%;
    outline: none;
    font-size: 16px;
    border: none;
}

/* username  */
.container form .input-box {
    position: relative;
}

.container form .input-box i {
    position: absolute;
    top: 50%;
    left: 10px;
    transform: translateY(-50%);
    color: #151212; /* Adjust the color of the icon */
}
```

```
.container form .input-box input {
   width: calc(100% - 30px); /* Adjust the width of the input box */
   padding-left: 30px; /* Ensure the text doesn't overlap with the icon */
   height: 100%;
   outline: none;
   font-size: 16px;
   border: none;
}

.container form .underline::before {
   content: '';
   position: absolute;
   height: 2px;
   width: 100%;
   background: #ccc;
   left: 0;
   bottom: 0;
}

  </style>
  </head>
  <body>
   <div class="container">

     <form action="#" method="post">

       {% csrf_token %}
       <div class="title">Login</div>
       <div class="input-box underline">
         <i class="fas fa-user"></i> <!-- Font Awesome user icon -->
         <input type="text" placeholder=" username" id="user" name="username" required>
         <div class="underline"></div>
         <div class="error-message" id="username-error"></div>
       </div>
```

```
        <div class="input-box">

          <i class="fas fa-lock"></i><input type="password" placeholder=" Password" id="pass"
name="password" required>
          <div class="underline"></div>
          <div class="error-message" id="password-error"></div>
        </div>
        <div class="input-box button">
          <input type="submit" name="submit" id="sub" value="Log In">
        </div>
      </form>
        <div class="option"></div>
        <div class="option"><p class="alternative">Don't have an account? <a href="{% url
'register' %}">Register Now</a></p></div>
        <div class="forgotpass">
          <a href="{% url 'reset_password' %}">Forgot Your Password</a>
        </div>
        <!-- <div class="twitter">
          <a href=""><i class="fab fa-google"></i>Sign in With Google</a>
        </div> -->

    </div>
    <script>
      function validateForm() {
          // Get references to the email and password input fields
          var email = document.forms["loginForm"]["username"].value;
          var password = document.forms["loginForm"]["password"].value;

          // Get references to the error message containers
          var emailError = document.getElementById("username-error");
          var passwordError = document.getElementById("password-error");

          // Reset error messages
```

```
            emailError.innerHTML = "";
            passwordError.innerHTML = "";



            // Check if the email and password are not empty
            if (email === "") {
                emailError.innerHTML = "Email is required.";
                return false; // Prevent the form from submitting
            }

            if (password === "") {
                passwordError.innerHTML = "Password is required.";
                return false; // Prevent the form from submitting
            }

            // You can add more specific validation rules for email and password here

            // If everything is valid, the form will submit
            return true;



        }

    </script>
    <!-- Add this script at the end of your HTML file -->
<script>
  document.addEventListener("DOMContentLoaded", function () {
      // Add an event listener to the form
      document.querySelector("form").addEventListener("log in", function (event) {
          // Prevent the default form submission
          event.preventDefault();

          // Validate the form (you can keep your existing validation logic here)
          // Assuming the form is valid, show a success message
          alert("You are successfully logged in!");

          // Redirect the user to the userhome page
          window.location.href = "{% url 'userhome' %}";
      });
  });
</script>
  </body>
</html>
```

Login : views,py

```
def login(request):
    #session st
    # if 'username' in request.session:
    if request.user.is_authenticated:

        return redirect(userhome)
        #session en
    if request.method == 'POST':
        username = request.POST["username"]
        password = request.POST["password"]
        # if user is not None:
        #     auth_login(request, user)
        #     return redirect('/userhome')
        # else:
        #    messages.success(request,("Invalid credentials."))
        # print(username)  # Print the email for debugging
        # print(password)  # Print the password for debugging

        if username and password:
            user = authenticate(request, username =username , password=password)
            if user is not None:
```

```
            #session start
            # request.session['username'] = username
            # login(request, user)

            #session end
            auth_login(request,user)

            if request.user.user_types==CustomUser.CUSTOMER:

                return redirect('userhome')
            elif request.user.user_types == CustomUser.SELLER:
                 print("user is seller")
                 return redirect('sellerhome')

            elif request.user.user_types == CustomUser.ADMIN:
                print("user is admin")
                return redirect('http://127.0.0.1:8000/admin/')
            # else:
            #     print("user is normal")
            #     return redirect('')

        else:
            messages.success(request,("Invalid credentials."))
    else:
        messages.success(request,("Please fill out all fields."))

  return render(request, 'login.html')
```

Add Product

```
{% load static %}
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>add_product</title>
    <link rel="stylesheet" href="style.css" />
            <style>
            /* Import Google font - Poppins */
@import
url("https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700&display
=swap");
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: "Poppins", sans-serif;
}
body {
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 20px;
  background: rgb(130, 106, 251);
}
.container {
  position: relative;
  max-width: 700px;
  width: 100%;
  background: #fff;
  padding: 25px;
  border-radius: 8px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}
.container header {
  font-size: 1.5rem;
  color: #333;
  font-weight: 500;
  text-align: center;
}
.container .form {
  margin-top: 30px;
}
.form .input-box {
  width: 100%;
  margin-top: 20px;
}
.input-box label {
  color: #333;
}

textarea {
    height: 100%;
}

.form :where(.input-box input, .select-box) {
  position: relative;
  height: 50px;
  width: 100%;
  outline: none;
  font-size: 1rem;
  color: #707070;
  margin-top: 8px;
  border: 1px solid #ddd;
  border-radius: 6px;
  padding: 0 15px;
}
.input-box input:focus {
  box-shadow: 0 1px 0 rgba(0, 0, 0, 0.1);
}
.form .column {
  display: flex;
  column-gap: 15px;
}
.form .gender-box {
  margin-top: 20px;
}
.gender-box h3 {
  color: #333;
  font-size: 1rem;
  font-weight: 400;
  margin-bottom: 8px;
}
.form :where(.gender-option, .gender) {
  display: flex;
  align-items: center;
  column-gap: 50px;
  flex-wrap: wrap;
}
.form .gender {
  column-gap: 5px;
}
.gender input {
  accent-color: rgb(226, 223, 241);
}
.form :where(.gender input, .gender label) {
  cursor: pointer;
```

```
}
.gender label {
  color: #707070;
}
.address :where(input, .select-box) {
  margin-top: 15px;
}
.select-box select {
  height: 100%;
  width: 100%;
  outline: none;
  border: none;
  color: #707070;
  font-size: 1rem;
}
.form button {
  height: 55px;
  width: 100%;
  color: #fff;
  font-size: 1rem;
  font-weight: 400;
  margin-top: 30px;
  border: none;
  cursor: pointer;
  transition: all 0.2s ease;
  background: rgb(243, 243, 245);
}
.form button:hover {
  background: rgb(221, 220, 227);
}
/*Responsive*/
@media screen and (max-width: 500px) {
  .form .column {
    flex-wrap: wrap;
  }
  .form :where(.gender-option, .gender) {
    row-gap: 15px;
  }
}


                .modal{
                        display: none;
                        position: fixed;
                        top: 0;
                        left: 0;
                        width: 100%;
                        height: 100%;
                        background-color: rgba(0, 0, 0, 0.7);
                        z-index: 1;
                }

                .modal-content {
                        background-color: #fff;
                        border-radius: 5px;
                        position: absolute;
                        top: 50%;
                        left: 50%;
                        transform: translate(-50%, -50%);
                        padding: 20px;
                        text-align: center;
                }

                #successModal {
                        display: none;
                }

                .form-group {
                    margin-bottom: 20px;
               }


    .error-message {
    color: red;
    font-size: 0.8rem; /* Adjust the font size as needed */
    margin-top: 5px;
  }


/* sidebar  start*/


    /* sidebar end */






        </style>
  </head>
  <body>


    <section class="container">
      <header>Add product Form</header>
      <form method="post" action="{% url 'add_product' %}" enctype="multipart/form-data"
id="productForm" class="form">
        {% csrf_token %}
        <div class="input-box">
          <label>Product Name</label>
          <input type="text" id="productName" name="product_name" placeholder="Enter Product
```

```
Name" required oninput="validateProductName()" />
          <span id="productNameError" class="error-message"></span>
        </div>

        <div class="input-box">
          <label>Product Description</label>
          <input type="text" id="productDescription" name="product_description"
placeholder="Enter Product Description" maxlength="500" required
oninput="validateProductDescription()" />
          <span id="productDescriptionError" class="error-message"></span>
        </div>

        <div class="input-box address">
          <label>Category</label>
          <div class="column">
            <div class="select-box">
              <select name="category" id="productCategory">
                <option value="{{ category.id }}">Category</option>
                {% for category in category %}
                  <option value="{{ category.id }}">{{ category.name }}</option>
                {% endfor %}
                <option></option>
              </select>
            </div>
            <div class="select-box">
              <select name="sub_category" id="productSubcategory">
                <option value="{{ sub.id }}"> SubCategory</option>
                {% for sub in sub %}
                  <option value="{{ sub.id }}" data-category="{{ sub.id }}">{{ sub.name }}</option>
                {% endfor %}
                <option hidden>Sub Category</option>
              </select>
            </div>
          </div>
```

```
    </div>
    <script>
     document.addEventListener("DOMContentLoaded", function () {
       var categorySelect = document.getElementById("productCategory");
       var subcategorySelect = document.getElementById("productSubcategory");

       // Function to update subcategories based on the selected category.
       function updateSubcategories() {
         var selectedCategory = categorySelect.value;

         // Hide all subcategory options.
         for (var i = 0; i < subcategorySelect.options.length; i++) {
           subcategorySelect.options[i].style.display = "none";
         }

         // Show subcategories that match the selected category.
         for (var i = 0; i < subcategorySelect.options.length; i++) {
           var subcategoryOption = subcategorySelect.options[i];
           if (subcategoryOption.getAttribute("data-category") === selectedCategory ||
subcategoryOption.getAttribute("data-category") === "") {
             subcategoryOption.style.display = "block";
           }
         }
       }

       // Initial update based on the selected category on page load.
       updateSubcategories();

       // Add an event listener to update subcategories when the category is changed.
       categorySelect.addEventListener("change", updateSubcategories);
     });
     </script>

                  <!-- <div class="input-box address">
```

```
        <label>Category</label>

        <div class="column">
          <div class="select-box">
            <select name="category" id="productCategory">
              {% for category in category %}
                    <option value="{{ category.id }}">{{category.name}}</option>
              {%endfor%}
                                        <option></option>
            </select>
          </div>

          <div class="select-box">

            <select name="sub_category" id="productSubcategory">
              {% for sub in sub %}
              <option value="{{ sub.id }}">{{sub.name}}</option>
              {%endfor%}
              <option hidden>Sub Category</option>
            </select>
          </div>
        </div> -->
                      <!-- <option hidden>Category</option>
              <option value="1">Toys & Games</option>
              <option value="2">Ride-Ons & Cycle</option>
              <option value="3">Soft Toys</option>
              <option value="4">Vehicles</option>
              <option value="5">Action Figures</option>
              <option value="6">Board Games</option> -->
                <!-- <option value="1">Teddy Bears</option>
              <option value="2">Dolls & Playsets</option>
              <option value="3">Outdoors Sports</option>
              <option value="4">Tricycle</option>
              <option value="5">Scooter</option>
              <option value="6">Superhero</option>
              <option value="7">Ludo,Snakes & Ladders</option>
              <option value="7">Playing Cards</option> -->
        <div class="column">
          <div class="select-box">
            <select name="brand" id="productBrand">
              <option hidden>Brand</option>
              {% for brand in brands %}
               <div class="form-check">
                  <input type="checkbox" class="form-check-input" id="brand{{ brand.id }}"
      name="brand" value="{{ brand.id }}">
                  <label class="form-check-label" for="brand{{ brand.id }}">{{ brand.name
      }}</label>
               </div>
            {% endfor %}

              <!-- <option value="1">Disney</option>
              <option value="2">Lego</option>
              <option value="3">Barbie</option>
              <option value="4">Hotwheels</option>
                                         <option value="5">WowWee</option>
              <option value="6">Nerf</option> -->

            </select>
          </div>

                      <div class="select-box">
            <select name="age" id="age">
              <option hidden>Age</option>
              <option value="1">3-5 Years</option>
              <option value="2">0-18 Months</option>
              <option value="3">18-36 Months</option>
              <option value="4">5-7 Years</option>
              <option value="5">7-9 Years</option>
```

```
            <option value="6">9-12 Years</option>
            <option value="7">12+ Years</option>

          </select>
        </div>
      </div>
    </div>



                <div class="column">
      <div class="input-box">
        <label>Material</label>
        <input type="text" name="material" placeholder="material"  required
oninput="validateMaterial()" />
        <span id="materialError" class="error-message"></span>
      </div>
      <div class="input-box">
        <label>Price</label>
        <input type="number" name="price" placeholder="" min="0.01" step="0.01"required
oninput="validatePrice()" />
        <span id="priceError" class="error-message"></span>
      </div>
    </div>




    <div class="gender-box">
      <h3>Gender</h3>
      <div class="gender-option">
        <div class="gender">
          <input type="radio" id="check-male" name="gender" checked />
          <label for="check-male">male</label>
        </div>
        <div class="gender">
          <input type="radio" id="check-female" name="gender" />
          <label for="check-female">Female</label>
        </div>
        <div class="gender">
          <input type="radio" id="check-other" name="gender" />
          <label for="check-other">Unisex</label>
        </div>
      </div>
    </div>



            <div class="input-box address">
      <label>Seller</label>

      <div class="column">
        <div class="select-box">
          <select name="seller">
            <option hidden>select a seller</option>

                {% for seller in sellers %}
                    <option value="{{ seller.id }}">{{ seller.shop_name }}</option>
                {% endfor %}
                <!-- Options for sellers should be populated from your database -->

          </select>
        </div>





        <label for="stockQuantity">Stock Quantity</label>
        <div class="column"></div>
```

```
                <input type="number" id="stockQuantity" name="stock_quantity" required
oninput="validateStockQuantity()" />
                <span id="stockQuantityError" class="error-message"></span>



          </div>
          <div class="column">
            <div class="input-box">
            <label>Status</label>
                                <div class="select-box">
              <select name="status">
                <option hidden>Status</option>
                <option>Available</option>
                <option>Not Available</option>
              </select>
            </div>

          </div>
          <div class="input-box">

          </div>
          </div>
        </div>




        <div class="input-box address">
          <label>Product Image(JPEG,JPG,PNG only)</label>
          <input type="file" id="productImage" name="product_image" accept=".jpeg, .jpg, .png"
required>



        </div>
```

```
                  <div class="form-group">
         <button type="submit" id="addProductButton" >Add Product<a href="{% url 'sellerhome'
%}"></a></button>
      </div>

                  <div class="modal" id="successModal">
      <div class="modal-content">
         <p>PRODUCT ADDED SUCCESSFULLY</p>
      </div>
   </div>

   <script src="{% static 'add_product.js' %}"></script>



    </form>
   </section>



   <!-- validation -->
   <script>
      function validateProductName() {
      var productNameInput = document.getElementById("productName");
      var productName = productNameInput.value;
      var productNameError = document.getElementById("productNameError");

      // Validate product name
      var lettersOnlyRegex = /^[a-zA-Z]+$/;

      if (!lettersOnlyRegex.test(productName)) {
        productNameError.textContent = "Product name should contain only letters.";
      } else {
        productNameError.textContent = "";
      }
     }
    // Add an event listener to call the validation function on input
    document.getElementById("productName").addEventListener("input", validateProductName);



    function validateProductDescription() {
    var productDescriptionInput = document.getElementById("productDescription");
    var productDescription = productDescriptionInput.value;
    var productDescriptionError = document.getElementById("productDescriptionError");

    // Validate product description
    var lettersOnlyRegex = /^[a-zA-Z]+$/;

    if (!lettersOnlyRegex.test(productDescription) || productDescription.length > 100) {
      productDescriptionError.textContent = "Product description should contain only letters and
have a maximum length of 200 characters.";
    } else {
      productDescriptionError.textContent = "";
    }
  }

  // Add an event listener to call the validation function on input
  document.getElementById("productDescription").addEventListener("input",
validateProductDescription);



    function validateMaterial() {
    var materialInput = document.getElementById("material");
    var material = materialInput.value;
    var materialError = document.getElementById("materialError");

    // Validate material
    var lettersOnlyRegex = /^[a-zA-Z]+$/;
    if (!lettersOnlyRegex.test(material)) {
      materialError.textContent = "Material should contain only letters.";
    } else {
      materialError.textContent = "";
    }
  }

  // Add an event listener to call the validation function on input
  document.getElementById("material").addEventListener("input", validateMaterial);



    function validatePrice() {
    var priceInput = document.getElementById("price");
    var price = priceInput.value;
    var priceError = document.getElementById("priceError");

    // Validate price as a positive number
    var priceRegex = /^\d+(\.\d{1,2})?$/;

    if (!priceRegex.test(price) || parseFloat(price) < 0) {
      priceError.textContent = "Please enter a valid positive price with up to two decimal places.";
      priceError.style.color = "red";
    } else {
      priceError.textContent = "";
    }
  }

  // Add an event listener to call the validation function on input
  document.getElementById("price").addEventListener("input", validatePrice);

    function validateStockQuantity() {
      var stockQuantity = document.getElementById("stockQuantity").value;
```

```
        var stockQuantityError = document.getElementById("stockQuantityError");

        // Validate stock quantity
        if (isNaN(stockQuantity) || parseInt(stockQuantity) < 0) {
          stockQuantityError.textContent = "Please enter a valid positive stock quantity.";
        } else {
          stockQuantityError.textContent = "";
        }
      }

      // Add similar functions for other input fields as needed
    </script>

  </body>
</html>
```

Add product -views.py

```
@login_required
def add_product(request):
    if request.method == 'POST':
        product_name = request.POST['product_name']
        product_description = request.POST['product_description']
        category = request.POST['category']
        sub_category = request.POST['sub_category']
        brand = request.POST['brand']
        age = request.POST['age']
        material = request.POST['material']
        price = request.POST['price']
        gender = request.POST['gender']
        seller_id = request.user.id  # Use the currently logged-in seller's ID
        stock_quantity = request.POST['stock_quantity']
        status = request.POST['status']
        product_image = request.FILES['product_image']
        seller = CustomUser.objects.get(id=seller_id)  # Get the seller using their ID

        product = Product(
            name=product_name,
            description=product_description,
            category_id=category,
            sub_category_id=sub_category,
            brand_id=brand,
            age_id=age,
            material=material,
            price=price,
            gender=gender,
            seller=seller,
            quantity = stock_quantity,
            status=status,
            product_image=product_image,
        )

        product.save()
        # You can add any additional logic here (e.g., sending a confirmation email)
        return HttpResponse('Product added successfully')  # You can customize the response as
needed

    sellers = CustomUser.objects.filter(user_types=CustomUser.SELLER)  # Filter sellers based on
user_types
    category=Category.objects.all()
    sub=Subcategory.objects.all()
    context = {'sellers': sellers ,'category':category,'sub':sub}
    return render(request, 'add_product.html', context)
```

3.Cart

```
{% load static %}
{% include "base.html" %}
{% block content %}
<html>
<head>
  <link rel="stylesheet" href="{% static 'css/cart.css' %}">
  <style>
    /* Your existing styles */
  .container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background-color: #f4f4f4;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  }

  .cart-header {
    text-align: center;
    margin-bottom: 20px;
    color: white;
  }

  .cart-item {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px;
    border-bottom: 2px solid #ddd;
  }

  .cart-item-details {
    display: flex;
    align-items: center;
  }
```

```
.cart-item-name {
   font-weight: bold;
   margin-right: 10px;
   flex: 1;
   white-space: nowrap;
   overflow: hidden;
   text-overflow: ellipsis;
}

.remove-from-cart-btn,
.quantity-btn {
   background-color: #ff6347;
   color: white;
   border: none;
   border-radius: 4px;
   padding: 5px 10px;
   cursor: pointer;
   transition: background-color 0.3s;
}

.remove-from-cart-btn:hover,
.quantity-btn:hover {
   background-color: #e74c3c;
}

.cart-item-quantity {
   display: flex;
   align-items: center;
   margin-right: 10px;
}

.cart-item-price {
   font-weight: bold;
}
```

```
.continue-shopping-link {
    display: inline-block;
    margin-top: 20px;
    text-decoration: none;
    color: #3498db;
    font-weight: bold;
}

.continue-shopping-link:hover {
    text-decoration: underline;
}

.checkout-button {
    display: inline-block;
    background-color: #28a745;
    color: white;
    padding: 10px 20px;
    border-radius: 4px;
    margin-top: 20px;
    margin-left: 90px;
    text-decoration: none;
    font-weight: bold;
    transition: background-color 0.3s;
}

.checkout-button:hover {
    background-color: #218838;
}

/* ... (other styles) */

.continue-shopping-link {
    display: inline-block;
    margin-top: 20px;
    text-decoration: none;
    color: #3498db;
    font-weight: bold;
    padding: 10px 20px;
    border: 2px solid #3498db;
    border-radius: 4px;
    transition: background-color 0.3s, color 0.3s, border-color 0.3s;
}

.continue-shopping-link:hover {
    background-color: #3498db;
    color: white;
    border-color: #3498db;
    text-decoration: none;
}

/* ... (other styles) */

/* try */
/* ... (other styles) */

.product-container {
    background-color: #dc7b7b;
    border: 1px solid #cf7f7f;
    border-radius: 8px;
    margin-bottom: 20px;
    transition: transform 0.3s ease-in-out;
}

.product-container:hover {
    transform: scale(1.05);
}
```

```
.card-body {
   padding: 20px;
}

.card-title {
   font-size: 1.5rem;
   margin-bottom: 10px;
   color: #333333;
}

.total-price {
   font-weight: bold;
   font-size: 1.25rem;
   color: #e44d26;
}

.cart-item-quantity {
   display: flex;
   align-items: center;
   margin-top: 10px;
}

.quantity-btn {
   background-color: #3498db;
   color: white;
   border: none;
   border-radius: 4px;
   padding: 5px 10px;
   cursor: pointer;
   transition: background-color 0.3s;
   margin-right: 5px;
}

.quantity-btn:hover {
        background-color: #2980b9;
    }

    .product-container {
    max-width: 800px; /* Set your desired medium width */
    margin: 0 auto;  /* Center the container */
    background-color: #ffffff;
    border: 1px solid #ddd;
    border-radius: 8px;
    margin-bottom: 20px;
    transition: transform 0.3s ease-in-out;
    }

    /* ... (other styles) */



    </style>
</head>

<body>
    <div class="container">
        <header>
            <h1 text-align="center">My Cart</h1>
        </header>

        <div class="row">
            <div class="col-md-8">
                <div class="product-details-container">
                    {% for item in cart_items %}
                        <div class="product-container card mb-3">
                            <div class="row g-0">
                                <div class="col-md-4">
                                    <img src="{{ item.product.product_image.url }}" alt="{{
item.product.name }}" style="max-width: 250px; height:210px;border: 2px solid #ddd;">
```

```
                </div>
                <div class="col-md-8">
                  <div class="card-body">
                    <h5 class="card-title">{{ item.product.name }}</h5>
                    <p class="total-price" id="total-price-{{ item.product.id }}">Total:
Rs.{{ item.total_price  }}</p>
                    <div class="cart-item-quantity">
                      <form action="{% url 'increase-cart-item' item.product.id %}"
method="post">
                        {% csrf_token %}
                        <button class="quantity-btn increase-quantity"
type="submit">+</button>
                      </form>
                      <span class="item-quantity">{{ item.quantity }}</span>
                      <form action="{% url 'decrease-cart-item' item.product.id %}"
method="post">
                        {% csrf_token %}
                        <button class="quantity-btn decrease-quantity" type="submit">-
</button>
                      </form>

                      <form method="post" action="{% url 'remove-from-cart'
item.product.id  %}">
                        {% csrf_token %}
                        <button type="remove-from-cart-btn" class="btn btn-danger"
onclick="showConfirmation({{item.product.id}})">Remove</button>
                      </form>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          {% endfor %}
        </div>
```

```
         </div>




         <div class="col-md-4">
            <div class="price-details-container">
               <div class="order-summary">
                  <h3>Price Details</h3>
                  <p class="total-price-data">Total amount : Rs.<span id="total-price">
{{total_amount}}</span></p>
                  <a class="continue-shopping-link"  href="{% url 'product_list' %}">Continue
Shopping</a>
                  <a class="checkout-button" href="{% url 'checkout' %}">Checkout</a>
               </div>
            </div>
         </div>
      </div>
   </div>

   <script src="{% static 'js/cart.js' %}"></script>
   <!-- Update the script block in your cart.html -->
<script>
   function showConfirmation(productId) {
      var confirmation = confirm("Are you sure you want to remove this product from the cart?");
      if (confirmation) {
         window.location.href = "{% url 'remove-from-cart' 0 %}".replace('0', productId);
      }
   }
</script>




</body>
</html>
```

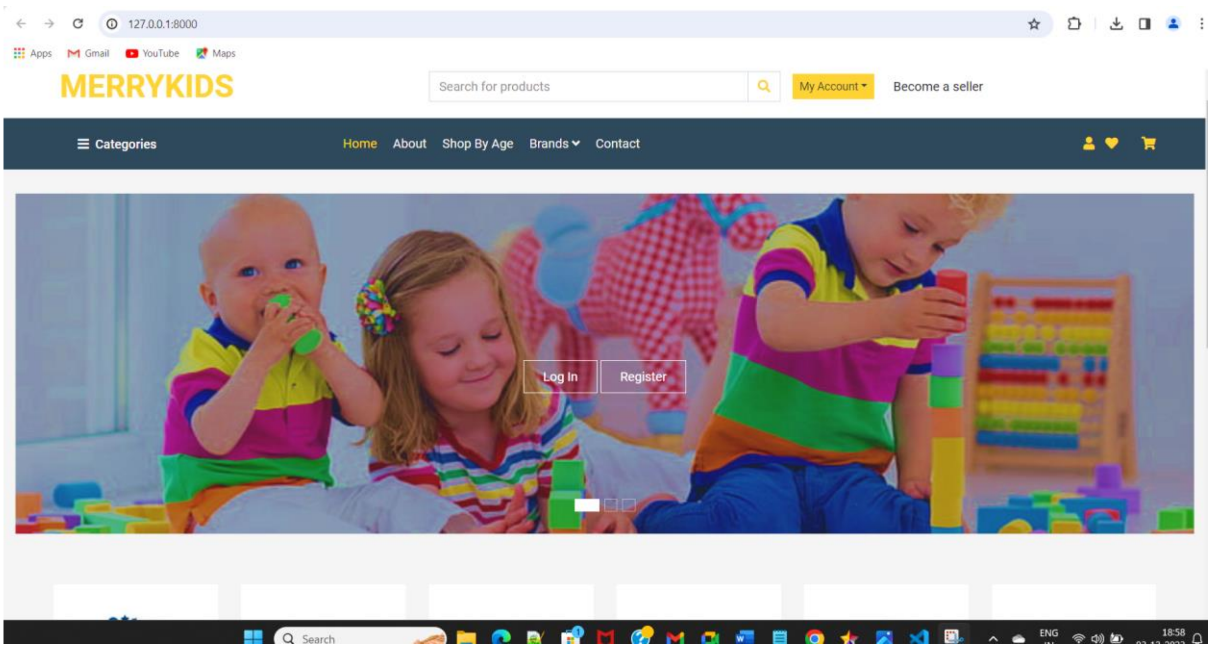
```
{% endblock %}
```

Cart Views.py

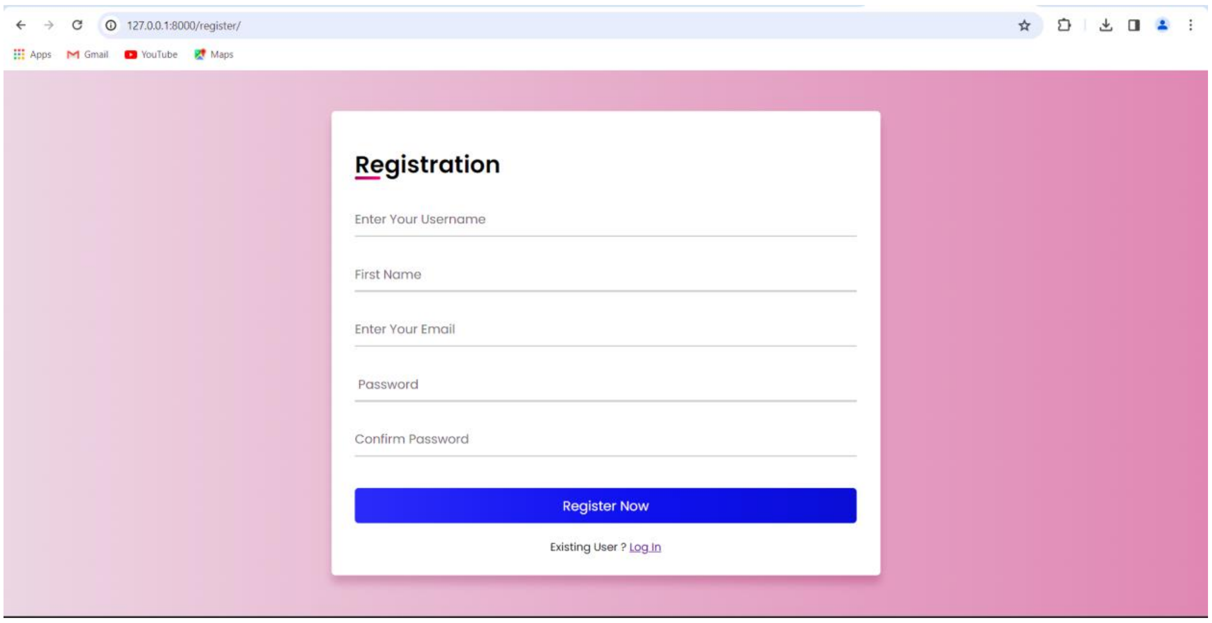
```
@login_required(login_url='login')
def add_to_cart(request, id):
    product = Product.objects.get(pk=id)  # Use correct model name 'Product'
    cart, created = Cart.objects.get_or_create(user=request.user)
    cart_item, item_created = CartItem.objects.get_or_create(cart=cart, product=product)

    # Check if the product is already in the cart
    if product_already_added_to_cart:
        return JsonResponse({'message': 'This product is already added to your cart.'})
    else:
        return JsonResponse({'message': 'Product added to your cart successfully.'})

    if not item_created:
        cart_item.quantity += 1
        cart_item.save()

    return redirect('cart')
```

9.2 Screen Shots

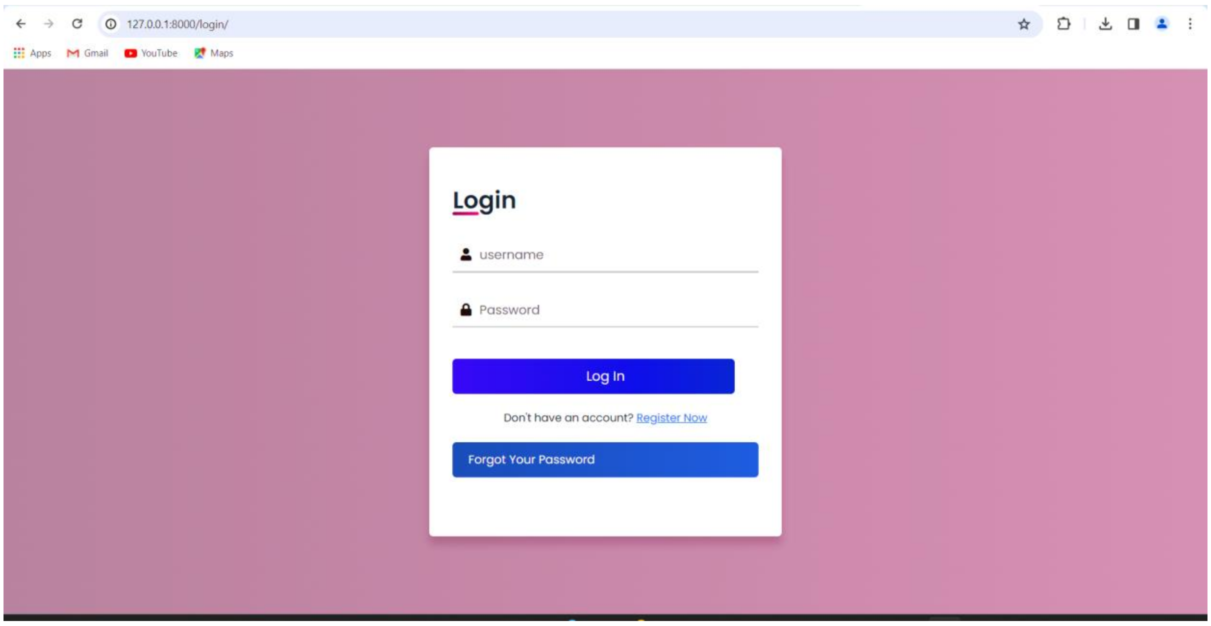
1. Index page



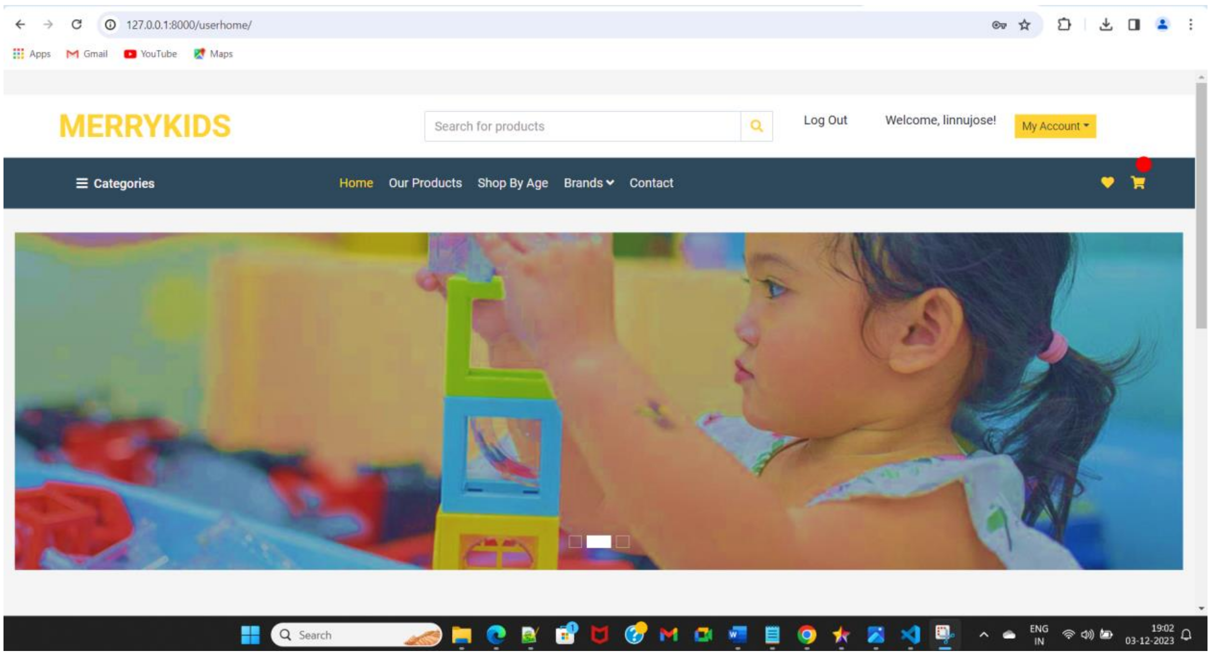
2. User Registration Form



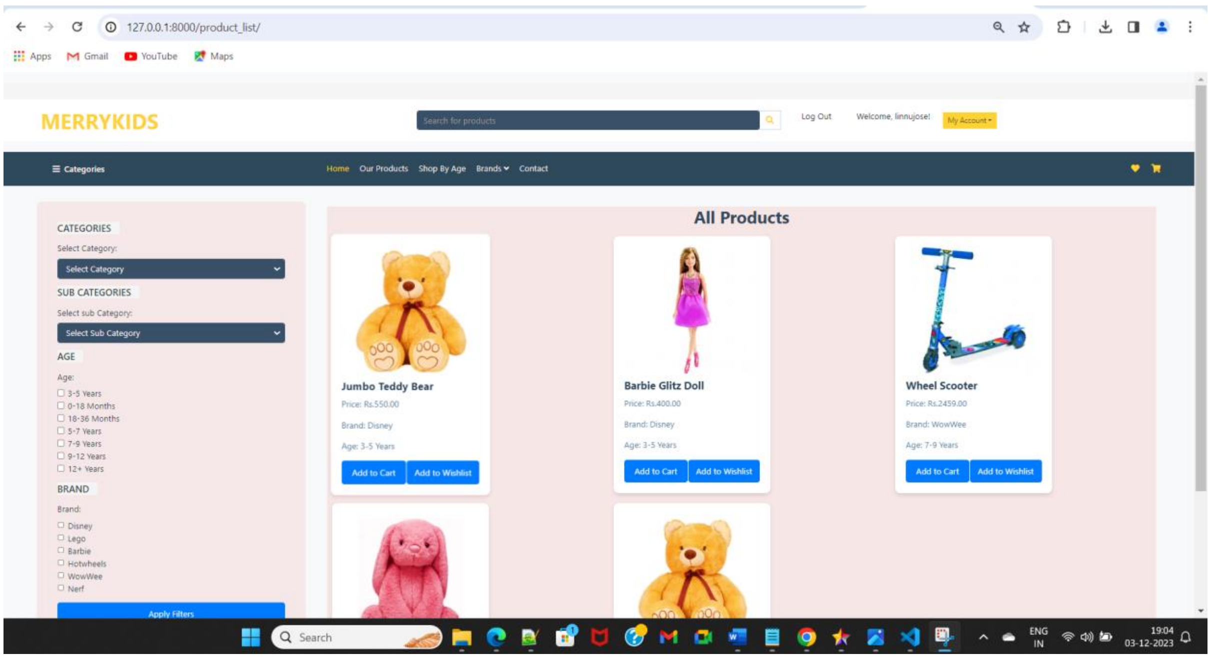
3. Login Form



4. User Home Page



5. Product List page



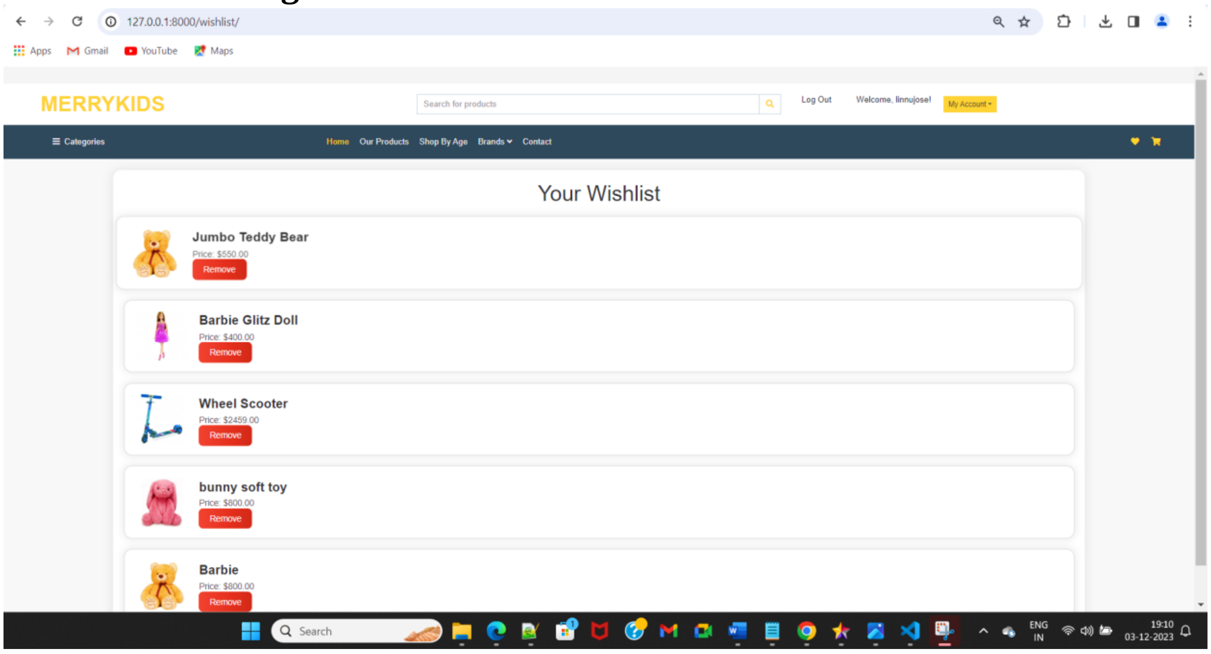
All Products

Jumbo Teddy Bear
Price: Rs.550.00
Brand: Disney
Age: 3-5 Years

Barbie Glitz Doll
Price: Rs.400.00
Brand: Disney
Age: 3-5 Years

Wheel Scooter
Price: Rs.2459.00
Brand: WowWee
Age: 7-9 Years

1. Wishlist Page



Your Wishlist

Jumbo Teddy Bear
Price: $550.00

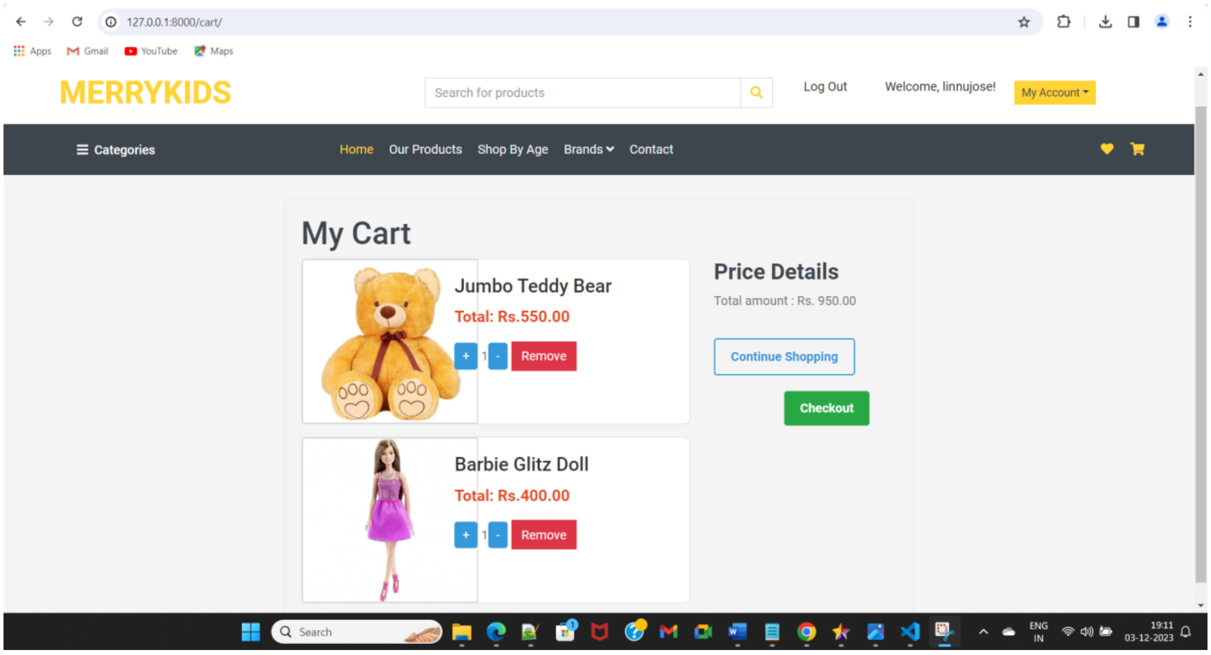
Barbie Glitz Doll
Price: $400.00

Wheel Scooter
Price: $2459.00

bunny soft toy
Price: $800.00

Barbie
Price: $800.00

2. Cart Page



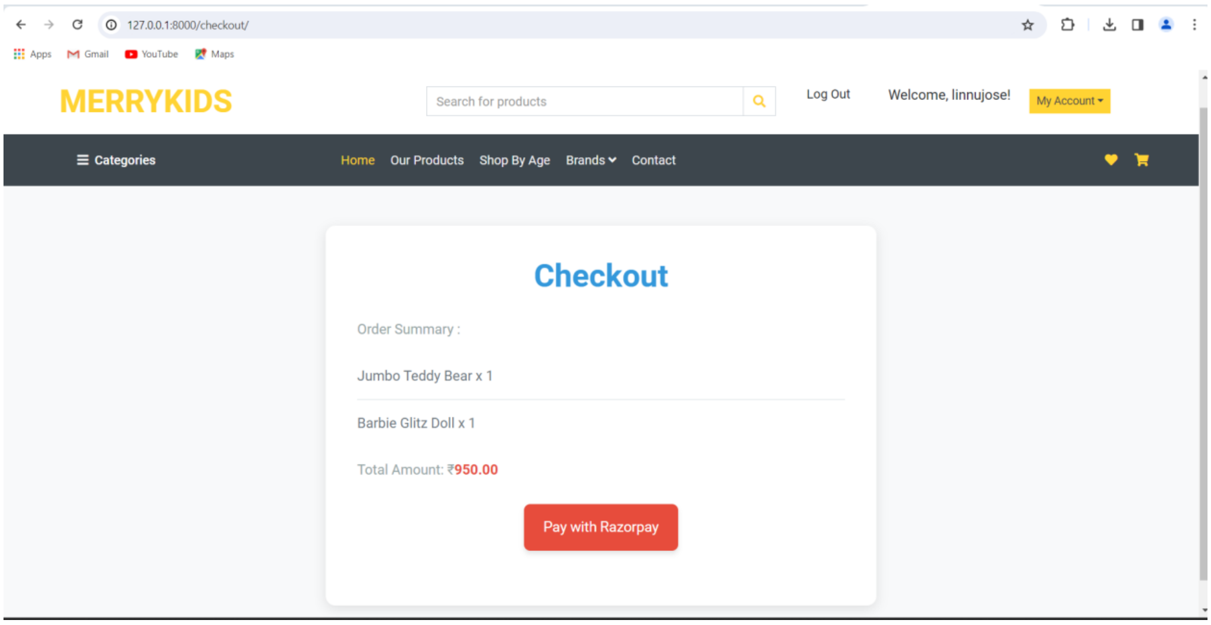
My Cart

Jumbo Teddy Bear
Total: Rs.550.00

Barbie Glitz Doll
Total: Rs.400.00

Price Details
Total amount : Rs. 950.00

3. Checkout Page



Checkout

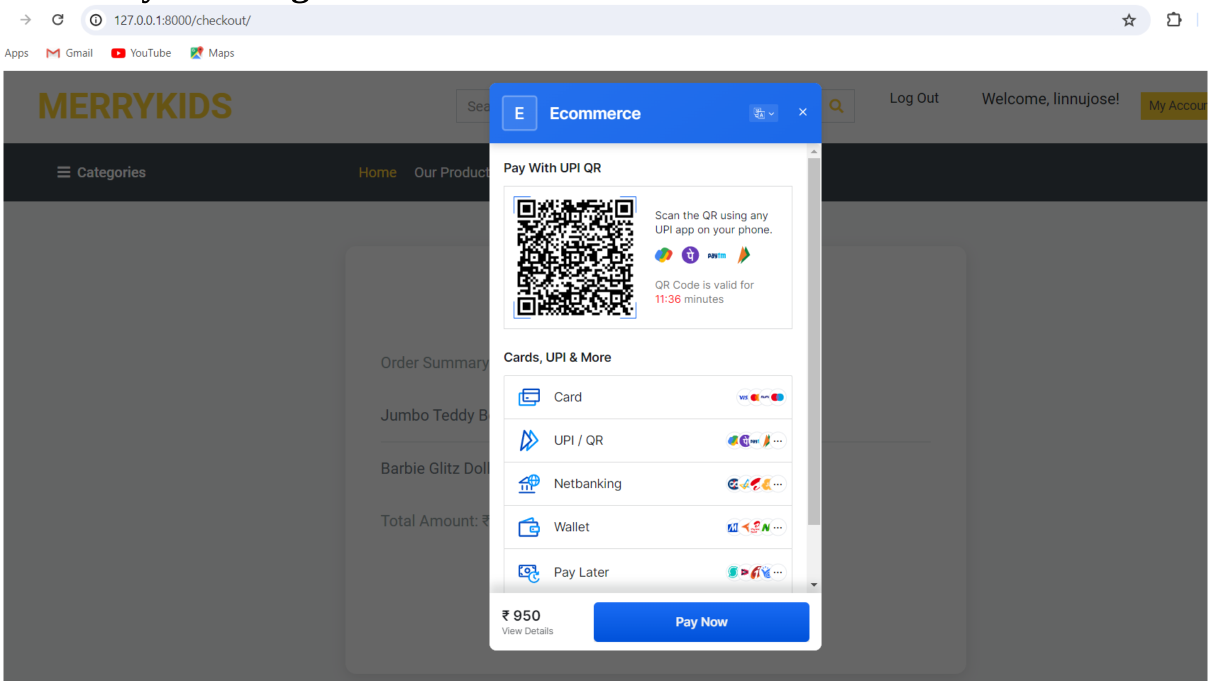
Order Summary :

Jumbo Teddy Bear x 1

Barbie Glitz Doll x 1

Total Amount: ₹950.00

4. Payment Page



Pay With UPI QR

Scan the QR using any UPI app on your phone.

QR Code is valid for 11:36 minutes

Cards, UPI & More

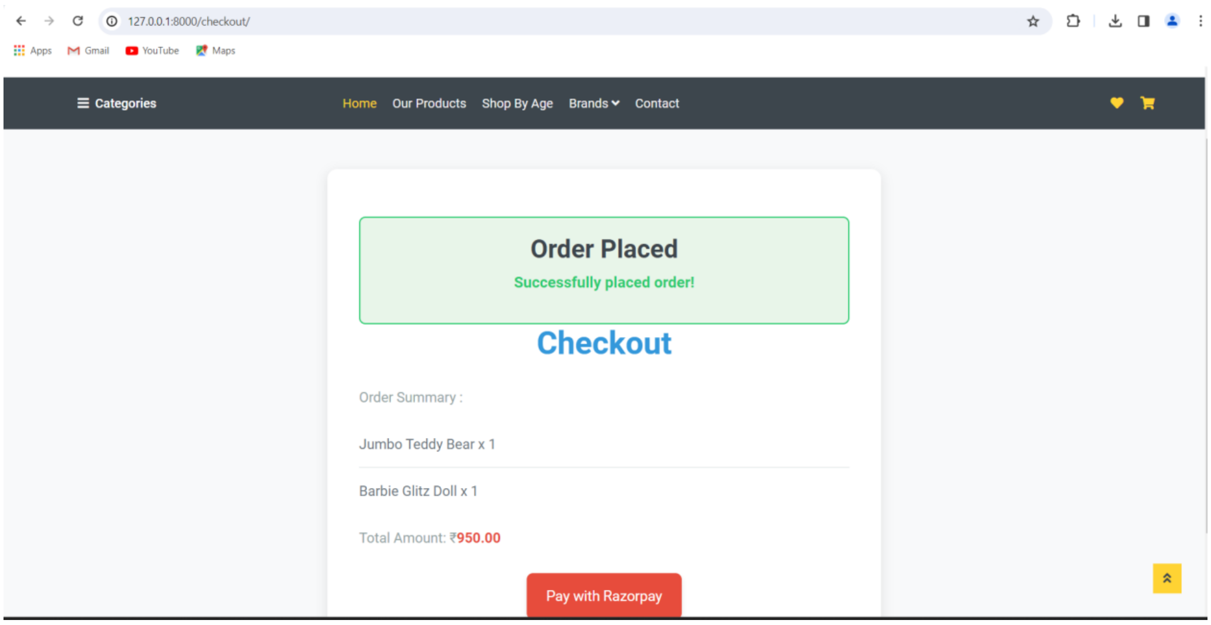
Order Summary

Jumbo Teddy B

Barbie Glitz Doll

Total Amount: ₹

₹ 950



Order Placed

Successfully placed order!

Checkout

Order Summary :

Jumbo Teddy Bear x 1

Barbie Glitz Doll x 1

Total Amount: ₹950.00

Attach Plagiarism Report