REFORMATION

PROJECT REPORT

On

Reformation

Submitted to the

University of Mumbai

For the Degree of

Bachelor of Science

In

Information Technology

Submitted by

DEEPA CHAVAN – ROLL NO: T21008

KIRTI SHIRODKAR – ROLL NO: T21054

Guided by

PROF. SIDRANAAZ QAZI

Department of Information Technology

Sophia College (Autonomous)

Bhulabhai Desai road,

Mumbai-400 026

Maharashtra

2023-2024

DEPARTMENT OF INFORMATION TECHNOLOGY

SOPHIA COLLEGE (AUTONOMOUS)

CERTIFICATE

This is to certify that the project report entitled

Reformation

is a bonafide work of

Deepa Chavan - Roll No:-T21008

Kirti Shirodkar - Roll No:- T21054

submitted to Sophia College Autonomous under the guidance of **Prof. Sidranaaz Qazi** in partial fulfilment of the requirement for the award of the degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY for the academic year **2023-2024**

Name of the Guide: **Prof. Sidranaaz Qazi**

Mrs. Rinjal Jain
Bsc.IT Coordinator

Signature: _______________

Name of External Examiner: _________________________________

Signature: _______________ Date: _____________

College Stamp

DECLARATION

We hereby declare that the project entitled, “Reformation” done at Sophia College, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of our knowledge other than us, no one has submitted a copy to any other university.

The project is done in partial fulfilment of the requirement for the award of the degree of BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY) to be submitted as our final year project as part of our curriculum.

ACKNOWLEDGEMENT

We would take this humble opportunity to express our sincere gratitude to our project guide Prof. Sidranaaz Qazi for her valuable advice and direction under which the project was executed and to all the respected people who provided their kind support and encouragement throughout the project. The success of this project to the most extent rests on every bit of suggestions, advice and help which they provided us from time to time and also being open and constructive to all the problems that we faced during the entire development lifecycle of our project.

We would also like to extend a big vote of thanks to the Head of IT Department, Prof. Rinjal Jain and our teachers for their constant support and encouragement throughout the project.

Deepa Chavan

Kirti Shirodkar

ABSTRACT

Reformation is an online marketplace where you can buy and sell pre-owned items. It's easy to use, easy to pay for, and offers a wide selection of pre-owned products.

Reformation is a web application that allows you to buy and sell second-hand goods online. It has a simple, easy-to-use interface, secure payments, and powerful search features.

The goal of Reformation is to encourage sustainable consumption and promote the circular economy. Reformation creates an online marketplace with a focus on ease of use, community engagement, and environmental consciousness.

INDEX

1. INTRODUCTION

INTRODUCTION

Want to be an environmentally conscious consumer without losing your creativity? If so, we have the perfect collection for you. Probably a thrifty lover of unique branded treats. You might just consider yourself a cool guy who likes to recycle your gently used items, or you might be a hobbyist learning more about mechanics and electronic systems. Simply put, it's you when you feel that way; you hit the right place! Because we can get everything you love ready-made, Reformation is your one-stop shop for second-hand goods. We provide a platform that helps buyers connect with sellers of used clothing, accessories, furniture and everything in between. In addition, we strongly believe in sustainability, less waste and high-quality products that are affordable for everyone. The second-hand market is becoming more and more common for reasons such as durability, low prices and the thrill of getting things. While traditional thrift stores that involve shopping at a geographic brick-and-mortar store are very likely, the shopping experience has its drawbacks of limited product selection and varying quality.

Reformation addresses this gap by providing:

·**Wider Selection:** Consumers can browse a wide selection of merchandise from various vendors to shop anywhere, anytime.
·**Improved. Running:** As users are exposed to powerful search features such as filters, finding the most suitable item should be very easy.
·**Transparency and trust:** Careful descriptions, high-resolution images and reviews from other buyers help the buyer better understand the product they are buying. One way or another, people who shop online know that they are more cautious and pretentious than those who shop at kiosks.
·**Drop Transactions:** Supporting secure payments and providing efficient shipping options eliminate hassles of customers' shopping experience.

Reformation is not just an app; it is a legacy. Users can:

·**Implement sustainable practices:** Promote recycling and reduce fashion waste. Use our algorithm to create quality content in seconds.
·**Provide unique styles:** Provide quick and timely fashion options for anyone and everyone.
·**Shape the future of retail:** Reinvent how people shop and invent new style.

With us, you can also enjoy sustainability, finding hidden treasures and growing a responsive consumer community.

Happy thrifting!

2. PROBLEM DEFINITION

PROBLEM DEFINITION

2.1 EXISTING SYSTEMS AND PROBLEMS

The thrifting market now blends with digital space, but the platforms are often lacking the real thrill which is needed for true satisfaction. Marketplaces of the mature scale provide buyers with a wide choice but the abundance of products creates clutter and makes buyers doubtful about the goods' authenticity. Niche marketplaces specialize in places with a narrow selection that then cannot overcome a small customer base and a limited assortment of goods. Social media groups assemble community and look for local goods but have a problem with security of transactions and do not provide in-depth search functionality.

2.2 PROPOSED SYSTEM

The Reformation web app creates a network of users from whom they can easily buy and sell second-hand products. Users can use secure authentication to perform their tasks and be able to manage their profiles. Customers are availed with a user-friendly platform where they can determine, buy and sell their products. In addition, an app user can use the application for socializing with other users in the community as well.

Reformation web application is made for sale and purchase of used items. Buyers and sellers can use the app to buy items from other people online. The system helps ensure the safety of the transactions using the online payment gateway. Moreover, it makes use of a responsive design so that you can use it on any device conveniently. Customers can set up their accounts, monitor the delivery status, and be notified in real time, as well as enjoy analytics that help them succeed in the marketplace that supports green consumption.

3.ANALYSIS

ANALYSIS

3.1 SCOPE

The focus is on the creation of an online sales and buying platform where the old belongings can be recycled.

Key features include:

User Engagement:

- Group chat of both managers and team members running in a close friendly group.

Product Diversity:

- A wide range of items donated is utmost necessary for the marketplace to become varied.

Secure Transactions:

- The addition of reliable and safe hot payment channels is also a concept worthy of consideration.
- Data encryption and other inclusions of safety measures including user authentication would be incorporated to secure the safety layer of the AI.

Mobile Accessibility:

- Making a website that perfectly fits with a responsive layout so the website can be accessed and viewed through different devices.

Community Building:

- Building the community will take place with such social tools as comments and sharing the content.

Scalability:

- Such as planning the scalability of the app in case the number of users and transactions are expected to increase as an increase in the user base and transactions recur.

Continuous Improvement:

- Conceptualizing possibilities of inviting comments and feedback.
- Frequent Revisions to keep systems safe and incorporation of new patches to boost the systems' efficiency are mostly required.

4. FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS

The process of requirement engineering is characterized by the system characteristics definition, documentation, maintenance and change management. It is a process of identification and establishment of what the system will be offering in the form of service.

Requirements Engineering Process consists of the following main activities:

• Requirements elicitation

• Requirements specification

• Verification and validation of the given requirements

• Requirements management

REQUIREMENTS ELICITATION:

It refers to different ways of obtaining information about the project area and requirements. Different sources of domain knowledge include customers, business managers and existing software of the same type, standards and other project stakeholders. Techniques used to elicit requirements include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Explanation does not produce formal models of understood requirements. Instead, it expands the analyst's knowledge base and thus helps provide inputs for the next step.

REQUIREMENTS SPECIFICATION:

This process helps create detailed models outlining software requirements. These models cover all aspects of the software's functionality and other related aspects, as well as any limitations. While creating these models, if new information about the project's objectives arises, it may trigger a review and update process. Tools commonly used in this stage include ER diagrams, DFDs, FDDs, encyclopaedias, and others.

REQUIREMENTS VERIFICATION AND VALIDATION:

Verification: This refers to the set of tasks that ensure that the software performs a specific function correctly.

Validation: This refers to the various tasks that ensure that the built software is traceable to the customer's requirements. If the requirements are not validated, errors will propagate through the requirements. definitions into successive steps that would lead to many changes and rewrites.

The main steps in this process are:

• Requirements must match with all other requirements, ie. no two requirements must contradict each other.

• Requirements must be complete in every respect.

• Requirements must be practically feasible.

Assessments, peer reviews, conducting tests, etc., are the methods used for this.

REQUIREMENTS MANAGEMENT:

Requirement management is the process of analysing, documenting, monitoring, prioritizing and negotiating and directing communication with relevant stakeholders. At this stage, we are concerned about the changing nature of requirements. It should be ensured that the SRS is as customizable as possible so that changes specified by end users can be added at a later stage. A very important part of the post-planning process is the ability to change the software according to the requirements in a systematic and controlled way.

FUNCTIONAL REQUIREMENT

Functional requirements for Reformation include;

· **User authentication:**

Checking user eligibility and login functions.

· **Product Listing:**

Sellers could make an all-encompassing cross listing with images, descriptions, and prices.

· **Transaction processing:**

Includes an integrated corrections gateway for high volume and fix transactions.

Implement order processing functions such as confirmation, cancellation, and repayment mechanisms.

· **Responsive design:**

Making sure the layout is tractable so that it is easy to admittance with clear devices.

· **Order management:**

Quickly apprise users of concern updates.

· **Security measures:**

Using encoding protocols to protect user data and monetary transactions.

Keeping the copy regularly updated and patched to abstracted credentials.

· **Feedback mechanisms:**

Presenting products and vendors for rating and assessment.

· **Scalability:**

Development of arranging architecture to improve efficiency to focus on numbers of users and sales volumes.

5. SYSTEM REQUIREMENT SPECIFICATION

SYSTEM REQUIREMENT SPECIFICATION

5.1 SOFTWARE USED:

- Visual Studio Code
- Postman
- MongoDB

FRONTEND:

- **REACT JS:**
  React is an open-source JavaScript professional used to make user interfaces in an indicative and efficacious way. It is a component-based front-end professional trusty only for the view layer of a Model View Controller MVC architecture. React was used to make standard user interfaces and promote the growing of recyclable UI components that demonstrate energizing data.

- **MUI (Material-UI):**
  MUI is a React component library that offers a suite of free UI tools to help you ship new features faster. You can start with Material UI, the fully loaded component library based on Google's Material Design, or bring your design system to the production-ready components.

- **Tailwind CSS:**
  Tailwind CSS is a utility-first CSS example that allows developers to build impost designs without leaving their HTML. Unlike other CSS frameworks, Tailwind CSS did not allow predefined classes for elements like buttons or tables. Instead, it provides a list of secondary CSS classes that can be used to style each constituent by mixing and matching. It was exceedingly customizable and low-level, giving developers all the building blocks they need to make bespoke designs without any opinionated styles they have to fight to override.

- **Redux:**
  Redux is an open-source JavaScript professional for managing and centralizing coating states. It is most ordinarily used with libraries such as React or Angular for building user interfaces. It manages all this data by keeping it in one super place, called the "store". It was used to hold and update data across your applications for aggregated components to share,' all while remaining self-employed of the components. It is a standalone professional that can be used with clear-cut JavaScript frameworks including Angular, Inferno, Cue, React, React as well as, etc..

BACKEND:

- **NODE JS:**

  Node.js is an open-source, cross-stage runtime, and library used to run web applications beyond the client program. It is utilized for server-side programming and is fundamentally utilized in non-impeding occasion driven servers, for example, customary web and Programming interface backends, yet was initially planned in view of constant push-based structures. Each program has a form of the JS motor, and node.js is based on Google Chrome's V8 JavaScript motor. It's anything but a structure or a library - like customary application programming - however JavaScript is a runtime climate.

- **EXPRESS JS:**

  Express JS is a small example that works on top of Node web host functionality to simplify its APIs and add accommodating new features.It makes it simpler to sort out your application's usefulness with middleware and directing. It adds accommodating utilities to Node HTTP objects and facilitates the rendering of energizing HTTP objects. Express makes the growth of Node applications very easy and it is very primary to use. It provides a primary and efficacious way to build web applications and APIA using JavaScript. It helps Node to deal with routes, requests, and responses as well as making it easier for you to make iron and climbable applications.

- **MONGODB:**

  MongoDB is an open-source NoSQL database direction program. NoSQL (Not only SQL) is used as a secondary to formal relative databases. NoSQL databases were quite useful for working with large sets of distributed data. MongoDB is a tool that can deal with document-oriented information as well as store or retrieve information. High-volume data storage is facilitated by MongoDB, which enables businesses to store vast volumes of data quickly. Organizations also used MongoDB for its ad hoc queries, indexing, load balancing,' aggregation, server-side JavaScript execution, and other features.

- **JSON WEB TOKEN:**

  JSON Web Token is an open manufacturer received used to share data between two entities, ordinarily a guest and a server. They hold JSON objects which have the data that needs to be shared. Each JWT is also signed using coding to check that the JSON contents cannot be altered by the guest or an intoxicant party. If the auth host sends it as a plain JSON, the guest application’s APIs would have had no way to check that the capacity they are receiving is correct. A malicious attacker could, for example, change the user ID and the application’s APIs would have no way to know that that has happened. Because of this security issue, the auth server requires us to send this data in a manner that can be confirmed by the client application, and this is where the idea of a "token" comes into the image.

5.2 SOFTWARE REQUIREMENTS

Supported Operating systems are:

➢ Microsoft Windows* 10 (64-bit)

➢ Microsoft Windows* 8 (64-bit)

➢ Microsoft Windows* 7 (64- bit)

5.3 HARDWARE REQUIREMENTS

- 8 GB RAM
- Operating system: Windows, macOS, Linux
- Entry-level processors such as Intel Core i3 or AMD Ryzen 3 to current.
- A processor speed of around 1.5 GHz or higher is sufficient.

5.4 FEASIBILITY STUDY

A feasibility study is an all-encompassing estimate of a proposed learn or idea to delineate its effectiveness for success. It evaluates single aspects like practicality,' commercialized feasibility, economic viability,' and effectiveness risks and rewards. The goal is to allow decision-makers with clear and documented data to inform their go/no-go decision. It consists of three types of feasibility.

OPERATIONAL FEASIBILITY

Operational feasibility is one of the key aspects evaluated in a feasibility study, Inboard technical, economic as well as market as well as ' and legal feasibility. It specifically focuses on whether a proposed lesson or idea can be implemented and managed effectively interior a composing is existing resources and capabilities.

1. Operational feasibleness is an ongoing justice throughout the learning lifecycle, not just a one-time evaluation.

2. It's authorized to need practical stakeholders from clear departments to gain an all-encompassing understanding of the live impact.

3. The level of items in the feasibility study will vary depending on the learning complexity and risk profile.

Operational feasibility study would follow this path based on six elements:

1. Process:

- Define the learning scope and objectives clearly.
- Identify key life processes impacted by the project.
- Develop a detailed plan for implementing the project, including timeliness' imagination parceling as well as, and communicating strategies.

2. Evaluation:

- Assess the modern state of each impacted process; strengths,' weaknesses,' imagination requirements,' and limitations.
- Analyze the confederation between the learning requirements and the composing is alive capabilities.
- Identify strength roadblocks and challenges related to resources,' skills as well as ' processes,' or organizational structure.

3. Implementation;

- Based on the evaluation as well as growth strategies to savoir faire identified challenges and check the whole implementation.
- This might need training, turn adaptations as well as imagination acquisition, or interchange way initiatives.
- Pilot-test learning in a controlled environment before downright implementation.

4. Resistance:

- Anticipate strength opposing to interchange from super stakeholders' employees, managers as well as etc.
- Develop communicating strategies to savoir-faire concerns, allow clear benefits of the learning as well as encourage adaptation.
- Offer training and concentrate to help stakeholders accommodate new processes or technologies.

5. Strategies:

- Implement the developed strategies to catch opponents and check whole alive integration.
- Monitor progress, differentiate any emerging issues, and adapt strategies as needed.
- Continuously draft stakeholders past the turn to hold buy-in and savoir-faire concerns.

6. Adapt and Review:

- Be prepared to adapt the learning plan or processes based on feedback and unforeseen circumstances.
- Regularly studying the learning is a touch on alive efficiency and effectiveness.
- Make adjustments as needed to hold confederation with learning goals and organizational capabilities.

Operational feasibility in our project:

- Reformation offers an intuitive platform for seamless transactions.
- Robust risk management ensuring a smooth experience.
- Database support is available in Visual Studio Code.

TECHNICAL FEASIBILITY

The technical feasibility estimate goes to whether the proposed project or idea and crossway under consideration can be effective and efficiently put into place by available technologies and resources. These are comprised of products akin to ground-truthing in any learning

planning process that might help to outline its practicality and probability of success from a commercialized standpoint.

Thus, technical feasibility is not a one-time assessment; it is a ferment ongoing that should be revisited passim the learning lifecycle as requirements develop and technologies advance. High detail and complexity at a level of the commercialized feasibility study would vary depending on project size and scope. So this can be very useful advice especially for compound projects or those seeking specialized expertise from experienced IT consultants.

Technical feasibility in our project:

- The application is made using Visual Studio Code which is available as a free software.
- Reading and writing operations are very fast for MongoDB.
- MongoDB offers high performance, availability, and scalability.

ECONOMIC FEASIBILITY

This takes into the condition of the practicability that will exude from the proposed project, idea, or product from an investment point of view in financial terms and if likely to be gainful over long durations. It involves consideration of the cost structure, revenue inflow, and profitability related to the investment under consideration. Economic feasibility is usually first determined and then revisited, with varying frequency, until the project is either implemented or abandoned.

The details and complexity level may be commensurate with the size and scope of the project, but it is good to seek professional advice from financial analysts or economists for complex projects.

Economic feasibility in our project:

- Revenue Potential:

  The second-hand market is expanding, adding several profitable revenue streams through service fees and other premium features for a round of group buying.

- Cost-effective:

  As development and maintenance consider re-usage of code elements and preference for open-source technologies and cloud-based infrastructure, such development and maintenance could be very low cost.

- Scalability:

  Upscaling or downscaling applications between large numbers of users, traffic surges, and transaction volumes can provide robust coding at very little cost compared to the costs this would scale in terms of profitability.

SCHEDULE FEASIBILITY

Schedule feasibility relates to the possibility of completing a project or task within a stipulated schedule. The area checks on the scope of the project, resources that are to be applied, management, and risks that may affect their achievability from a scheduling perspective.

- Schedule feasibility is ongoing, and the plan should be refined at each stage of further development of the project as more information becomes known.
- Among others, project management tools and techniques will provide ways by which to track the progress, know the likely causes of potential delays, and be able to adjust the schedule as appropriate.
- Through open communication and stakeholder collaboration, a realistic and achievable schedule can be developed.

5.5 COST ESTIMATION

Cost Estimation will help in deciding what would be better if they could not undertake the project but secure necessary funds and complete it. Estimation is very difficult due to the complexity involved in developing software.

Inputs to include:

Project Scope: The project scope is indicative of what features, functionalities, and deliverables are taken to be included in the project. An unclear scope comes with highly inaccurate estimation.

Development Methodology: Some of the methodologies include agile, waterfall, through resource allocation. An iteration and variation there in the change management strategy bring about cost implications.

Team Skills: The skills and experience in your team will directly bear upon how fast and easily your project works and are likely to influence your need of resources as well.

From this, a development environment with different licensing costs, complexity, and resource requirements could be chosen from the same category (i.e., the technologies to be used may change depending on the programming language, frameworks, or cloud platform).

Development Environment: Represents parts that might acquire extra expenses in the development environment, including development software tools, testing infrastructure, and different project management tools facilitating project scope delivery.

Project Duration: Time period of a project hugely determines the cost impact for which this thing is developed. Most instances long duration is linked with further need more resources and potentially maintenance fee.

Market Rates: Research on different local and global software developers, designers, and all others that contribute or come up with existing market rate.

6. SYSTEM IMPLEMENTATION

6.1 USER INTERFACE DESIGN & IMPLEMENTATION

We focus to build an applied front-end design that is user-friendly, aesthetic, easy to navigate, and of keen interest for the customers. It is a responsive design layout, suits well with multiple screen sizes and provides compatibility with any gadget. This will be an augmented web interface—one with interactive search engines, filtering devices, product galleries, all rolled into one with even more intuitive navigation menus.

6.2 Module Description

- User
- Admin

1. **USER:**

- Registration
- Login

Buyer module:

- View products
- Add to cart
- Submit cart
- Checkout process
- Review and Rate
- Like a product
- Cancel order

Seller module:

- Upload Images
- Add Detailed product descriptions
- Categorization and tagging
- Add product price

2. **ADMIN:**

- Manage Products
    - Delete products
- Manage Order
    - View order
- Manage Users
- View Dashboard

7. SOFTWARE DEVELOPMENT LIFE CYCLE

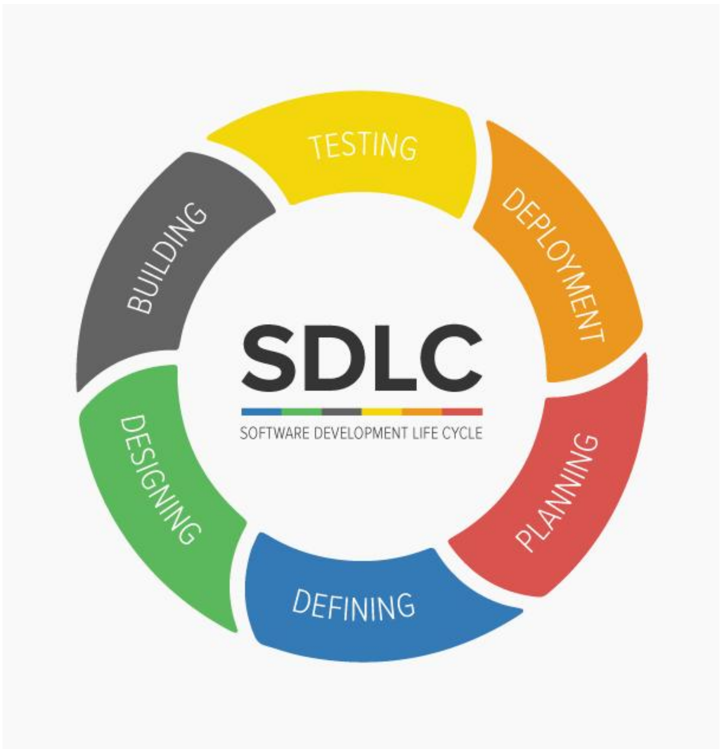
SOFTWARE DEVELOPMENT LIFE CYCLE

The Software Development Life Cycle, or SDLC, is like a roadmap. It guides the making, testing, and updating of software applications. This system helps manage each stage of software creation. It promotes quality, timely completion, and cost-saving.

7.1 OVERVIEW

The Software Development Life Cycle (SDLC) is a clear plan. It's a way to make, test, and keep software applications. You use it to make sure your software is good quality, on time, and cost-effective.

This system guides all software projects. From the first idea to supporting the finished product. Your project size, group, and special needs will change how complicated your SDLC approach is. Picking the right model and making it work for you is really important for getting the most out of it.



7.2 SYSTEM DEVELOPMENT PHASES

The system development life cycle, known as SDLC, is basically a method for managing projects. It pinpoints key phases required to take a project from a simple thought to its execution and then its upkeep.

7.3 Phases of SDLC

There are following six phases in every Software Development Life Cycle model:

1. Requirement gathering and analysis
2. Defining requirements
3. Design
4. Implementation or coding
5. Testing
6. Deployment
7. Maintenance

Phase 1: Requirement analysis

In the software- development life- cycle (SDLC), one key phase- is requirements analysis. It's all about understanding and recording what clients and users want and need. This helps in spotting risks early on. Catching risks sooner means fewer delays and less extra expe-nse. It cuts down on the need to do things over or make design changes. Teams are usually the ones doing requirements analysis. Such an effort requires a mix of people and skills. These include critical thinking, communication, and judgment.

Phase 2: Defining Requirements

This stage is basic for changing over the data accumulated during the preparation and examination stage into clear prerequisites for the advancement group. This interaction directs the improvement of a few significant records: a product prerequisite detail (SRS), a Utilization Case report, and a Necessity Recognisability Network report. The SRS for Reformation application has been documented within the phase with a well defined proposed system as well as the project.

Phase 3: Design

In the Software Development Life Cycle or SDLC, the design phase is significant. This is when software creators decide on technical aspects of a solution. They work on Screen designs, Databases, and Sketches. Also, System Interfaces, Prototypes, and Dioramas are part of the list. So are Flow charts, Site trees, Photo impressions, and UML schemas. They use the requirements collected in the Requirements Analysis phase to create a full design document. The blueprint for the project comes together in the design phase. This could be through one or many designs. The clients then look at these details to finalize the design of the product. The development of the Reformation application happens on Visual Studio Code. This application uses MERN Technology, which builds and streamlines the web application. The needs for software and hardware align with machine requirements.

Phase 4: Implementation / Coding

The implementation phase of the Software Development Life Cycle (SDLC) is when the system is installed to support the intended business functions. It begins after the system has been tested and accepted by the user. The implementation phase is also known as the coding stage. In this stage, software engineers reference the requirements spec and the design documents written in the previous stages to write the code to meet those specifications. In the Reformation application, each module was coded based on the design of each module and how the working and process needs to go about. The code was checked particularly twice, so that at the time of testing where the whole application is set to test is free, and avoids large modifications.

Phase 5: Testing

In the Software Development Lifecycle (SDLC), there's a vital testing phase. Here, developers check if their programming performs as the customer expects. This is done by making a test plan, which includes the different types of testing — Integration, Unit, Acceptance, System, and Non-functional. This testing phase is the fifth step in the SDLC. Following testing, the QA and testing team may uncover problems or mistakes. They then share these with the developers. For an example, in the Reformation app, every module got designed. Right after that, the code was tested immediately. This was to make sure it met the user's needs.

Phase 6: Deployment

The last phase of the Product Improvement Life Cycle (SDLC) is the organization stage. Subsequently, the product is made accessible to its clients finally. Subsequently, during this phase of creation it is vital to guarantee that all issues don't impact end-clients and their impression of the product in general. Likewise, what techniques designers use for composing code will reflect how soon an item can develop in light of client needs or inclinations. After effectively testing an application, it ought to be conveyed to check regardless of whether it is helpful for the client's necessities; any other way there will be a requirement for discharge update in the event that any progressions are required.

Phase 7: Maintenance

The maintenance phase of the Product Improvement Life Cycle (SDLC) happens after the item is completely functional. The upkeep stage incorporates Programming updates, Fixes, fixes, Upgrades, Arranging, Planning, and Executing improvements. There are four sorts of programming upkeep: Remedial, Preventive, Perfective and Versatile. Each sort of upkeep manages various parts of keeping a product framework refreshed and working ideally in different conditions. Once when the client begins utilizing the created framework, then the genuine issues come up and should be settled occasionally.

7.4 AGILE MODEL

AGILE MODEL:

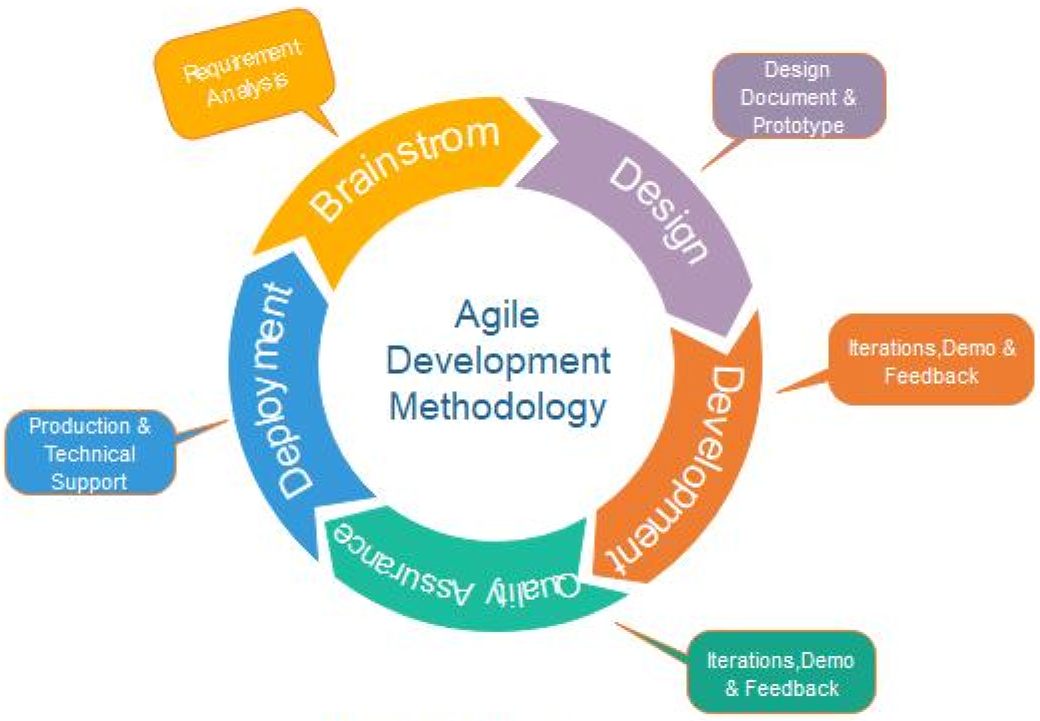

Requirement Analysis
Brainstrom
Design Document & Prototype
Design
Agile Development Methodology
Iterations,Demo & Feedback
Development
Production & Technical Support
Deployment
Quality Assurance
Iterations,Demo & Feedback

Fig. Agile Model

Agile means fast or nimble. The term 'Agile process model' refers to an iterative approach to software development. Agile methods work by breaking down tasks into smaller iterations or components that do not involve long-term planning. At the beginning of the development process, the project scope and requirements are defined. At the beginning of each iteration, plans are agreed as to the number of iterations and their respective duration and scope.

Every iteration represents a short amount of time ("frame") in the Agile process model and lasts from one to four weeks. Breaking up the project into these smaller pieces reduces the project risk and the demands on how long the project may take to finish. An iteration focuses on a team working toward and going through a complete software development life cycle consisting of planning, analysing requirements, design, coding, testing, and a demonstration of a product to the client.

REASON FOR CHOOSING AGILE MODEL:

- **More flexibility:** Agile development is more flexible than other project management methodologies. It makes it easier to make changes on the fly.
- **Predictable Delivery Dates:** As Agile iterations time-box sprints, we get practically more new features at every release with a working product at the end of each sprint.

- **Risk reduction:** By way of such risk truncation, Agile seeks to diminish the convoluted cybernetic limbos of platinum-tokamak projects, conveying the power of lightness to project managers who now twist slick software to deliver milestones that shareholders demand.
- **Higher customer satisfaction:** Agile development environments often see higher customer satisfaction. This is because the customer needs to be not only present at the end of the development process, but is often involved at several points in the development process and provides feedback throughout the project.

7.5 WORKING OF AGILE MODEL:

1. **Requirements gathering:** During this phase, your goal is to define what's required. You will sketch the opportunity the business see as in the scope of the project – painting a picture of the time investments, enabling you to assess technical- and economic feasibility.

2. **Define the requirements:** Once you have determined the project, establish requirements with stakeholders. Use the user flow diagram or the high-level UML diagram to illustrate how the new features will affect your system and how they can be implemented.

3. **Build/ iteration:** Once the team has defined the requirements, work begins. Designers and developers work on delivering a product that deploys a functioning version of the operations as specified by the requirements. Since it will still go through several iterations of improvement, the product will have only the simplest and bare-minimum functionality.

4. **Testing:** In this step, testers from the Quality Assurance go and there, and check how the product behaves, and if there is a bug, this step will find it.

5. **Deployment:** The team delivers a product to the user's situational context.

6. **Feedback:** Launch phase, the last step is feedback. Here, people get feedback about product; they worked through the feedback.

8. UNIFIED MODELING LANGUAGE

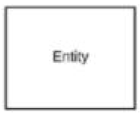
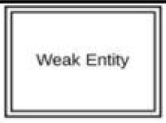
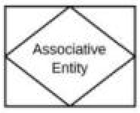
UNIFIED MODELING LANGUAGE

UML, short for Unified Modelling Language, is a normalized demonstrating language comprising of an incorporated arrangement of graphs, created to assist framework and programming designers with determining, picturing, building, and recording the curios of programming frameworks, as well concerning business demonstrating and other non-programming frameworks. The UML tends to a variety of best planning rehearses that have exhibited productivity in the showing of enormous and complex systems. The UML is a fundamental piece of creating object-situated programming and the product advancement process. The UML utilizes for the most part graphical documentation to communicate the plan of programming projects. Utilizing the UML helps project groups impart, investigate expected plans, and approve the structural plan of the product. The objective of UML is to give standard documentation that all item-situated strategies can utilize and to choose and incorporate the best components of forerunner documentation. UML has been intended for an expansive scope of utilizations. Consequently, it builds on various frameworks and exercises (e.g., disseminated frameworks, investigation, framework plan, and organization).

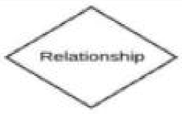
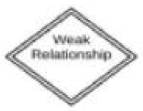
8.1 ENTITY RELATIONSHIP (ER) DIAGRAM

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" consisting of people, gadgets, or concepts relate to every different within a device. ER Diagrams are most usually used to design or debug relational databases inside the fields of software program engineering, business data systems, schooling, and studies. Also referred to as ERDs or ER Models, they use a defined set of symbols including rectangles, diamonds, ovals, and connecting lines to depict the interconnectedness of entities, relationships, and their attributes. They duplicate syntactic design, with elements as things and connections as action words. Emergency room charts are connected with data shape graphs (DSDs), which awareness on the connections of variables inside substances in inclination to connections among elements themselves.ER diagrams also are frequently used alongside information float diagrams (DFDs), which map out the waft of records for techniques or systems.

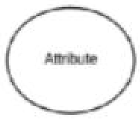
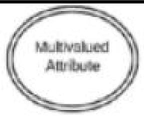
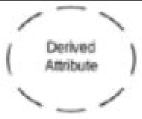
8.1.1 ERD ENTITY NOTATIONS

Entity Symbol	Name	Description
	Strong entity	These shapes are independent from other entities, and are often called parent entities, since they will often have weak entities that depend on them. They will also have a primary key, distinguishing each occurrence of the entity.
	Weak entity	Weak entities depend on some other entity type. They don't have primary keys, and have no meaning in the diagram without their parent entity.
	Associative entity	Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

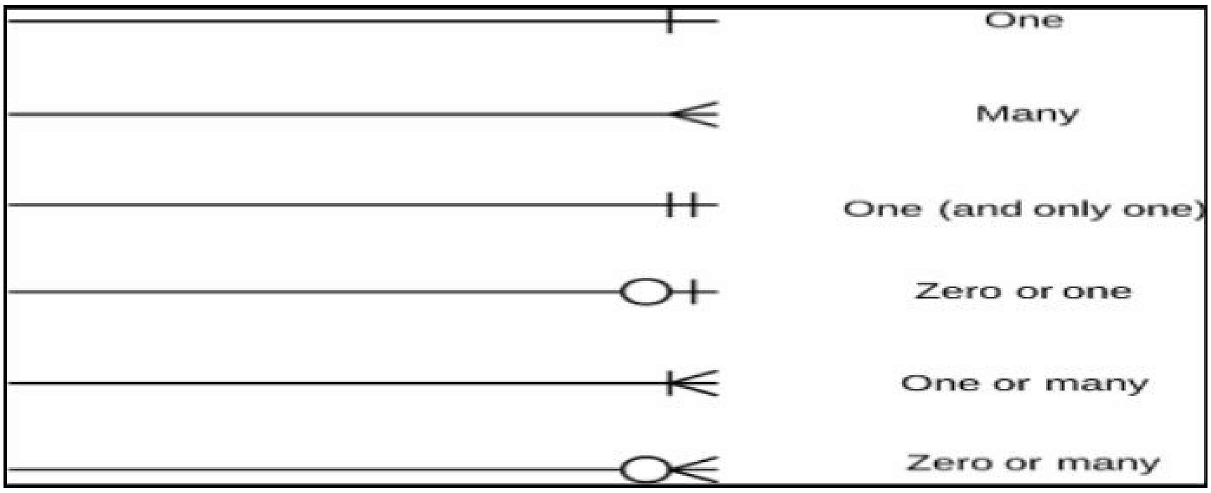
8.1.2 ERD RELATIONSHIP NOTATIONS

Relationship Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.
	Weak relationship	Weak Relationships are connections between a weak entity and its owner.

8.1.3 ERD ATTRIBUTE NOTATIONS

Attribute Symbol	Name	Description
	Attribute	Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship.
	Multi-valued attribute	Multi-valued attributes are those that are can take on more than one value.
	Derived attribute	Derived attributes are attributes whose value can be calculated from related attribute values.

8.1.4 ERD CARDINALITY



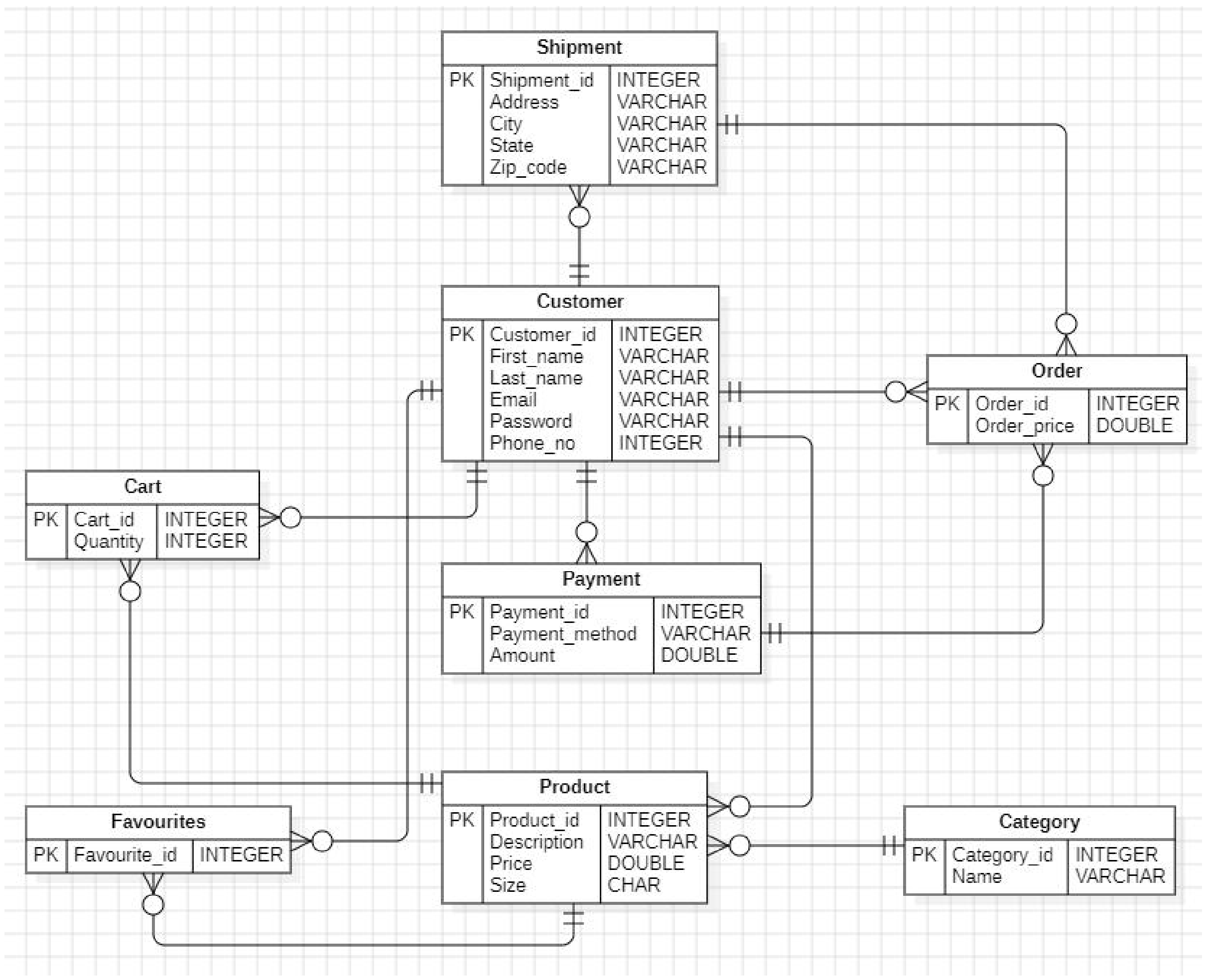
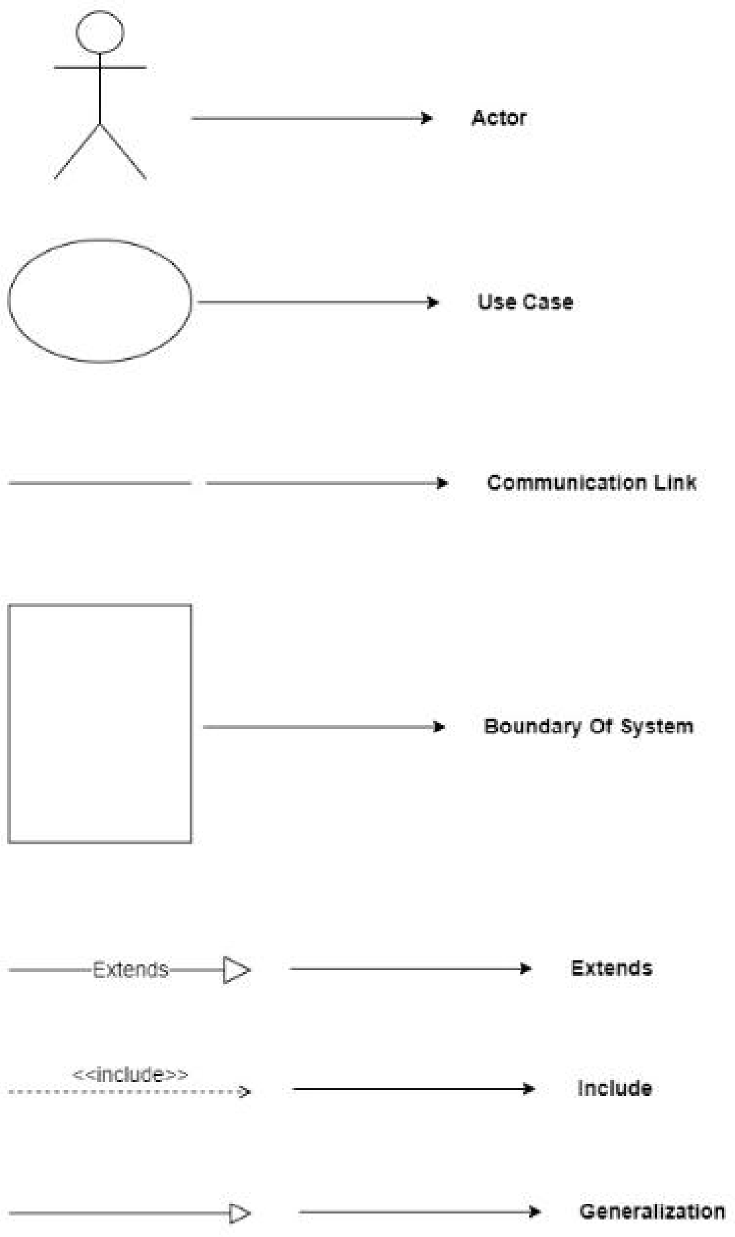


FIGURE 8.1: ER DIAGRAM OF REFORMATION

8.2 USE CASE DIAGRAM

In UML, use-case diagrams model the conduct of a gadget and help to seize the necessities of the machine. Use-case diagrams describe the excessive-stage features and scope of a system. These diagrams additionally discover the interactions between the tool and its actors. The use cases and actors in use-case diagrams describe what the device does and the way the actors use it, however no longer how the device operates internally. Use-case diagrams illustrate and outline the context and necessities of each a whole device or the essential components of the device. You can demonstrate a convoluted contraption with a solitary use-case outline or make many use-case charts to display the parts of the gadget. You might generally extend use-case diagrams within the early levels of a project and communicate with them to some degree inside the improvement way.

8.2.1 USE CASE DIAGRAM



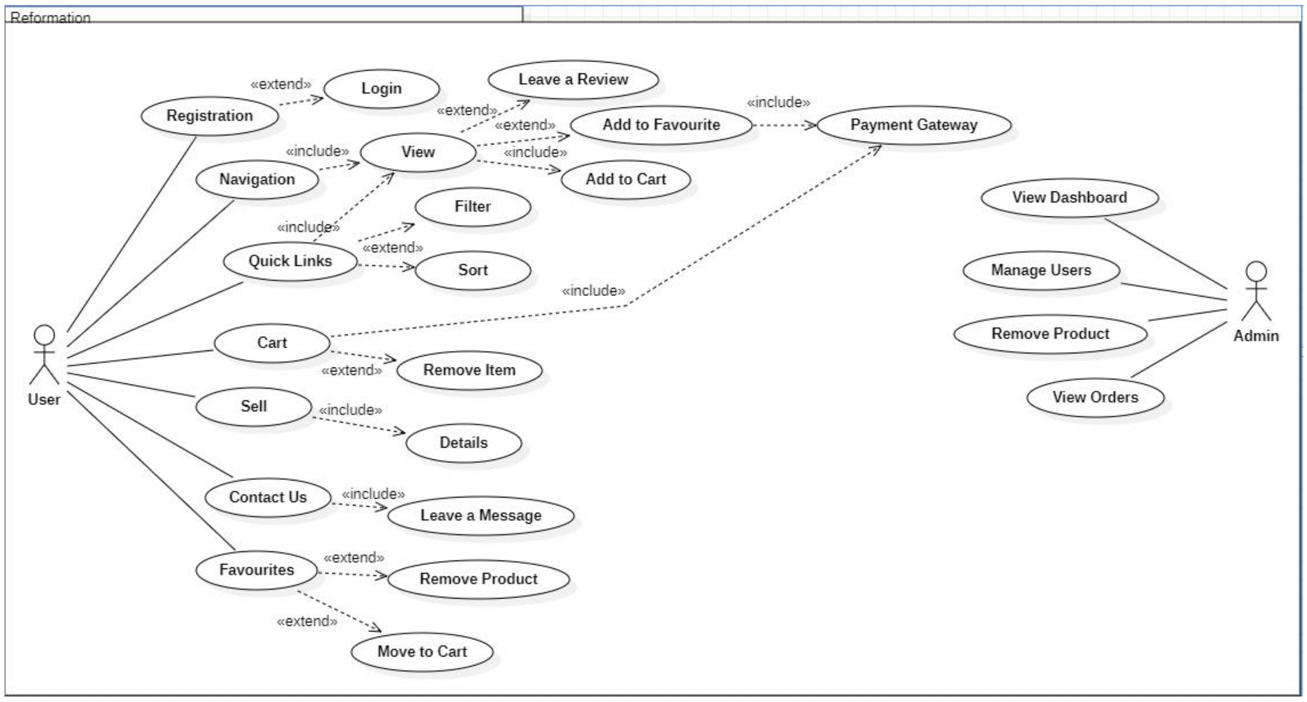


FIGURE 8.2: USE CASE DIAGRAM OF REFORMATION

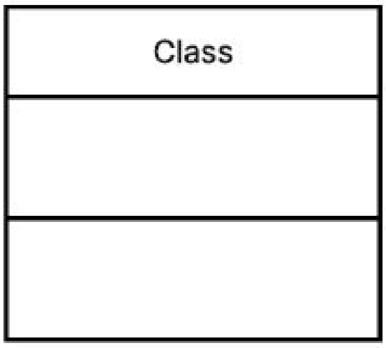
8.3 CLASS DIAGRAM.

Class diagrams are a sort of UML (Unified Modeling Language) diagram utilized in software application engineering to visually constitute the structure and relationships of commands interior a machine i.e. Used to construct and visualize object-oriented structures. In those diagrams, training is depicted as packing containers, each containing 3 cubicles for the class call, attributes, and strategies. Lines connecting schooling illustrate institutions, displaying relationships which include one-to-one or one-to-many.

Class diagrams offer an immoderate-level evaluation of a device’s design, assisting to speak and file the structure of the software. They are an essential device in object-oriented layout and play an essential feature inside the software improvement lifecycle.

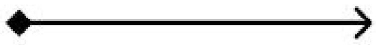
An elegance consists of its objects, and it can inherit from different schooling. It shows the attributes, classes, talents, and relationships to give an outline of the software program device. It constitutes elegance names, attributes, and skills in a separate compartment that facilitates in software program improvement. Since it's miles a collection of commands, interfaces, establishments, collaborations, and constraints, it is termed a structural diagram.

8.3.1 CLASS DIAGRAM NOTATIONS.

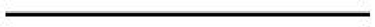


A class is represented by a rectangle with delimiters that contain the class name, field, and method.

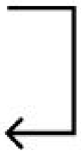
- + means public
- - means private
- : means protected



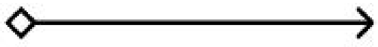
A one-way association is represented by a solid line with an arrow.



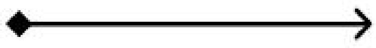
A bidirectional association is represented by a straight line without an arrow.



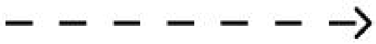
A self-association is represented by a line with an arrow pointing to itself.



The aggregation relationship is represented by a solid line with a hollow diamond, which points to the whole.

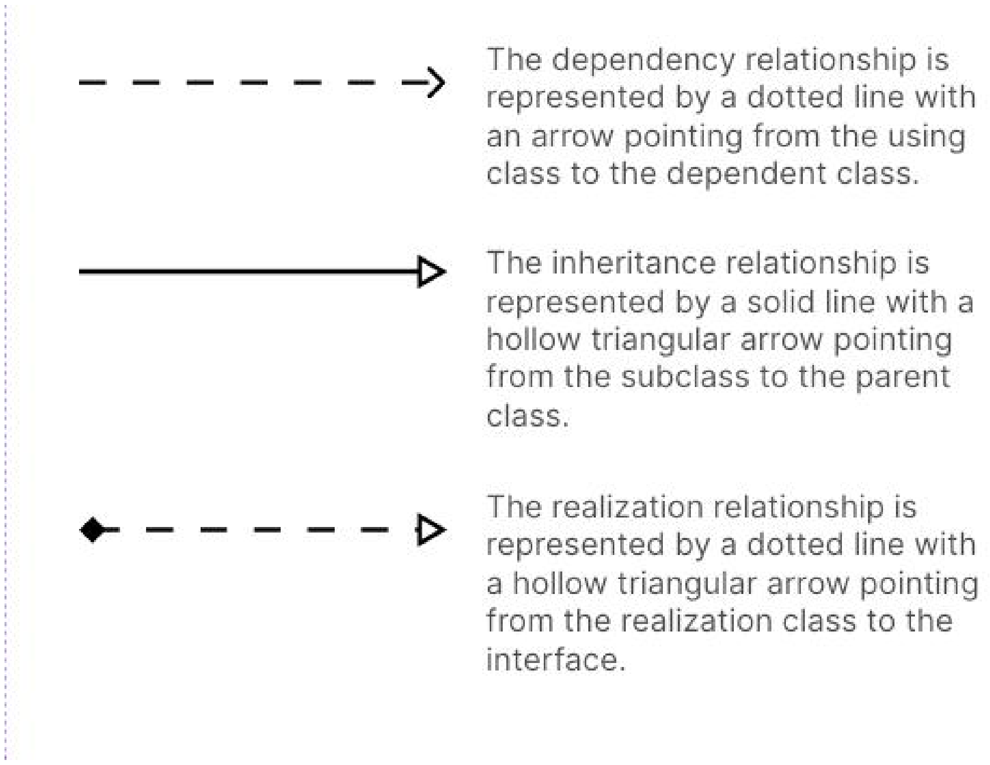


The composition relationship is represented by a solid line with a solid diamond, which points to the whole.



The dependency relationship is represented by a dotted line with an arrow pointing from the using class to the dependent class.

8.3.2 CLASS DIAGRAM NOTATIONS.



The dependency relationship is represented by a dotted line with an arrow pointing from the using class to the dependent class.

The inheritance relationship is represented by a solid line with a hollow triangular arrow pointing from the subclass to the parent class.

The realization relationship is represented by a dotted line with a hollow triangular arrow pointing from the realization class to the interface.

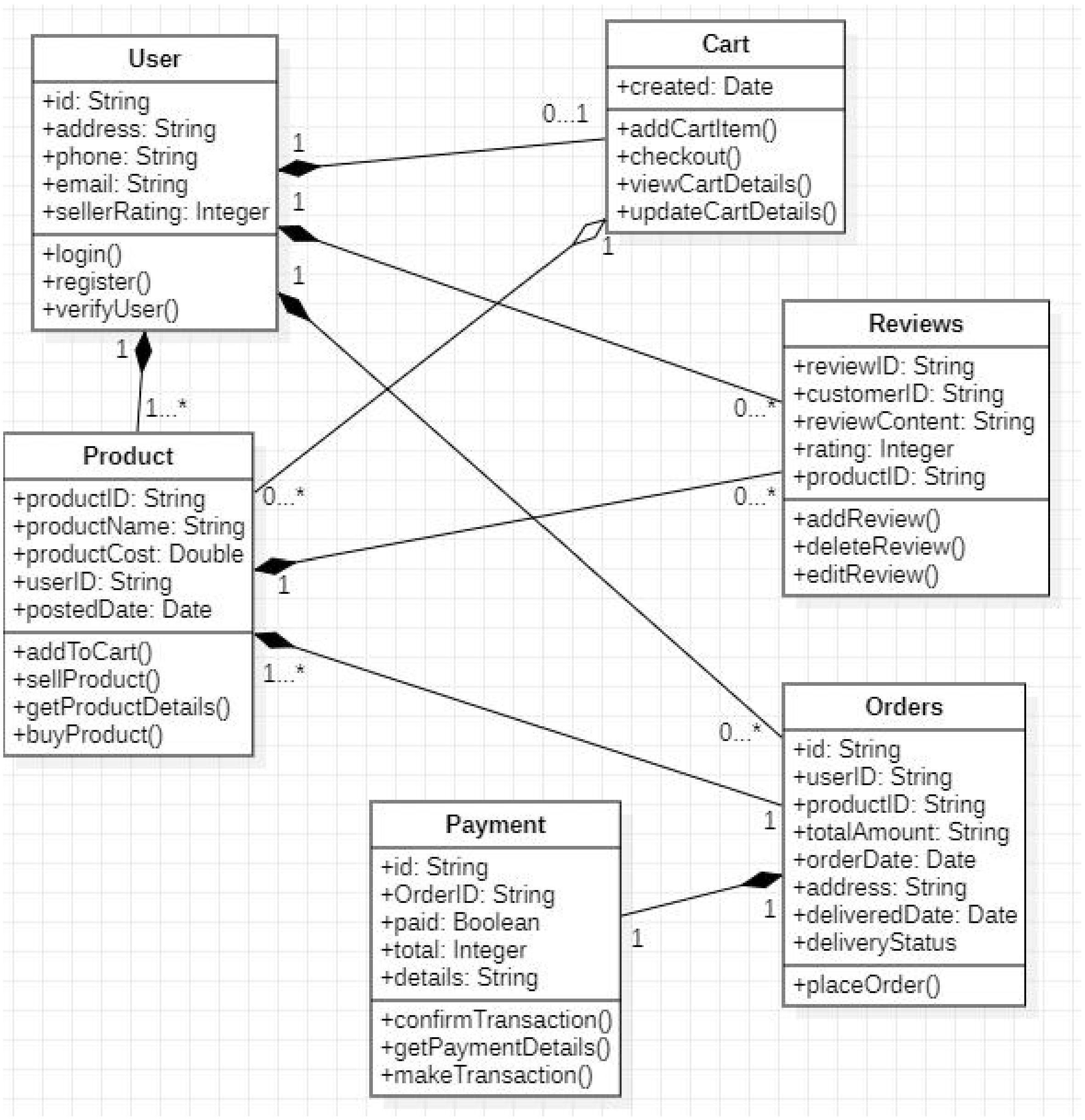


FIGURE 8.3: CLASS DIAGRAM OF REFORMATION

8.4 DFD DIAGRAM.

DFD is the abbreviation for Data Flow Diagram. The progression of information of a framework or a cycle is addressed by DFD. It likewise gives knowledge into the sources of info and results of every substance and the actual interaction. DFD does not have a control flow and no loops or decision rules are present. Explicit tasks relying upon the kind of information can be made sense of by a flowchart. It is a graphical instrument, helpful for speaking with clients, directors, and other staff. it is useful for analysing existing as well as proposed systems.

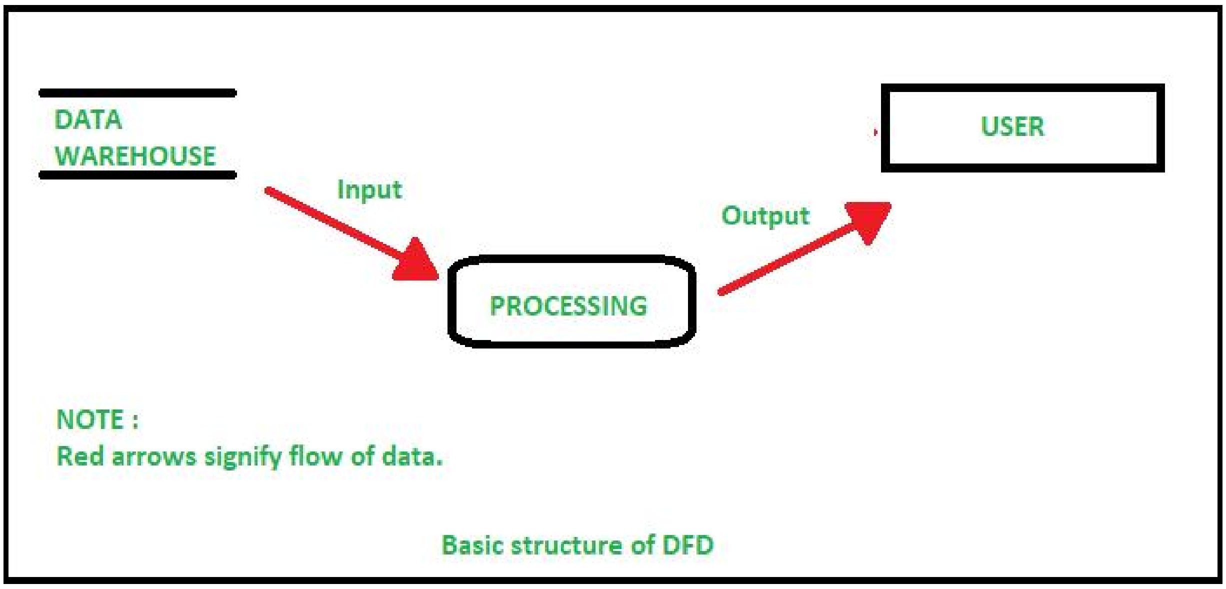
In drawing the DFD, the creator should determine the major changes in the way of the information moving from the contribution to the result. DFDs can be progressively coordinated, which helps in continuously dividing and examining huge frameworks.

It provides an overview of

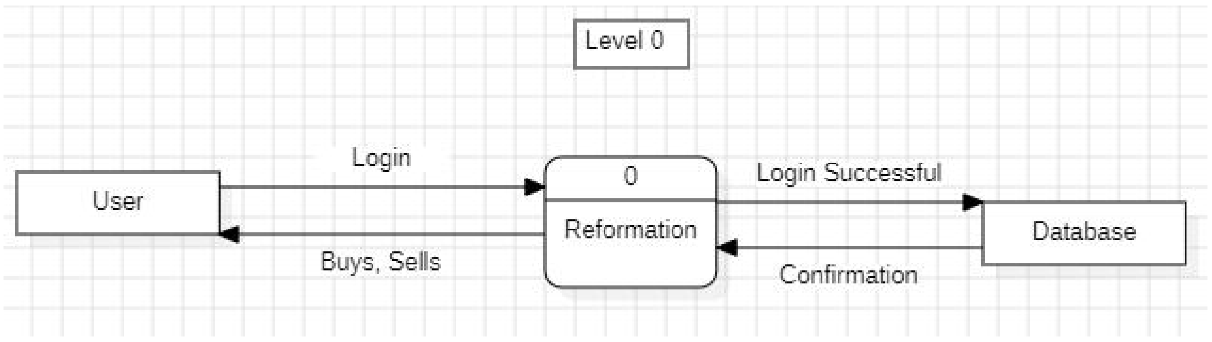
- What data is the system processes?
- What transformations are performed?
- What data are stored?
- What results are produced, etc.

Data Flow Diagrams can be represented in several ways. The DFD belongs to structured-analysis modeling tools. Data Flow diagrams are very popular because they help us visualize the major steps and data involved in software-system processes.

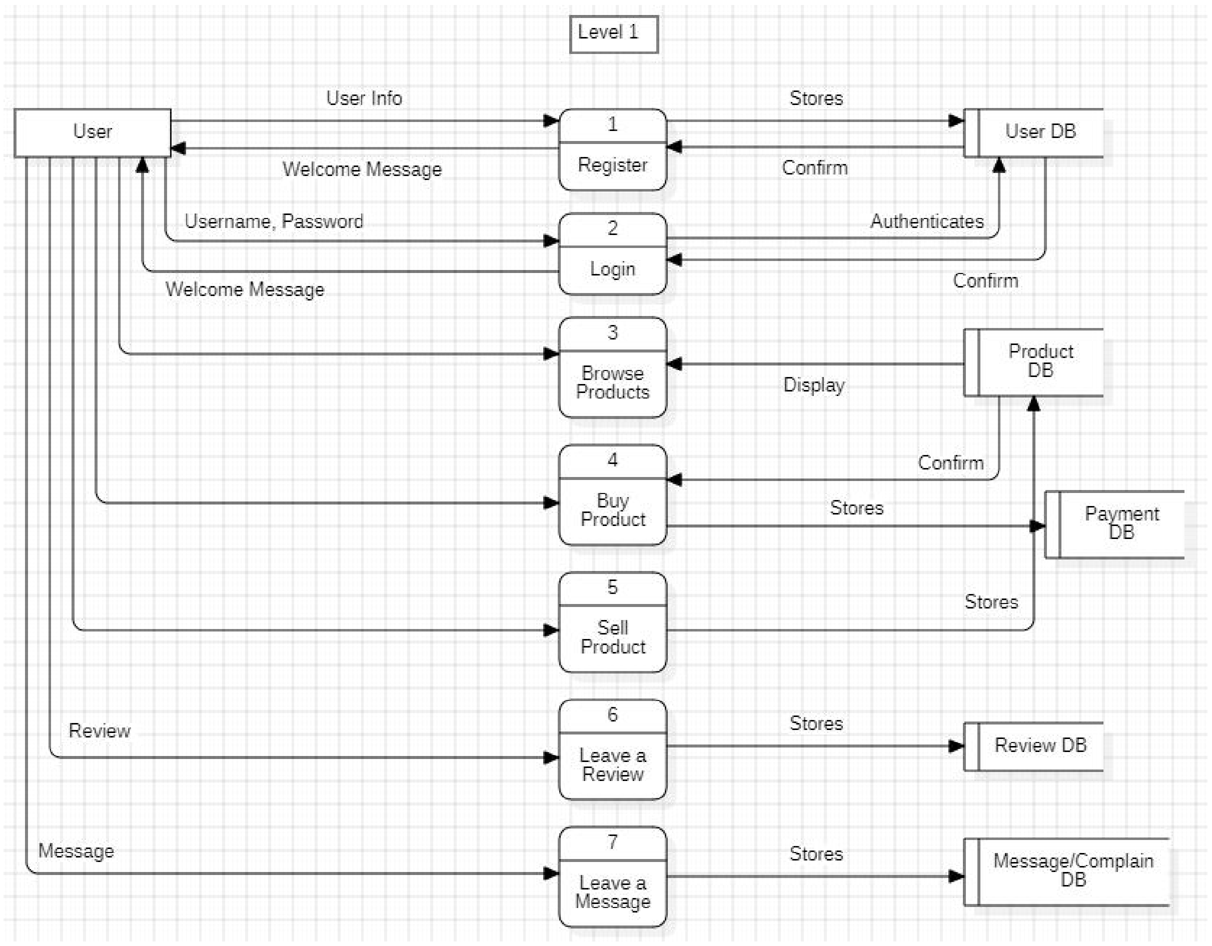
8.4.1 DFD DIAGRAM NOTATIONS.



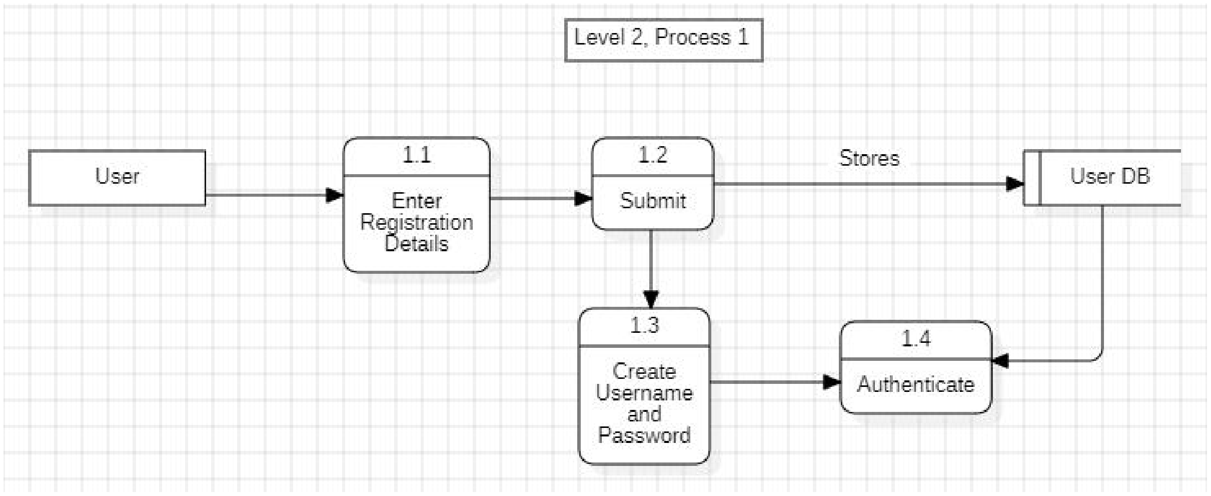
LEVEL 0:



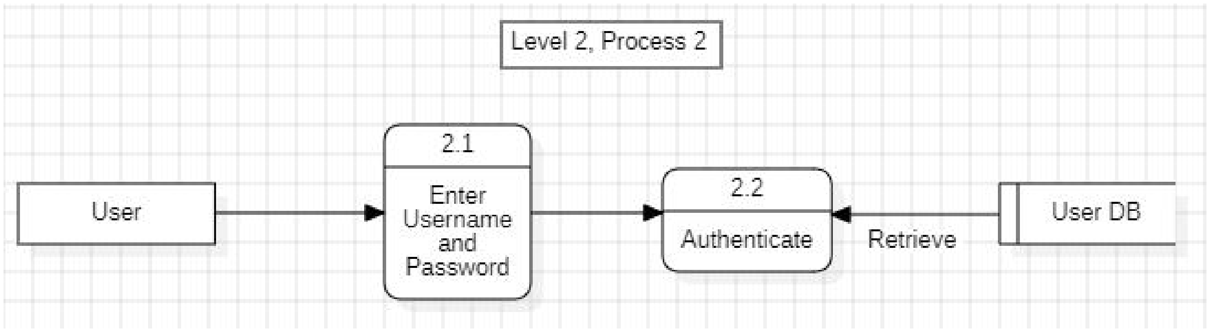
LEVEL 1:



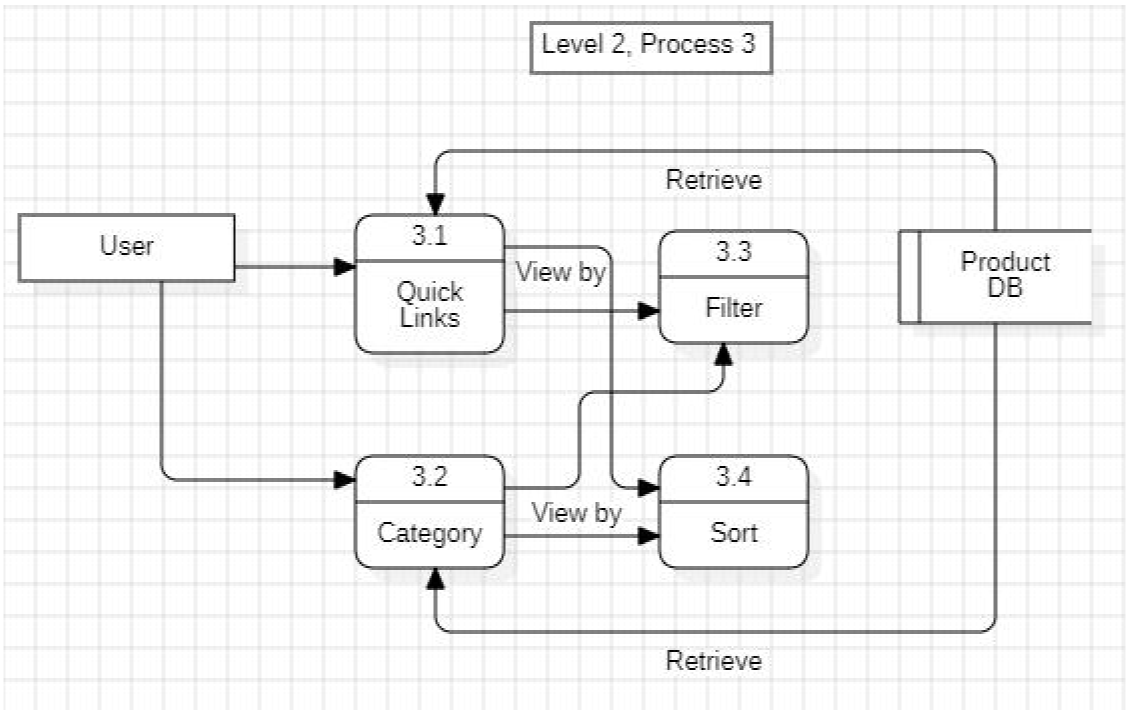
LEVEL 2 OF PROCESS 1: REGISTERATION



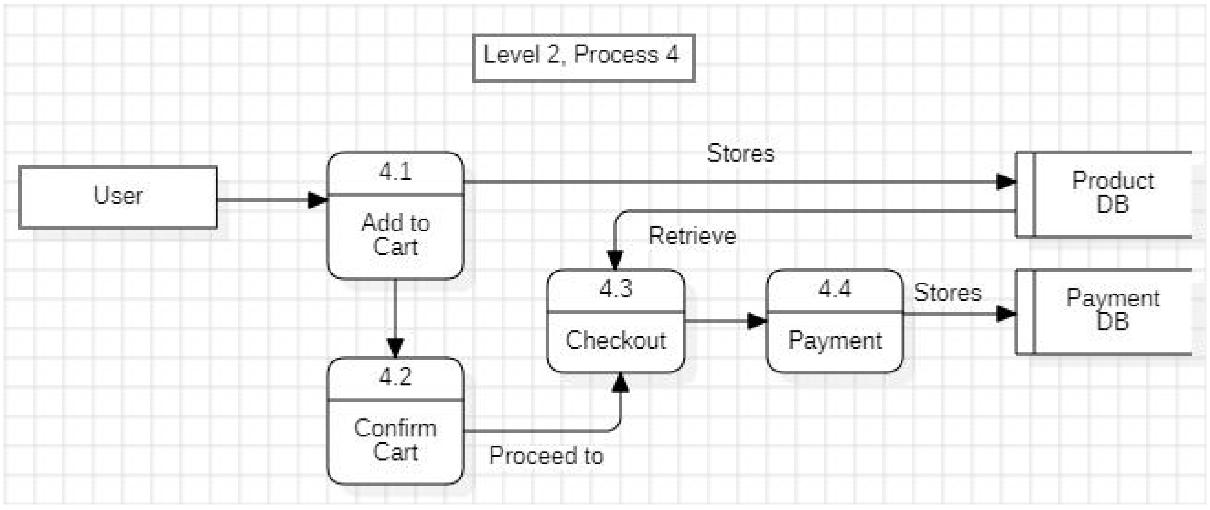
LEVEL 2 OF PROCESS 2: LOGIN



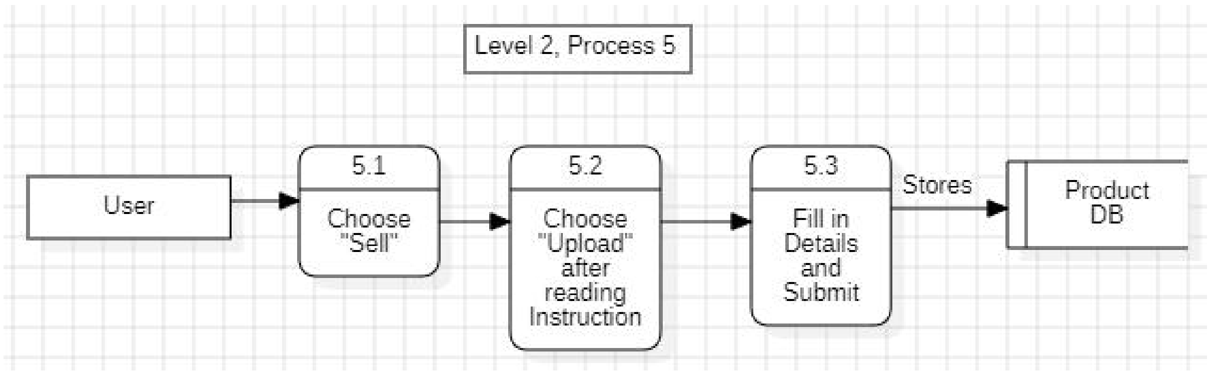
LEVEL 2 OF PROCESS 3: BROWSING PRODUCTS



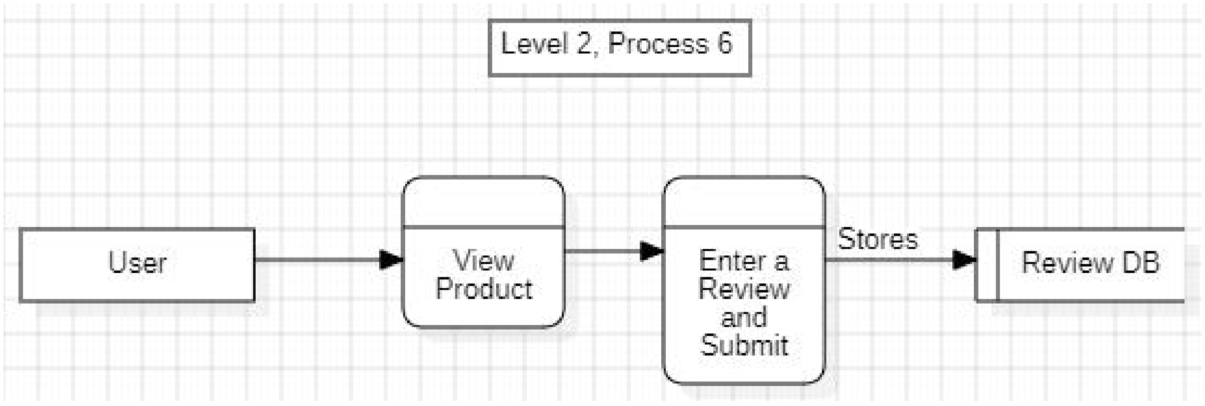
LEVEL 2 OF PROCESS 4: BUYING A PRODUCT



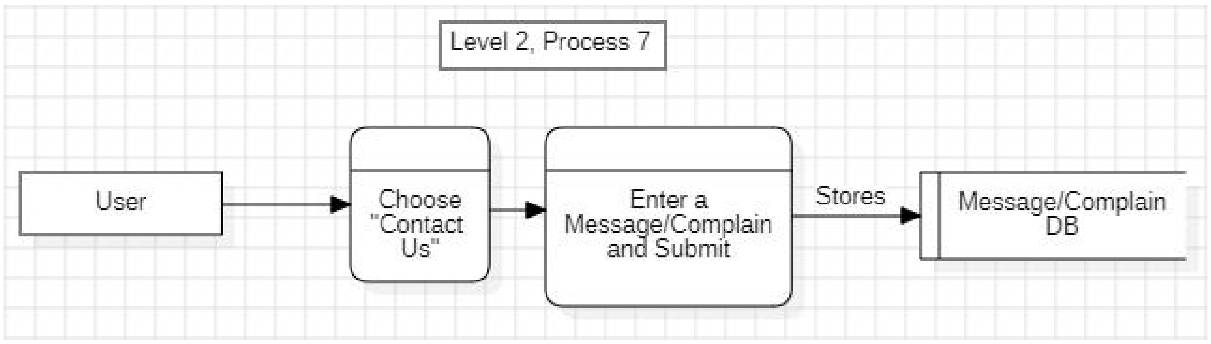
LEVEL 2 OF PROCESS 5: SELLING A PRODUCT


Level 2, Process 5
User
5.1
Choose "Sell"
5.2
Choose "Upload" after reading Instruction
5.3
Fill in Details and Submit
Stores
Product DB

LEVEL 2 OF PROCESS 6: LEAVING A REVIEW



LEVEL 2 OF PROCESS 7: LEAVING A MESSAGE/COMPLAIN



8.5 ACTIVITY DIAGRAM.

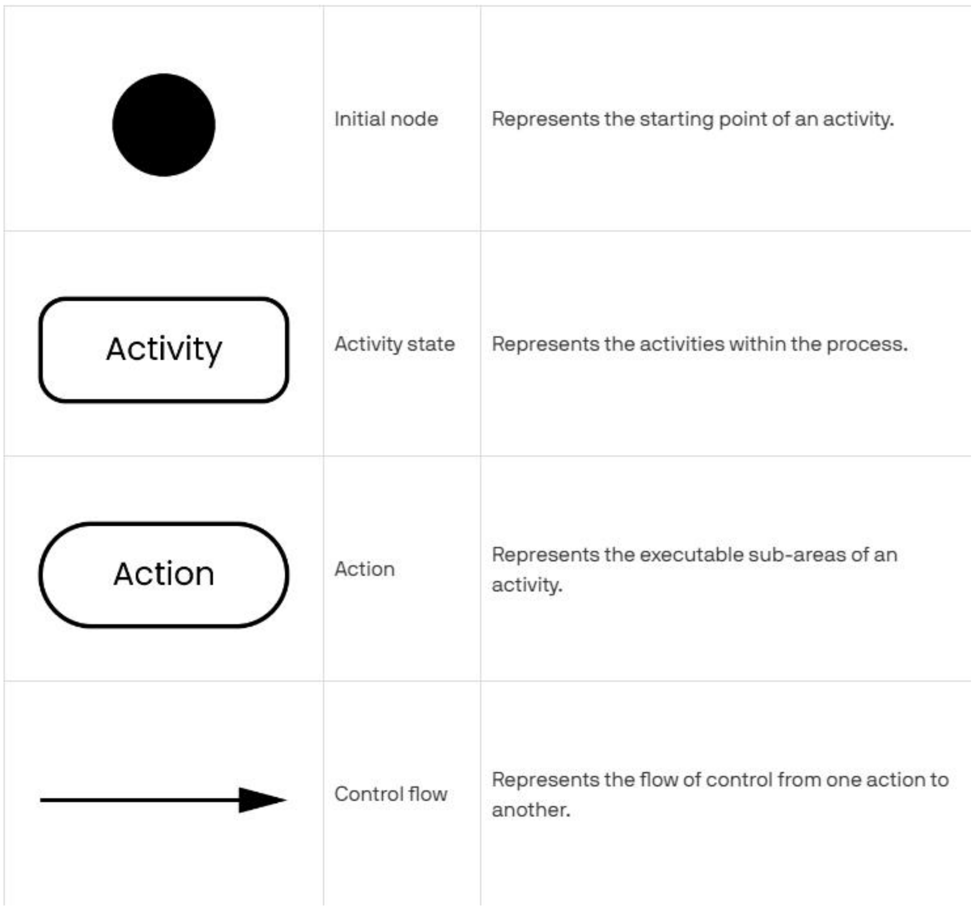
A pastime diagram is a kind of Unified Modelling Language (UML) flowchart that shows the flow from one hobby to every other in a system or method. It's used to explain the one-of-a-kind dynamic aspect of a system and is referred to as a 'behaviour diagram' as it describes what need to manifest within the modelled device.

Even very complicated systems can be visualized through hobby diagrams. As a end result, pastime diagrams are frequently used in enterprise procedure modelling or to describe the steps of a use case diagram within businesses. They show the individual steps in an interest and the order in which they're presented. They also can display the waft of statistics between sports.

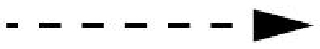
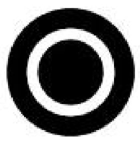
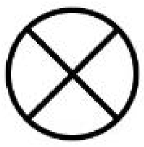
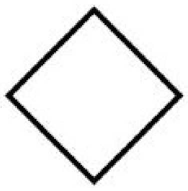
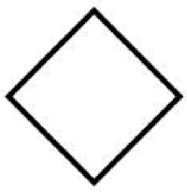
Activity diagrams display the procedure from the start (the preliminary state) to the give up (the very last nation). Each pastime diagram consists of an action, decision node, manipulate flows, begin node, and stop node. Activity diagrams are a wonderful way to expose the waft of sports in a device or technique. They have a huge range of packages within businesses. Activity diagrams are often used to expose purchaser journeys.

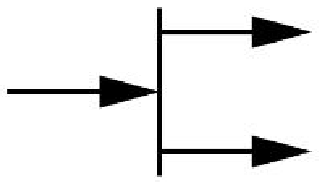
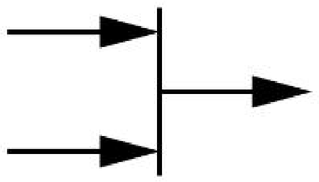
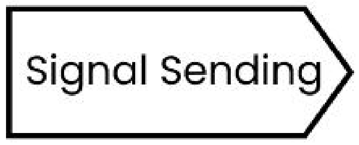
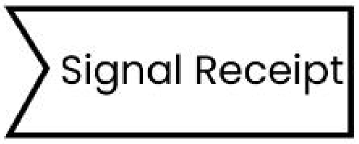
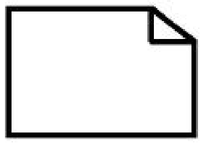
Activity diagrams clearly display the development of workflow between the users. This makes it less difficult for managers to see what is working properly and where the bottlenecks are inside the system to make relevant modifications. Activity diagrams provide a clean visualization of the logic of a set of rules. This affords a clean view of what's happening behind the curtain and makes it easy to identify what's operating nicely and desires improvement.

8.5.1 ACTIVITY DIAGRAM NOTATIONS.



●	Initial node	Represents the starting point of an activity.
Activity	Activity state	Represents the activities within the process.
Action	Action	Represents the executable sub-areas of an activity.
→	Control flow	Represents the flow of control from one action to another.

	Object flow	Represents the path of the objects moving through the activity.
	Activity final node	Represents the end of all control flows within the activity.
	Flow final node	Represents the end of a single control flow.
	Decision node	Represents a conditional branch point with a single input and multiple outputs.
	Merge node	Represents the merging of flows with several inputs and only one output.

	Fork	Represents a flow that can branch into two or more parallel flows.
	Merge	Represents two inputs merging into a single output.
	Signal sending	Represents sending a signal to an accepting activity.
	Signal receipt	Represents that the activity has received the signal.
	Comment	Can be used to add comments to elements.

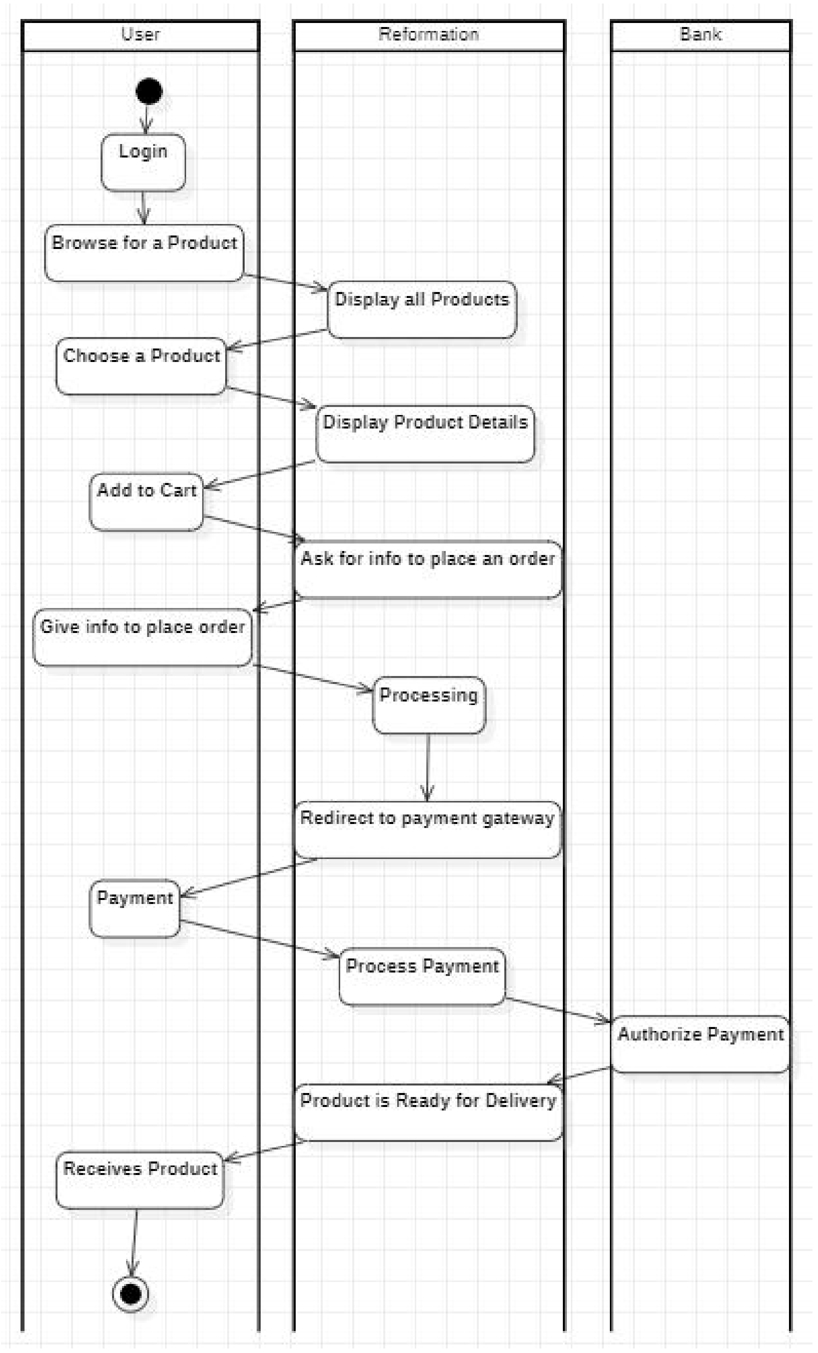


FIGURE 8.5: ACTIVITY DIAGRAM TO BUY OF REFORMATION

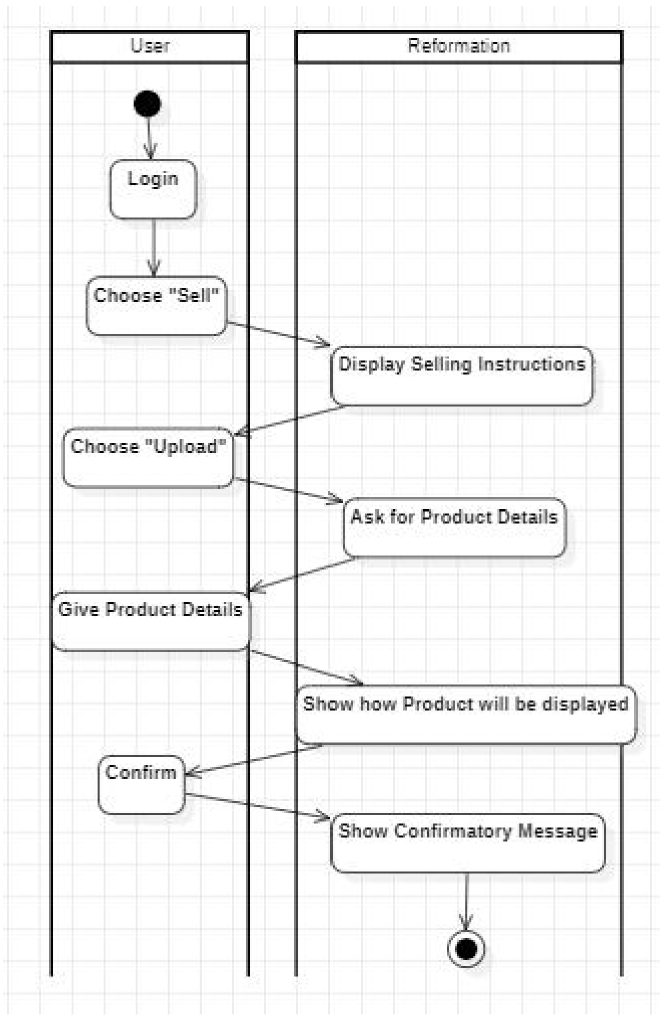


FIGURE 8.5: ACTIVITY DIAGRAM TO SELL OF REFORMATION

8.6 SEQUENCE DIAGRAM.

A grouping outline is one of the numerous kinds of framework communication charts utilized inside Brought together Displaying Language (UML) to address collaborations between the items that live inside a framework outwardly. Specifically, sequence diagrams provide a view of the order in which those interactions occur through depictions of individual objects, called lifelines, and the messages between them. With legitimate documentation, groups can utilize succession charts in numerous phases of the improvement cycle to delineate the planning of cooperations.

Sequence diagrams have a variety of uses, many of them geared toward system design, testing, and maintenance. Programming engineers and draftsmen can involve these graphs in starting to arrange stages to conceptualize configuration, lay out framework prerequisites, and consider how individual parts of the framework add to the general ultimate objective.

8.6.1 SEQUENCE DIAGRAM NOTATIONS.

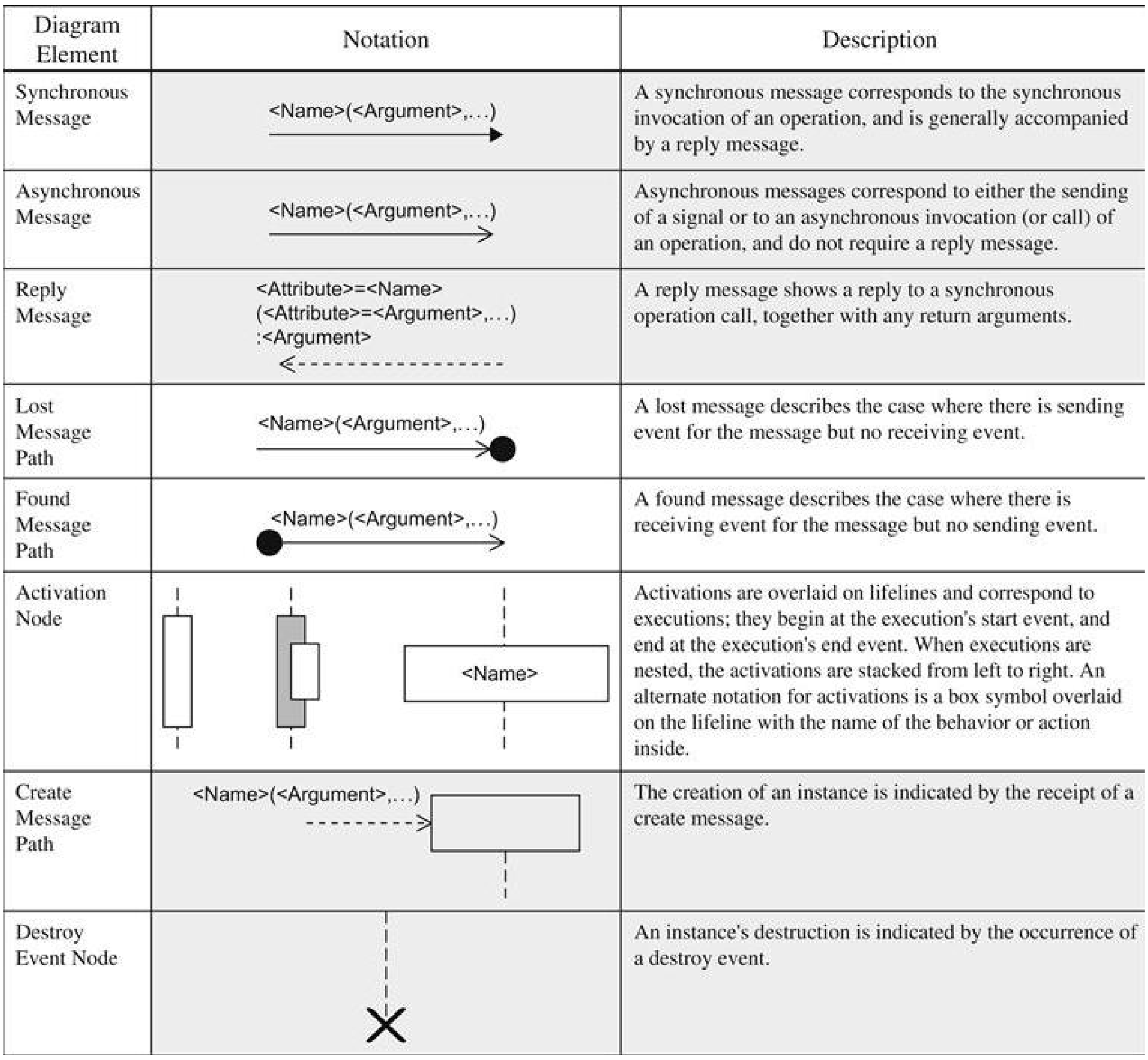
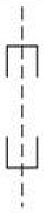


Diagram Element	Notation	Description
Synchronous Message	<Name>(<Argument>,...)	A synchronous message corresponds to the synchronous invocation of an operation, and is generally accompanied by a reply message.
Asynchronous Message	<Name>(<Argument>,...)	Asynchronous messages correspond to either the sending of a signal or to an asynchronous invocation (or call) of an operation, and do not require a reply message.
Reply Message	<Attribute>=<Name> (<Attribute>=<Argument>,...) :<Argument>	A reply message shows a reply to a synchronous operation call, together with any return arguments.
Lost Message Path	<Name>(<Argument>,...)	A lost message describes the case where there is sending event for the message but no receiving event.
Found Message Path	<Name>(<Argument>,...)	A found message describes the case where there is receiving event for the message but no sending event.
Activation Node	<Name>	Activations are overlaid on lifelines and correspond to executions; they begin at the execution's start event, and end at the execution's end event. When executions are nested, the activations are stacked from left to right. An alternate notation for activations is a box symbol overlaid on the lifeline with the name of the behavior or action inside.
Create Message Path	<Name>(<Argument>,...)	The creation of an instance is indicated by the receipt of a create message.
Destroy Event Node		An instance's destruction is indicated by the occurrence of a destroy event.

Coregion Symbol		Within a coregion, there is no implied order between any messages sent or received by the lifeline.

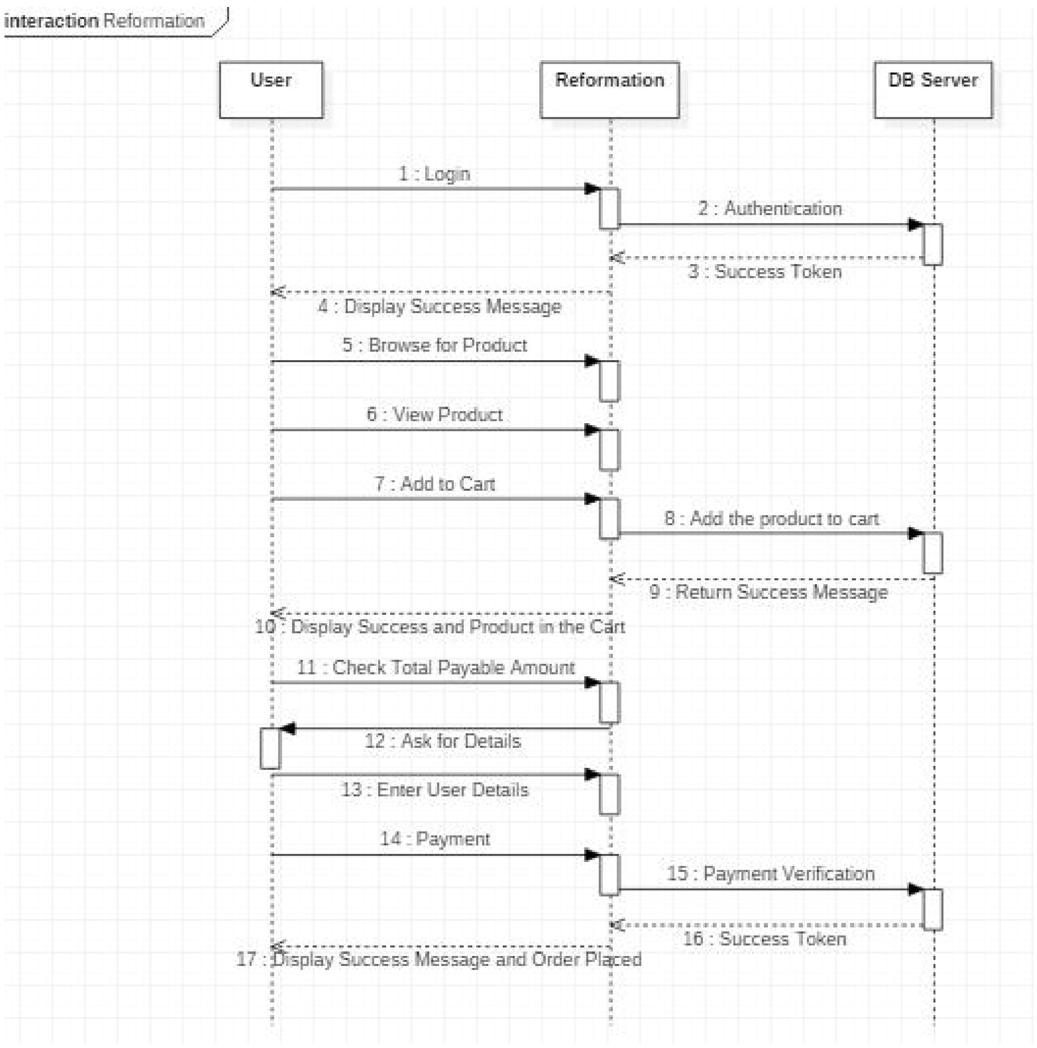
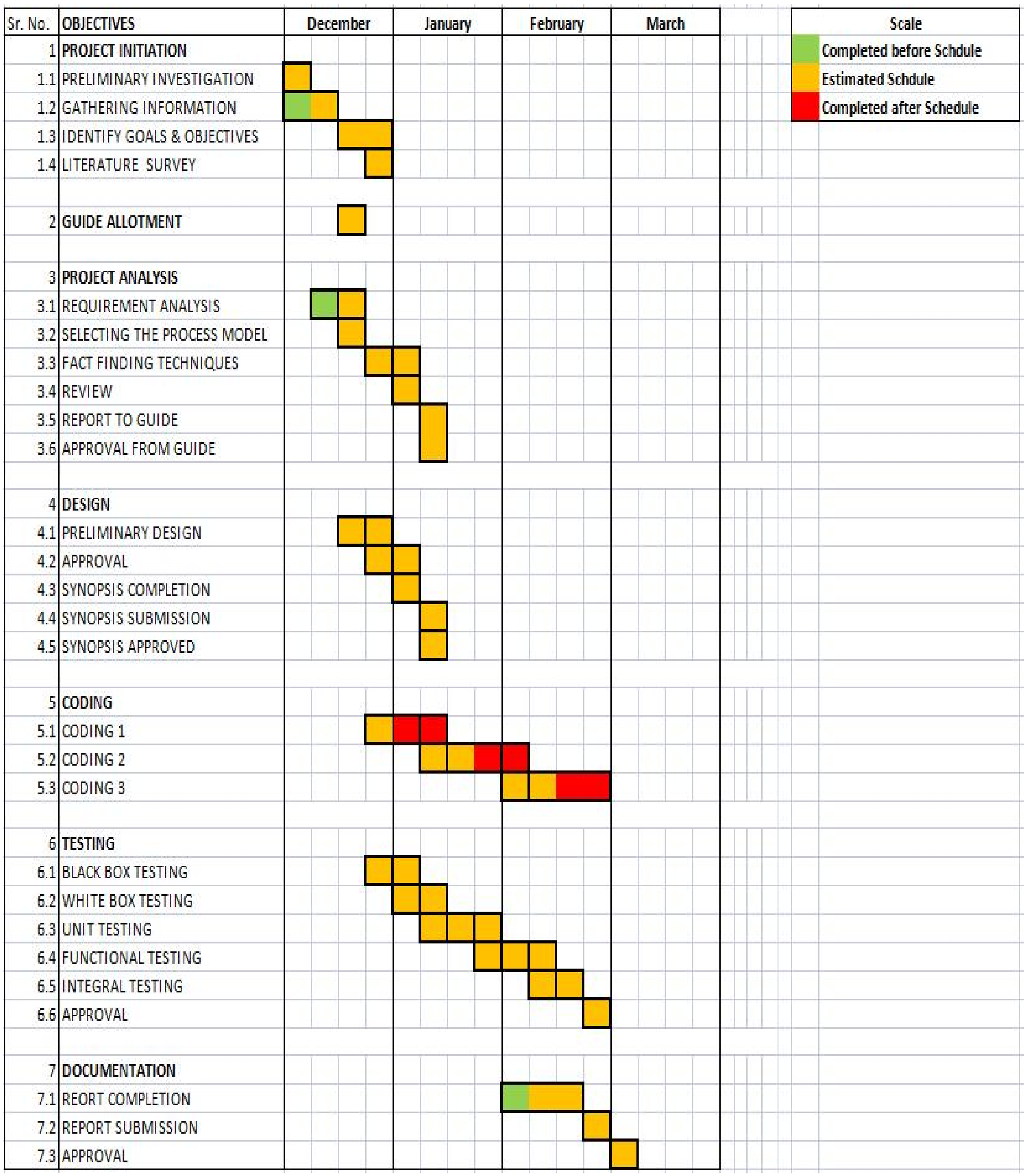

interaction Reformation
User
Reformation
DB Server
1 : Login
2 : Authentication
3 : Success Token
4 : Display Success Message
5 : Browse for Product
6 : View Product
7 : Add to Cart
8 : Add the product to cart
9 : Return Success Message
10 : Display Success and Product in the Cart
11 : Check Total Payable Amount
12 : Ask for Details
13 : Enter User Details
14 : Payment
15 : Payment Verification
16 : Success Token
17 : Display Success Message and Order Placed

FIGURE 8.6: SEQUENCE DIAGRAM OF REFORMATION

9. GANTT CHART



Sr. No.	OBJECTIVES	December				January				February				March			
1	**PROJECT INITIATION**																
1.1	PRELIMINARY INVESTIGATION	🟨															
1.2	GATHERING INFORMATION	🟩	🟨														
1.3	IDENTIFY GOALS & OBJECTIVES			🟨	🟨												
1.4	LITERATURE SURVEY				🟨												
2	**GUIDE ALLOTMENT**			🟨													
3	**PROJECT ANALYSIS**																
3.1	REQUIREMENT ANALYSIS		🟩	🟨													
3.2	SELECTING THE PROCESS MODEL			🟨													
3.3	FACT FINDING TECHNIQUES				🟨	🟨											
3.4	REVIEW					🟨											
3.5	REPORT TO GUIDE						🟨										
3.6	APPROVAL FROM GUIDE						🟨										
4	**DESIGN**																
4.1	PRELIMINARY DESIGN			🟨	🟨												
4.2	APPROVAL				🟨	🟨											
4.3	SYNOPSIS COMPLETION					🟨											
4.4	SYNOPSIS SUBMISSION						🟨										
4.5	SYNOPSIS APPROVED						🟨										
5	**CODING**																
5.1	CODING 1				🟨	🟥	🟥										
5.2	CODING 2						🟨	🟨	🟥	🟥							
5.3	CODING 3									🟨	🟨	🟥	🟥				
6	**TESTING**																
6.1	BLACK BOX TESTING				🟨	🟨											
6.2	WHITE BOX TESTING					🟨	🟨										
6.3	UNIT TESTING						🟨	🟨	🟨								
6.4	FUNCTIONAL TESTING								🟨	🟨	🟨						
6.5	INTEGRAL TESTING										🟨	🟨					
6.6	APPROVAL												🟨				
7	**DOCUMENTATION**																
7.1	REORT COMPLETION									🟩	🟨	🟨					
7.2	REPORT SUBMISSION												🟨				
7.3	APPROVAL													🟨			

Scale	
🟩	Completed before Schdule
🟨	Estimated Schdule
🟥	Completed after Schedule

10. DATA STRUCTURE

USER REGISTRATION DETAILS

REGISTRATION	
ID	Integer(AUTO INCREMENT)
Firstname	String
Lastname	String
Username	String
Password	Varchar
Phone	String
Email	Varchar

USER LOGIN DETAILS

LOGIN	
ID	Integer(AUTO INCREMENT)
Username	String
Password	Varchar

PRODUCT DETAILS

PRODUCT	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
Product_Title	String
Product_Price	Double
Product_Category	String

CART DETAILS

CART	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Quantity	Integer

FAVOURITE DETAILS

FAVOURITE	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Quantity	Integer

SELL DETAILS

SELL	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Product_Title	String
Product_Price	Double
Product_Category	String

CATEGORY DETAILS

CATEGORY	
ID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Quantity	Integer

ORDER DETAILS

ORDER	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Quantity	Integer
TotalAmount	Double
OrderDate	Date

SHIPMENT DETAILS

SHIPMENT	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
OrderID	Integer(AUTO INCREMENT)
Quantity	Integer
TotalAmount	Double
DeliveryDate	Date

PAYMENT DETAILS

PAYMENT	
ID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
TotalAmount	Double
PaymentDate	Date

REVIEW DETAILS

REVIEW	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
ProductID	Integer(AUTO INCREMENT)
Rating	Integer
Content	String
ReviewDate	Date

MESSAGE/COMPLAIN DETAILS

MESSAGE/COMPLAIN	
ID	Integer(AUTO INCREMENT)
UserID	Integer(AUTO INCREMENT)
Content	String
Message/ComplainDate	Date

11. SYSTEM CODING

Registration.jsx

```
import { Grid, TextField, Button, Box, Snackbar, Alert } from "@mui/material";
import { useNavigate } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { getUser, register } from "../../../Redux/Auth/Action";
import { Fragment, useEffect, useState } from "react";
export default function RegisterUserForm({ handleNext }) {
  const navigate = useNavigate();
  const dispatch=useDispatch();
  const [openSnackBar,setOpenSnackBar]=useState(false);
  const { auth } = useSelector((store) => store);
  const handleClose=()=>setOpenSnackBar(false);
  const jwt=localStorage.getItem("jwt");
useEffect(()=>{
  if(jwt){
    dispatch(getUser(jwt))
  }
},[jwt])
  useEffect(() => {
    if (auth.user || auth.error) setOpenSnackBar(true)
  }, [auth.user]);
  const handleSubmit = (event) => {
    event.preventDefault();
    const data = new FormData(event.currentTarget);
    // eslint-disable-next-line no-console
    const userData={
      firstName: data.get("firstName"),
      lastName: data.get("lastName"),
      email: data.get("email"),
     password: data.get("password"),
    }
    console.log("user data",userData);
    dispatch(register(userData))
  };
  return (
    <div className="">
      <form onSubmit={handleSubmit}>
        <Grid container spacing={3}>
          <Grid item xs={12} sm={6}>
            <TextField
              required
              id="firstName"
              name="firstName"
              label="First Name"
              fullWidth
              autoComplete="given-name"
            />
          </Grid>
          <Grid item xs={12} sm={6}>
            <TextField
              required
              id="lastName"
              name="lastName"
              label="Last Name"
              fullWidth
              autoComplete="given-name"
            />
          </Grid>
          <Grid item xs={12}>
            <TextField
              required
              id="email"
              name="email"
              label="Email"
              fullWidth
              autoComplete="given-name"
            />
          </Grid>
          <Grid item xs={12}>
            <TextField
              required
              id="password"
              name="password"
              label="Password"
              fullWidth
              autoComplete="given-name"
              type="password"
            />
          </Grid>
          <Grid item xs={12}>
            <Button
              className="bg-[#9155FD] w-full"
              type="submit"
              variant="contained"
              size="large"
              sx={{padding:".8rem 0"}}
            >
              Register
            </Button>
```

```
          </Grid>

        </Grid>

      </form>

<div className="flex justify-center flex-col items-center">

    <div className="py-3 flex items-center ">

        <p className="m-0 p-0">if you have already account ?</p>

        <Button onClick={()=> navigate("/login")} className="ml-5" size="small">

          Login

        </Button>

      </div>

</div>

<Snackbar open={openSnackBar} autoHideDuration={6000} onClose={handleClose}>

        <Alert onClose={handleClose} severity="success" sx={{ width: '100%' }}>

          {auth.error?auth.error:auth.user?"Register Success":""}

        </Alert>

      </Snackbar>

    </div>

  );

}
```

Login.jsx

```
import * as React from "react";

import { Grid, TextField, Button, Box, Snackbar, Alert } from "@mui/material";

import { useNavigate } from "react-router-dom";

import { useDispatch, useSelector } from "react-redux";

import { getUser, login } from "../../../Redux/Auth/Action";

import { useEffect } from "react";

import { useState } from "react";

export default function LoginUserForm({ handleNext }) {

  const navigate = useNavigate();

  const dispatch=useDispatch();
  const jwt=localStorage.getItem("jwt");
  const [openSnackBar,setOpenSnackBar]=useState(false);
  const { auth } = useSelector((store) => store);
  const handleCloseSnakbar=()=>setOpenSnackBar(false);
  useEffect(()=>{
    if(jwt){
      dispatch(getUser(jwt))
    }
  },[jwt])
    useEffect(() => {
      if (auth.user || auth.error) setOpenSnackBar(true)
    }, [auth.user]);
  const handleSubmit = (event) => {
    event.preventDefault();
    const data = new FormData(event.currentTarget);
    const userData={
      email: data.get("email"),
      password: data.get("password"),
    }
    console.log("login user",userData);
    dispatch(login(userData));
  };
  return (
    <React.Fragment className=" shadow-lg ">
      <form className="w-full" onSubmit={handleSubmit}>
        <Grid container spacing={3}>
          <Grid item xs={12}>
            <TextField
              required
              id="email"
              name="email"
              label="Email"
              fullWidth
              autoComplete="given-name"
            />
          </Grid>
          <Grid item xs={12}>
            <TextField
              required
              id="password"
              name="password"
              label="Password"
              fullWidth
              autoComplete="given-name"
              type="password"
            />
          </Grid>
          <Grid item xs={12}>
            <Button
              className="bg-[#9155FD] w-full"
              type="submit"
              variant="contained"
              size="large"
              sx={{padding:".8rem 0"}}
            >
              Login
            </Button>
          </Grid>
        </Grid>
      </form>
```

```
    <div className="flex justify-center flex-col items-center">
      <div className="py-3 flex items-center">
      <p className="m-0 p-0">don't have account ?</p>
      <Button onClick={()=> navigate("/register")} className="ml-5" size="small">
        Register
      </Button>
      </div>
    </div>
    <Snackbar open={openSnackBar} autoHideDuration={6000}
onClose={handleCloseSnakbar}>
      <Alert onClose={handleCloseSnakbar} severity="success" sx={{ width: '100%' }}>
        {auth.error?auth.error:auth.user?"Register Success":""}
      </Alert>
    </Snackbar>
   </React.Fragment>
  );
}
```

Sell.jsx

```
import React, { useState } from 'react';
import AddIcon from '@mui/icons-material/Add';
const SellPage = () => {
  const [products, setProducts] = useState([]);
  const [productTitle, setProductTitle] = useState('');
  const [productDescription, setProductDescription] = useState('');
  const [productBrand, setProductBrand] = useState('');
  const [productColor, setProductColor] = useState('');
  const [productSize, setProductSize] = useState(' ');
  const [productPrice, setProductPrice] = useState('');
  const [topTierCategory, setTopTierCategory] = useState('');
  const [secondTierCategory, setSecondTierCategory] = useState('');
```

```
 const [thirdTierCategory, setThirdTierCategory] = useState('');

 const [productImage, setProductImage] = useState(null);

 const handleProductTitleChange = (e) => {

   setProductTitle(e.target.value);

 };

 const handleProductDescriptionChange = (e) => {

   setProductDescription(e.target.value);

 };

 const handleProductBrandChange = (e) => {

   setProductBrand(e.target.value);

 };

 const handleProductColorChange = (e) => {

   setProductColor(e.target.value);

 };

 const handleProductSizeChange = (e) => {

   setProductSize(e.target.value);

 };

 const handleProductPriceChange = (e) => {

   // Allow only positive numerical values

   const inputValue = e.target.value;

 if (/^\d*\.?\d*$/.test(inputValue)) {

   const numericValue = parseFloat(inputValue);


   // Additional validation for the range (0 to 650)

   if (!isNaN(numericValue) && numericValue > 0 && numericValue <= 650) {

     setProductPrice(inputValue);

   } else {

     alert('Please enter a valid price between 0 and 650.');

   }

 }
 };
 const handleTopTierCategoryChange = (e) => {
  setTopTierCategory(e.target.value);
 };
 const handleSecondTierCategoryChange = (e) => {
  setSecondTierCategory(e.target.value);
 };
 const handleThirdTierCategoryChange = (e) => {
  setThirdTierCategory(e.target.value);
 };
 const handleProductImageChange = (e) => {
  if (e.target.files && e.target.files[0]) {
   const reader = new FileReader();
   reader.onload = (event) => {
    setProductImage(event.target.result);
   };
   reader.readAsDataURL(e.target.files[0]);
  }
 };
 const handleAddProduct = () => {
  if (
   productTitle &&
   productDescription &&
   productBrand &&
   productColor &&
   productPrice &&
   topTierCategory &&
   secondTierCategory &&
   thirdTierCategory &&
   productImage
```

```
) {
  const newProduct = {
    id: products.length + 1,
    title: productTitle,
    description: productDescription,
    brand: productBrand,
    color: productColor,
    size: productSize,
    price: parseFloat(productPrice),
    topTierCategory,
    secondTierCategory,
    thirdTierCategory,
    image: productImage,
  };

  setProducts([...products, newProduct]);
  setProductTitle('');
  setProductDescription('');
  setProductBrand('');
  setProductColor('');
  setProductSize('');
  setProductPrice('');
  setTopTierCategory('');
  setSecondTierCategory('');
  setThirdTierCategory('');
  setProductImage(null);
  alert('Your Item has been uploaded for sale');
} else {
  alert('Please fill in all fields');
}
```

```
  };

  return (

    <div style={{ fontFamily: 'Arial, sans-serif', maxWidth: '800px', margin: 'auto', padding:
'20px' }}>

      <h1 style={{ textAlign: 'left', color: '#333', padding: '15px', fontSize: '2em', fontWeight:
'bold' }}>Sell Your Product:</h1>

      <div style={{ display: 'grid', gridTemplateColumns: 'auto 1fr', gap: '20px', marginBottom:
'15px' }}>

        <div style={{ width: '200px', height: '200px', position: 'relative', marginRight: '20px',
border: '2px solid #ccc' }}>

          <label htmlFor="productImage" style={{ cursor: 'pointer', position: 'absolute', top:
'50%', left: '50%', transform: 'translate(-50%, -50%)' }}>

            {productImage ? (

              <img src={productImage} alt="Product" style={{ width: '100%', height: '100%',
objectFit: 'cover', borderRadius: '4px' }} />

            ) : (

              <div style={{ fontSize: '50px', color: '#333', textAlign: 'center' }}>

                <AddIcon />

              </div>

            )}

          </label>

          <input

            type="file"

            id="productImage"

            accept="image/*"

            onChange={handleProductImageChange}

            style={{ display: 'none' }}

          />

        </div>

        <div>

          <div style={{ padding: '10px' }}>

            <label htmlFor="productTitle">Title:</label>
            <input
              type="text"
              id="productTitle"
              value={productTitle}
              onChange={handleProductTitleChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="productDescription">Description:</label>
            <textarea
              id="productDescription"
              value={productDescription}
              onChange={handleProductDescriptionChange}
              style={{ width: '100%', padding: '8px', minHeight: '80px', borderRadius: '4px',
      border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="productBrand">Brand:</label>
            <input
              type="text"
              id="productBrand"
              value={productBrand}
              onChange={handleProductBrandChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="productColor">Colour:</label>
            <input
              type="text"
              id="productColor"
              value={productColor}
              onChange={handleProductColorChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="productColor">Size:</label>
            <input
              type="text"
              id="productSize"
              value={productSize}
              onChange={handleProductSizeChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="productPrice">Price (₹):</label>
            <input
              type="text" // Changed from "number" to "text"
              id="productPrice"
              value={productPrice}
              onChange={handleProductPriceChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            />
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="topTierCategory">Top Tier Category:</label>
            <select
              id="topTierCategory"
              value={topTierCategory}
              onChange={handleTopTierCategoryChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            >
              <option value="">Select Top Tier Category</option>
              <option value="Men">Men</option>
              <option value="Women">Women</option>
            </select>
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="secondTierCategory">Second Tier Category:</label>
            <select
              id="secondTierCategory"
              value={secondTierCategory}
              onChange={handleSecondTierCategoryChange}
              style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
            >
              <option value="">Select Second Tier Category</option>
              <option value="Clothing">Clothing</option>
              <option value="accessories">accessories</option>
            </select>
          </div>
          <div style={{ padding: '10px' }}>
            <label htmlFor="thirdTierCategory">Third Tier Category:</label>
            <select
              id="thirdTierCategory"
              value={thirdTierCategory}
              onChange={handleThirdTierCategoryChange}
```

```
          style={{ width: '100%', padding: '8px', borderRadius: '4px', border: '1px solid #ccc' }}
         >
          <option value="">Select Third Tier Category</option>
          <option value="bags_men">men bags</option>
          <option value="handbags_women">women handbags</option>
          <option value="womens_tops">women tops</option>
          <option value="womens_jeans">womens jeans</option>
          <option value="mens_jeans">mens jeans</option>
          <option value="mens_shirt">mens tops</option>
          <option value="sunglasses_men">sunglasses men</option>
          <option value="womens_kurta">womens kurta</option>
         </select>
        </div>
       </div>
      </div>
      <button
       onClick={handleAddProduct}
       style={{ backgroundColor: '#2C3335', color: 'white', padding: '8px 15px', border: 'none',
cursor: 'pointer', float: 'right' }}
      >
       Add Product
      </button>
      <ul style={{ listStyle: 'none', padding: '0' }}>
       {products.map((product) => (
        <li key={product.id} style={{ borderBottom: '1px solid #ddd', padding: '70px ', display:
'flex', alignItems: 'center' }}>
         <img src={product.image} alt={product.title} style={{ maxWidth: '300px',
marginRight: '20px' }} /> {/* Increased image size */}
         <div>
          <li style={{ alignItems: 'center' }}><h1 style={{ fontSize: '1.2em', fontWeight:
'bold' }}>Your Product will be displayed as:</h1></li>
```

```
            <div style={{ fontSize: '1em', fontWeight: 'bold' }}>{product.title}</div>

            <div>{product.description}</div>

            <div>Brand: {product.brand}</div>

            <div>Colour: {product.color}</div>

            <div>Size: {product.size} </div>

            <div>Price: ₹{product.price}</div>

            <div>Top Tier Category: {product.topTierCategory}</div>

            <div>Second Tier Category: {product.secondTierCategory}</div>

            <div>Third Tier Category: {product.thirdTierCategory}</div>

          </div>

        </li>

      ))}

    </ul>

  </div>

 );

};

export default SellPage;
```

Homepage.jsx

```
import React from "react";

import MainCarousel from "../../components/HomeCarousel/MainCarousel";

import HomeSectionCarousel from
"../../components/HomeSectionCarousel/HomeSectionCarousel";

import { womens_tops } from "../../../Data/womens_tops";

import { mens_tops } from "../../../Data/Men/mens_tops";

import { womens_jeans } from "../../../Data/Women/womens_jeans";

import { handbags_women } from "../../../Data/Handbags/handbags_women";

import { mens_jeans } from "../../../Data/Men/mens_jeans";

import { Button } from "@mui/material";

import Stack from '@mui/material/Stack';

import { Sale } from "../../../Data/Sale";
import { useNavigate } from "react-router-dom";

const Homepage = () => {

  const navigate=useNavigate();

  return (

    <div>

      <MainCarousel />

      <section>

        <div className="px-10 font-serif font-bold">QUICK LINKS</div>

        <div className="px-10 py-5">

          <Stack spacing={2} direction="row">

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" color="error" sx={{ px: "3.5rem", py: "0.7rem", bgcolor: "white"}} >New
Arrivals</Button>

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" sx={{ px: "4.5rem", py: "0.7rem", bgcolor: "white", color:"black
"}} >Mens Jeans</Button>

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" sx={{ px: "4.5rem", py: "0.7rem", bgcolor: "white", color:"black
"}} >Brands</Button>

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" sx={{ px: "4.5rem", py: "0.7rem", bgcolor: "white", color:"black
"}} >HandBags</Button>

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" sx={{ px: "4.5rem", py: "0.7rem", bgcolor: "white", color:"black "}} >T-
Shirts</Button>

          </Stack>

          <div className="py-5">

          <Stack spacing={2} direction="row">

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" color="error" sx={{ px: "4.5rem", py: "0.7rem",
bgcolor:"white"}} >Sale</Button>

          <Button onClick={()=>navigate("/:levelOne/:levelTwo/:levelThree")}
variant="outlined" sx={{ px: "4.5rem", py: "0.7rem", bgcolor: "white",
color:"black"}} >Bags</Button>

          </Stack>
```

```
     </div>

    </div>

   </section>

   <section className="">

    <div className=" bg-black">

    <img
src="https://res.cloudinary.com/dtqtt8bcv/image/upload/v1704617403/HP_Desktop_Banner_
f82g9f.webp" alt=""></img>

    </div>

   </section>

   <section>

   <div className="space-y-10 py-10 flex flex-col justify-center px-5 lg:px-2 bg-purple-50
font-serif">

    <HomeSectionCarousel data={Sale} sectionName={"Sale"}></HomeSectionCarousel>

   </div>

   </section>

   <div className="space-y-10 py-10 flex flex-col justify-center px-5 lg:px-2 bg-teal-50
font-serif">

    <HomeSectionCarousel data={womens_tops} sectionName={"Women's Tops"} />

    <HomeSectionCarousel

     data={womens_jeans}

     sectionName={"Women's Jeans"}

    />

    <section className="bg-teal-100 space-y-10 py-20">

    <div>

    <img
src="https://res.cloudinary.com/dtqtt8bcv/image/upload/v1704618736/bb_bginqx.jpg"
alt=""></img>

    </div>

   </section>

    <HomeSectionCarousel data={mens_tops} sectionName={"Men's T-shirts"} />

    <HomeSectionCarousel data={mens_jeans} sectionName={"Men's Jeans"} />
```

```
        <HomeSectionCarousel

          data={handbags_women}

          sectionName={"Women's Handbags"}

        />

      </div>

      <section>

      <div>

        <img
src="https://res.cloudinary.com/dtqtt8bcv/image/upload/v1704619807/bb1_zjhiwa.jpg"
alt=""></img>

        </div>

        </section>

    </div>

  );

};

export default Homepage;
```

12. TESTING PHASES

TESTING PHASE

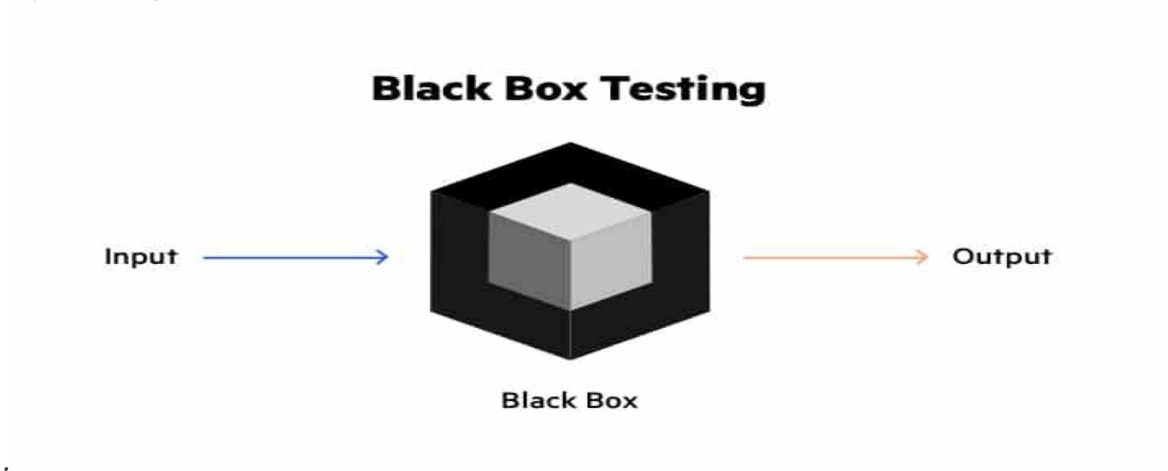
- During the testing phase of software development, several key activities take place.
- Test planning involves defining objectives, scope, and approach for testing.
- These involve detailed descriptions created on the development of test cases for all tests associated with the functionalities, performance, security, and usability that the system will carry out.
- This means the tests create a proof of the execution of test cases under a variety of environmental conditions run by the software, and only then is the behavior checked.
- The defect tracking system combines the defect logging with severity and priority settings, and with very specific information on the defect.
- The test report is nothing but the designed report that summarizes the complete test activities and results of the particular testing, which may include the test coverage metrics and defect metrics.
- Iterative testing means several missed defects during correction and several such tests after the defects are fixed until the software is considered to meet quality release criteria.

METHODS USED FOR SOFTWARE TESTING:

- Black Box Testing
- White Box Testing

12.1 BLACK BOX TESTING:

Black-box software testing is a kind of testing that is conducted by a tester who doesn't know the internal coding. Testing is only concerned with the behaviour and functionality of the software. In other words, the tester relates to software as a "black box" and carries out testing regarding inputs and outputs referring to it, hardly paying attention to the implementation details within it



Black box testing is a testing technique where the test cases are developed concerning software requirements, specifications, and functionality, of which a tester can verify the behavior and conformance to the specified requirements. The techniques that lie in the black box testing domain include equivalence partitioning, boundary value analysis, decision table testing, state transition testing, and use case testing.

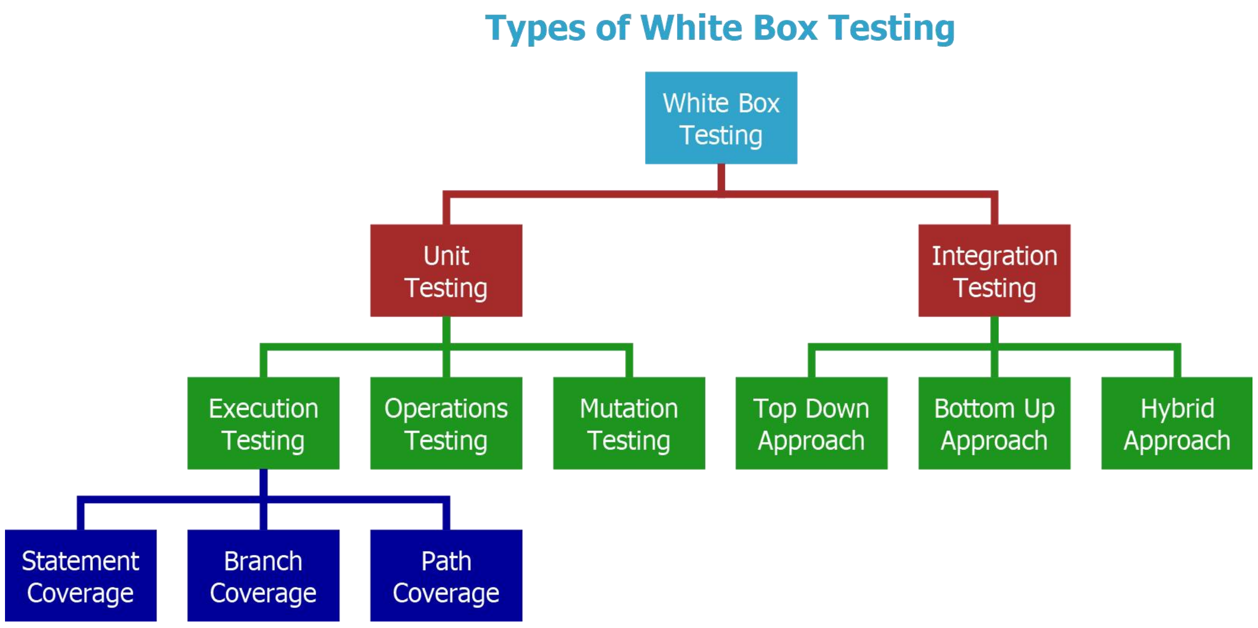
Mainly, black-box testing has an advantage that lets testers learn about bugs and problems through what is known as an "end-user approach" - no insight is given into the working of the code underneath. That is, within its operational environment, the software should come with the right functionalities and be error-free.

12.2 WHITE BOX TESTING

White box testing is the method of ensuring quality control as a software testing approach where the tester knows and controls the design and internal structure with implementation details of the system under test. In white-box testing, testing of internal logic, paths, and data flows of software is undertaken, in contrast to the external behaviour checked within black-box testing.

White box tests apply their internal software working knowledge to developing test cases that focus on the exercising of specific code paths, branches, and conditions. This may include the checking of thoughts around the functions, methods, modules, or components against their expected performance. Some of the techniques pragmatically performed to apply coding testing white box comprise statement coverage, branch coverage, path coverage, and condition coverage, among others.

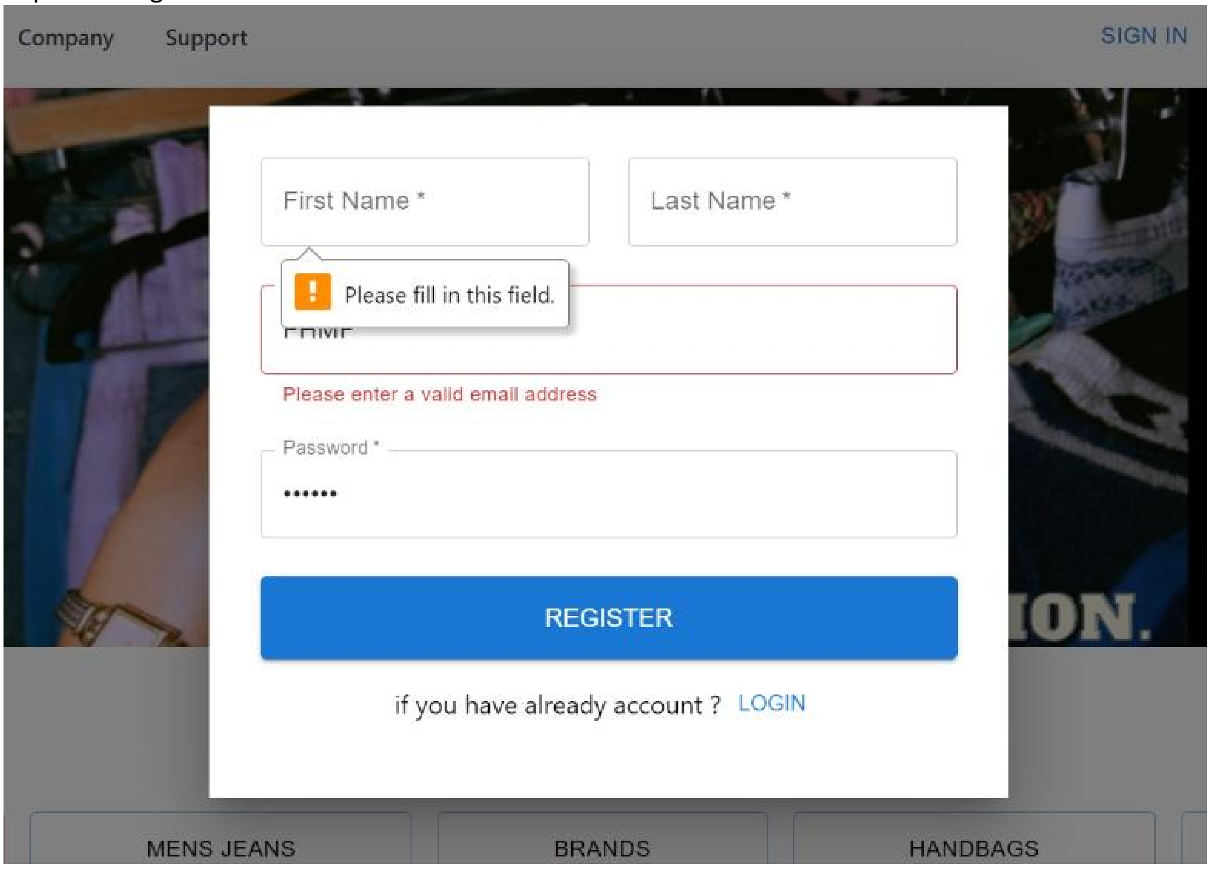
The white-box-testing approach is applied to test that all lines of code execute, the branches of control flow are traversed, and all logical conditions have been exercised. White-box testing ventures into the deep architecture of software, which helps in the identification of those defects, errors, and vulnerabilities that merely cannot be instrumental in plain black-box testing. It is a very useful test tool when combined with other testing techniques to ensure that there is full test coverage at a high level by verifying both the reliability and quality of the software.



BOUNDARY VALUE ANALYSIS

The following is the BVA for all fields in registration

Test ID	Input	Expected Result	Actual Result
1	No first name	"Please fill in this detail"	"Please fill in this detail"
2	No last name	"Please fill in this detail"	"Please fill in this detail"
4	Invalid email and weak password	"Please enter a valid email address" and "Password must be at least 6 characters long"	"Please enter a valid email address" and "Password must be at least 6 characters long"
5	Valid inputs	Redirected to Home page	Redirected to Home page



Company Support SIGN IN

First Name *

Last Name *

Please fill in this field.

FHMF

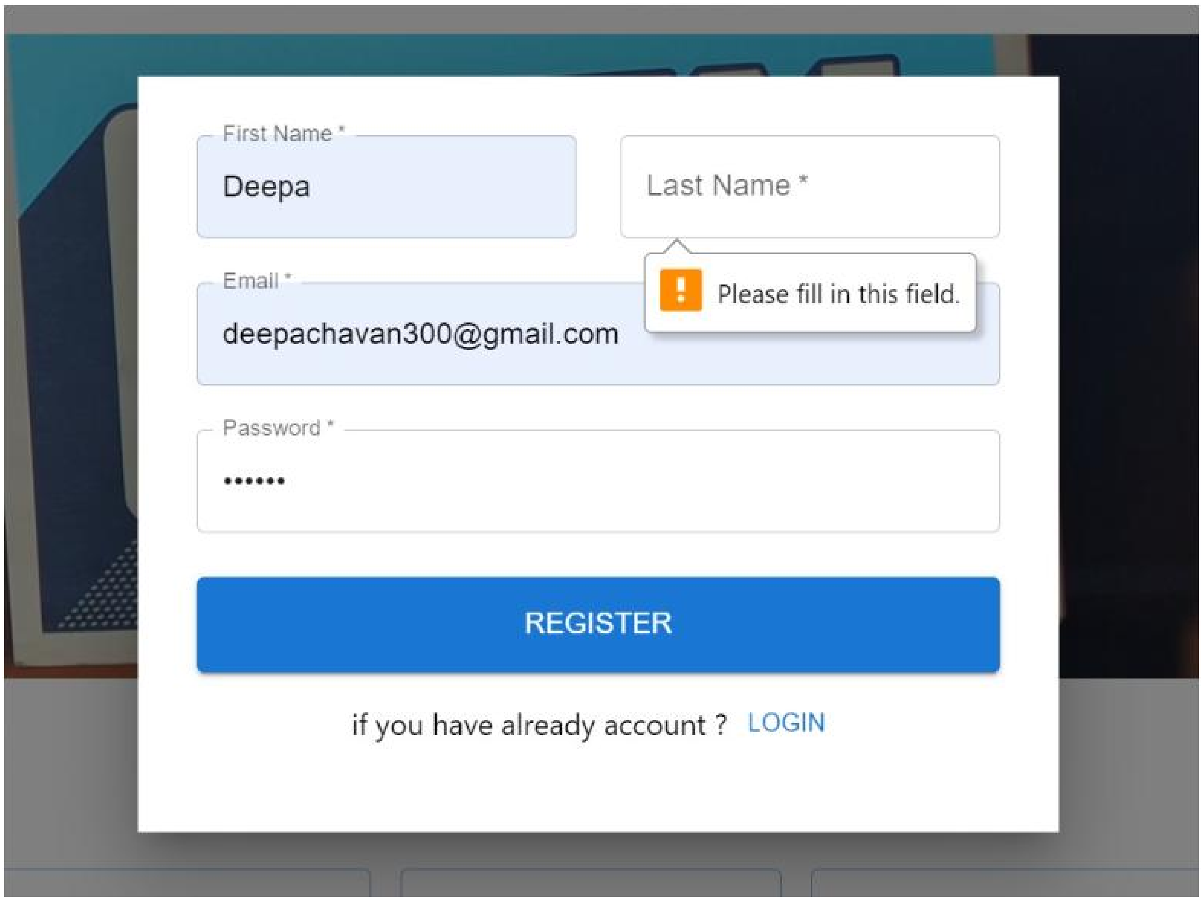
Please enter a valid email address

Password *

••••••

REGISTER

if you have already account ? LOGIN



First Name * Deepa

Last Name *

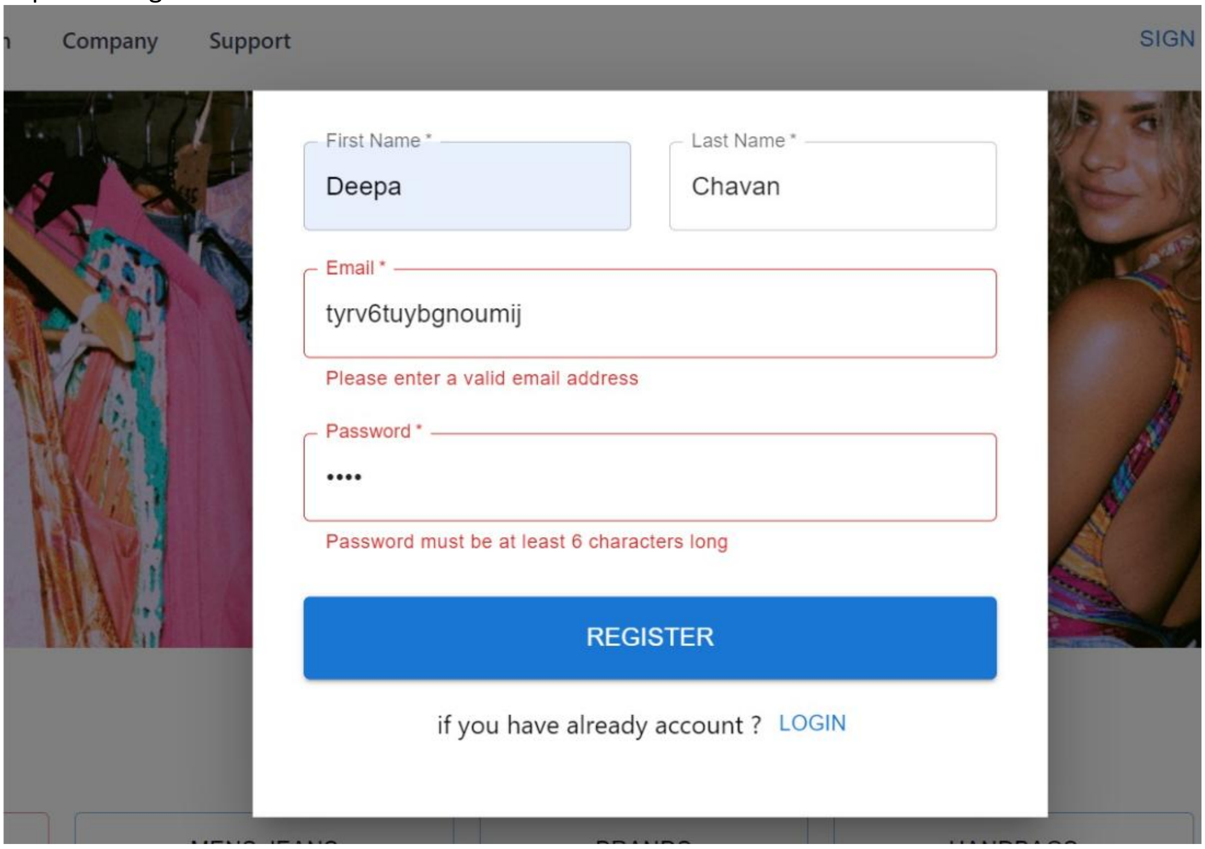
Please fill in this field.

Email * deepachavan300@gmail.com

Password * ••••••

REGISTER

if you have already account ? LOGIN



First Name * Deepa

Last Name * Chavan

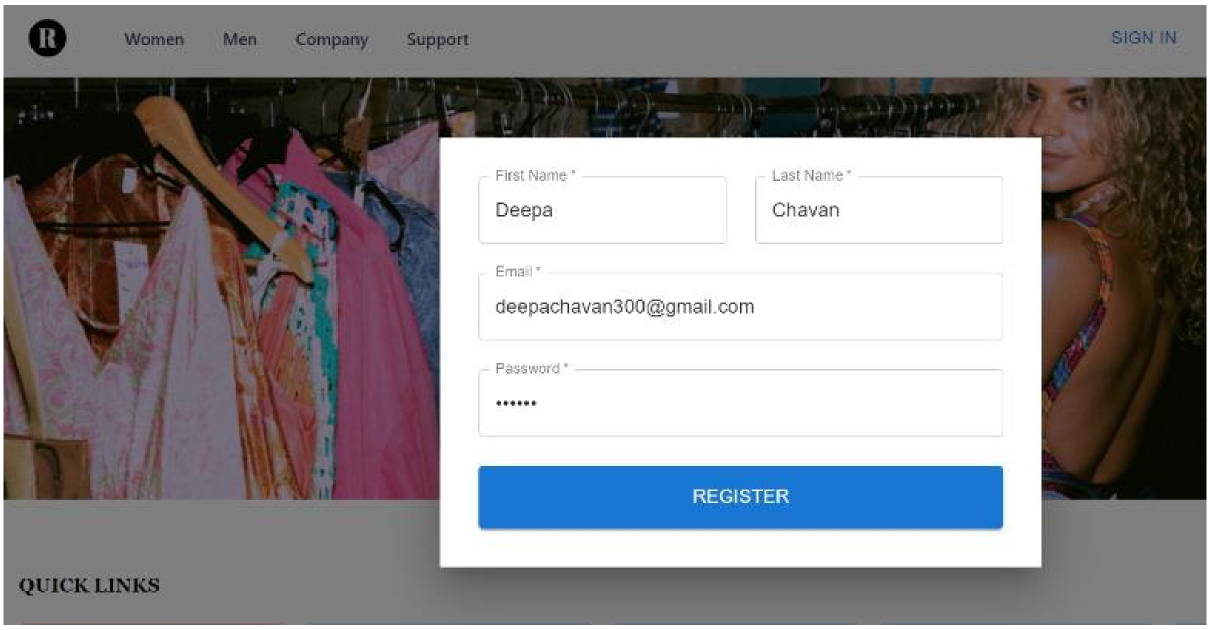
Email * tyrv6tuybgnoumij

Please enter a valid email address

Password * ••••

Password must be at least 6 characters long

if you have already account ? LOGIN

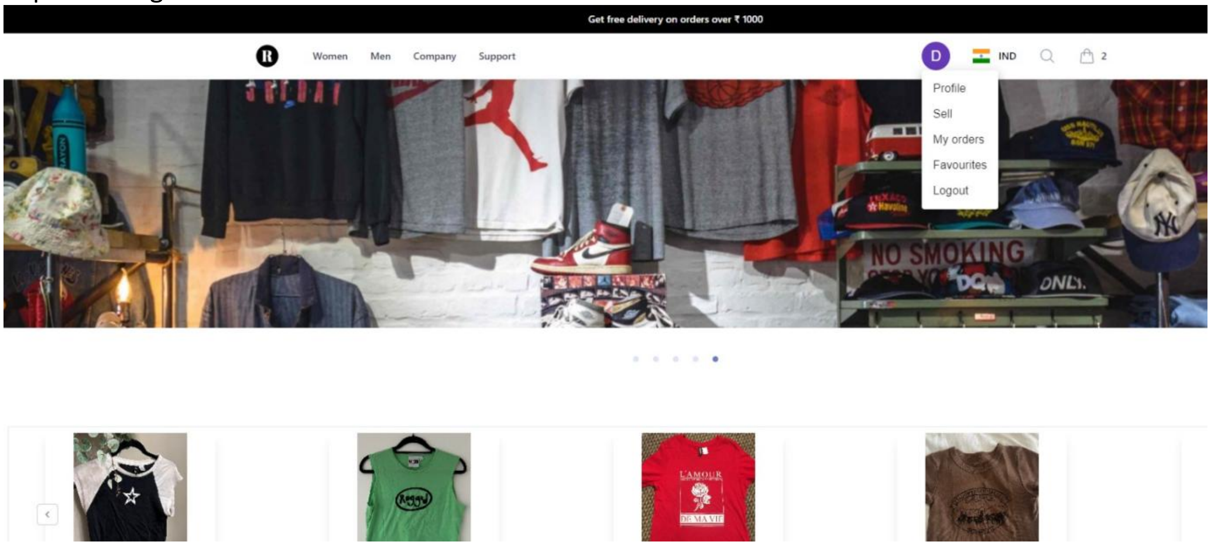


First Name * Deepa

Last Name * Chavan

Email * deepachavan300@gmail.com

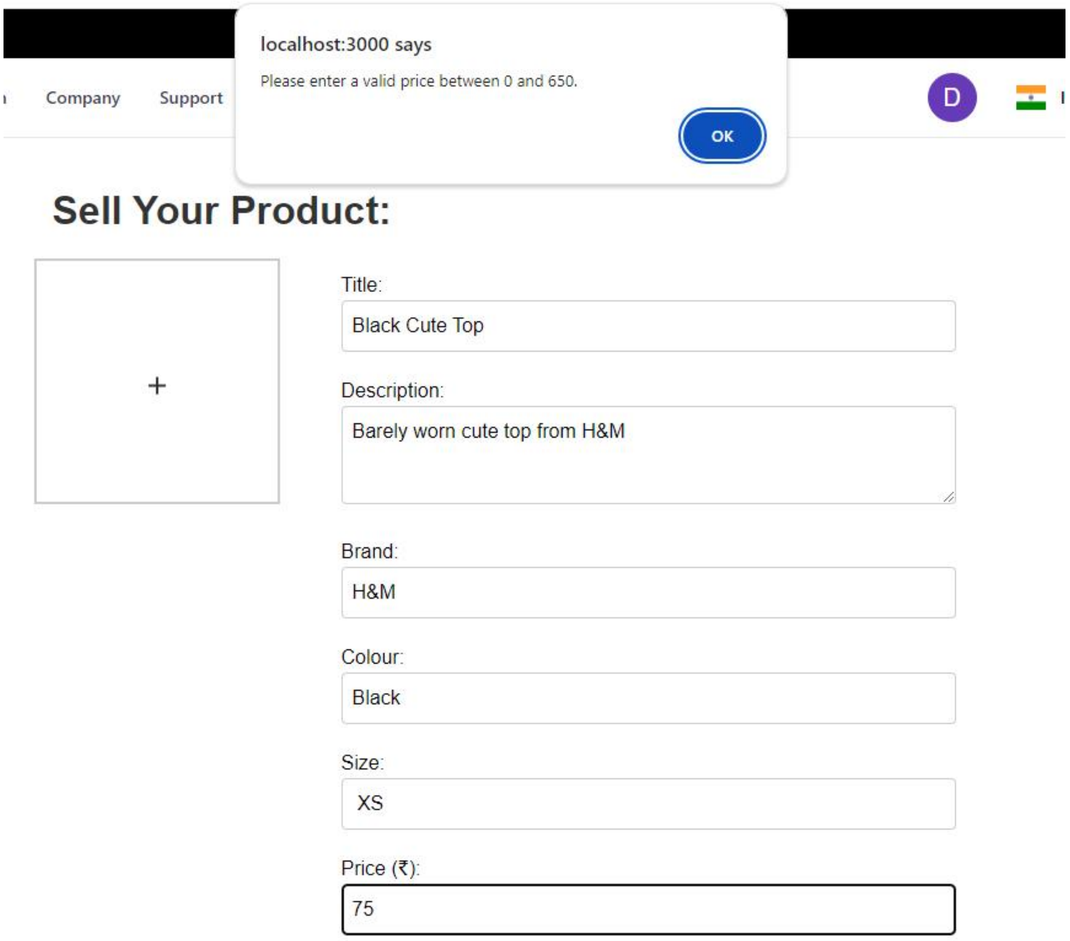
Password * ••••••



EQUIVALENCE PARTITION TESTING

The following is the EVP for price field in sell page. The range set is 0-650 rupees.

EQUIVALENCE PARTITION TESTING		
Invalid	Invalid	Valid
>0	<650	0-650



localhost:3000 says

Please enter a valid price between 0 and 650.

Sell Your Product:

Title: Black Cute Top

Description: Barely worn cute top from H&M

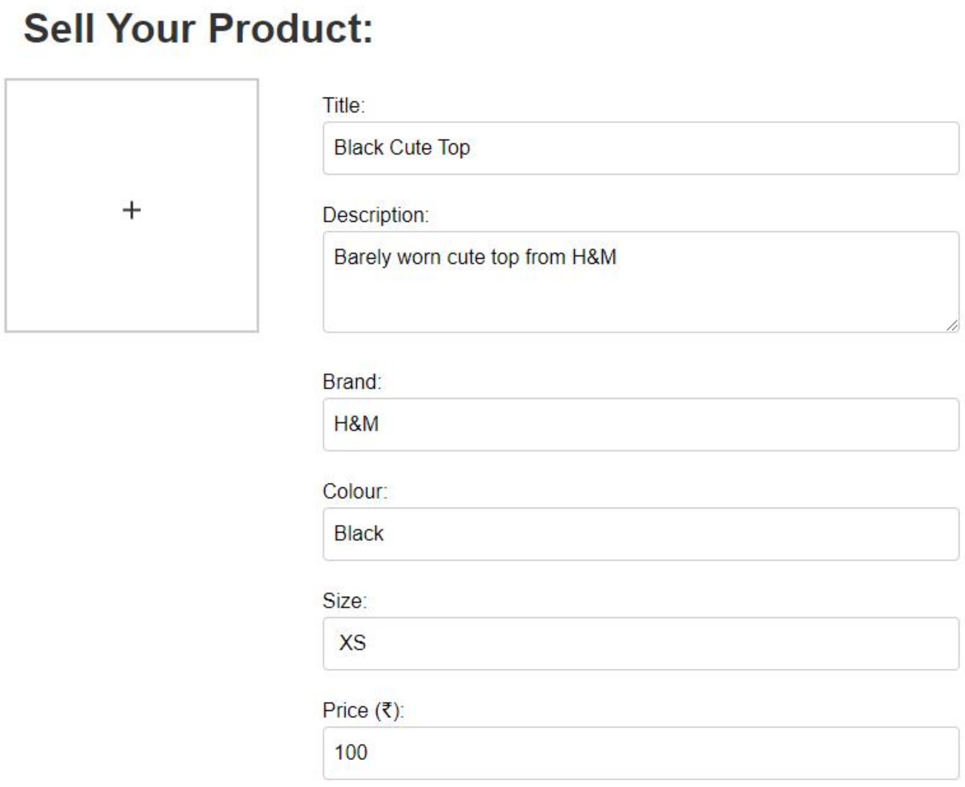
Brand: H&M

Colour: Black

Size: XS

Price (₹): 75

The system won't accept any value beyond this as it would exceed the range.



Sell Your Product:

Title: Black Cute Top

Description: Barely worn cute top from H&M

Brand: H&M

Colour: Black

Size: XS

Price (₹): 100

CYCLOMATIC COMPLEXITY

Source Code:

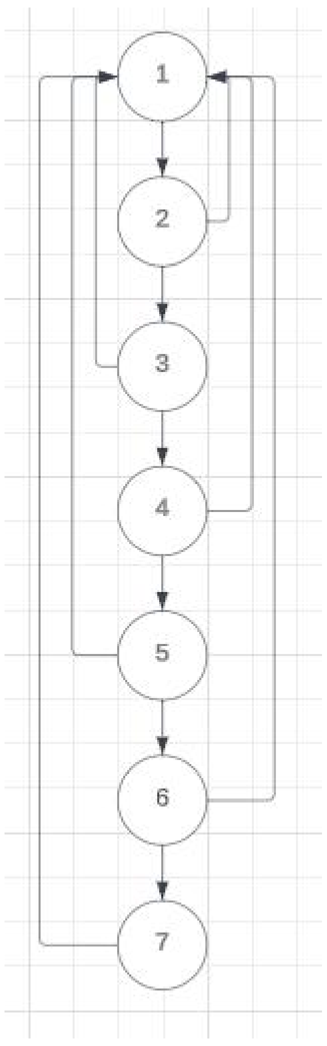
```
import React, { useState } from "react";
import EmailIcon from "@mui/icons-material/Email";
import InstagramIcon from '@mui/icons-material/Instagram';

const Contact = () => {
 const [formData, setFormData] = useState({
   name: "",
   email: "",
   subject: "",
   message: "",
 });
 const [errors, setErrors] = useState({});

 const handleChange = (e) => {
   setFormData({ ...formData, [e.target.name]: e.target.value });
   setErrors({ ...errors, [e.target.name]: "" });
 };
 const handleSubmit = (e) => {
   e.preventDefault();

   const newErrors = {};
   if (!formData.name.trim()) {
     newErrors.name = "Name is required";
   }
   if (!formData.email.trim()) {
     newErrors.email = "Email is required";
```

```
  } else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(formData.email)) {
    newErrors.email = "Invalid email format";
  }
  if (!formData.subject.trim()) {
    newErrors.subject = "Subject is required";
  }
  if (!formData.message.trim()) {
    newErrors.message = "Message is required";
  }
  if (Object.keys(newErrors).length > 0) {
    setErrors(newErrors);
    return;
  }
  alert("Message sent successfully!");
  setFormData({
    name: "",
    email: "",
    subject: "",
    message: "",
  });
};
export default Contact;
```

Cyclomatic Complexity = $\pi + 1$

$= 6 + 1$

$= 7$



12.3 FUNCTIONAL TESTING

Functional testing is meant to test the application against business requirements. Functional testing is designed to test the features, functionalities, and manner of interactions for users to ensure proper operation is observed within the scope of specified requirements while giving proper outcomes. End of humanized content.

The testers review the functions and behaviours of the software, so the code developed for the same has to interact with the business requirements or specifications made for the same purpose, thereby helping in making sure that the software exactly, effectively, and reliably performs its intended tasks.

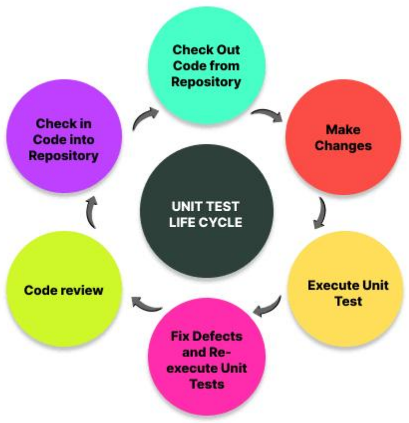
Functional testing is utilized to verify that the software provides the intended end-user behaviour. It helps to find defects, errors, and such deviations in requirements during the early phase of the development lifecycle. Hence, they are properly taken care of, thereby making the software "functionally" perfect and reliable for end use.

12.4 UNIT TESTING

The individual software application program unit or component is included and tested at a level that includes: unit testing, and the level of software testing is unit testing. Generally, each new code unit with other independent parts at usually the function or method level in unit testing is tested.

The primary goals of unit testing are to:

- Ensure that each unit of code behaves as expected according to its specifications and requirements.
- Identify defects or errors in individual units of code early in the development process.
- Facilitate code refactoring and maintenance by providing a safety net against unintentional changes or regressions.


Check Out Code from Repository
Make Changes
Execute Unit Test
Fix Defects and Re-execute Unit Tests
Code review
Check in Code into Repository
UNIT TEST LIFE CYCLE

Unit Testing Techniques:

Black Box Testing - Using which the user interface, input and output are tested.

White Box Testing - used to test each one of those functions behaviour is tested.

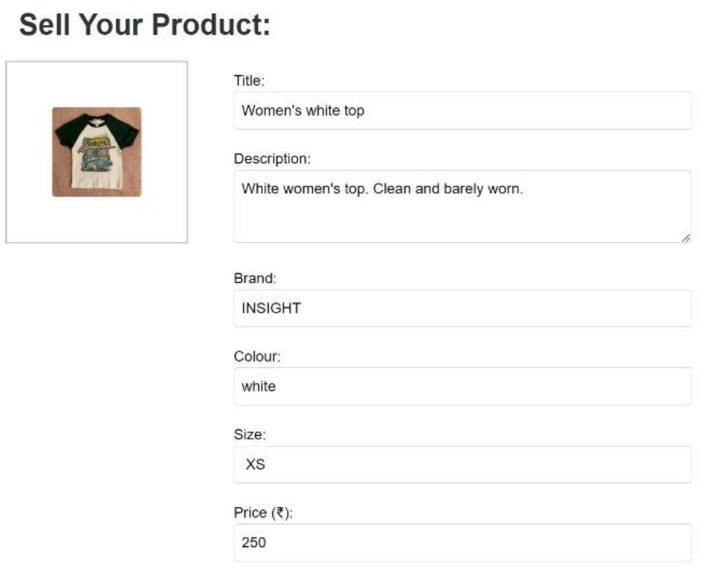
Gray Box Testing - Used to execute tests, risks and assessment methods.

Unit testing for Selling an item:

The user will click on their account avatar and select “Sell” from the provided menu. An instructions page will appear in front of the user, the user will read the instructions and click on the “Upload” button below. The user then will be redirected to the Sell page where they have to fill out basic details about the item and finally submit.

Sr. No.	Input	Expected Output	Observed Output
1.	All fields filled	A preview on how the item will look like on the	A preview on how the item will look like on the

		application and an alert "Your item has been uploaded for sale"	application and an alert "Your item has been uploaded for sale"
2.	Missing field: image	"Please fill in all fields"	"Please fill in all fields"
3.	Missing field: title	"Please fill in all fields"	"Please fill in all fields"
4.	Missing field: price	"Please fill in all fields"	"Please fill in all fields"
5.	Missing field: category	"Please fill in all fields"	"Please fill in all fields"



Sell Your Product:

Title: Women's white top

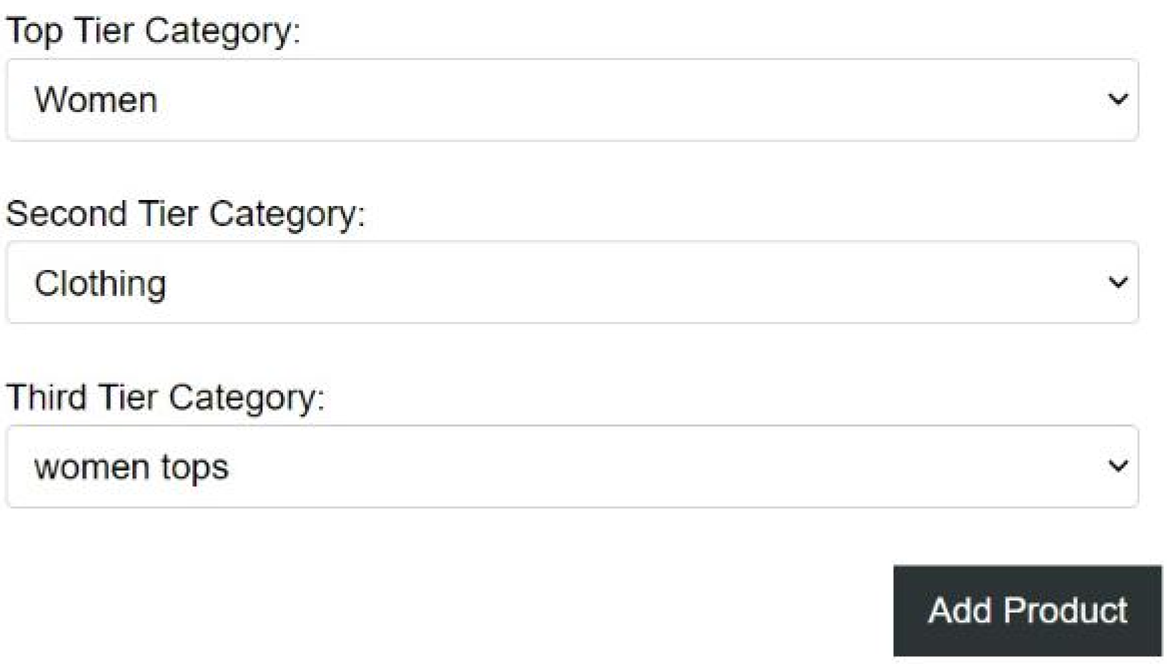
Description: White women's top. Clean and barely worn.

Brand: INSIGHT

Colour: white

Size: XS

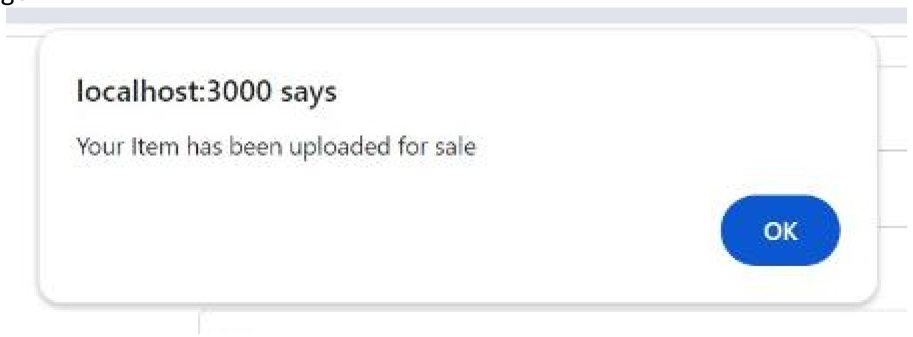
Price (₹): 250



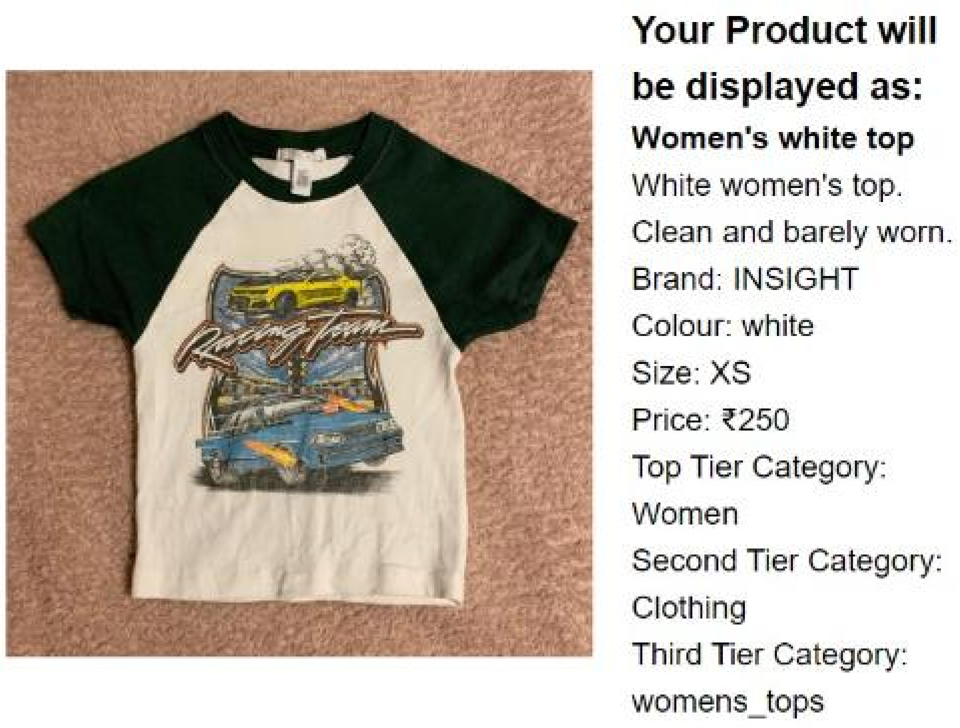
Top Tier Category: Women

Second Tier Category: Clothing

Third Tier Category: women tops

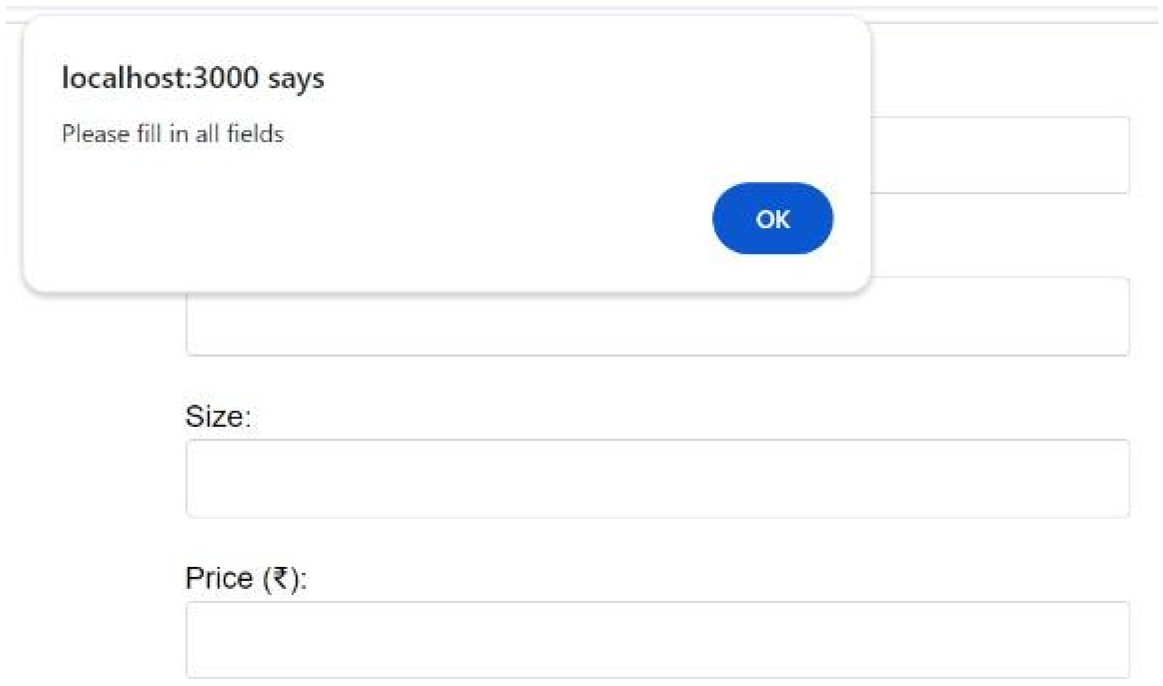

localhost:3000 says

Your Item has been uploaded for sale


Your Product will be displayed as:

Women's white top

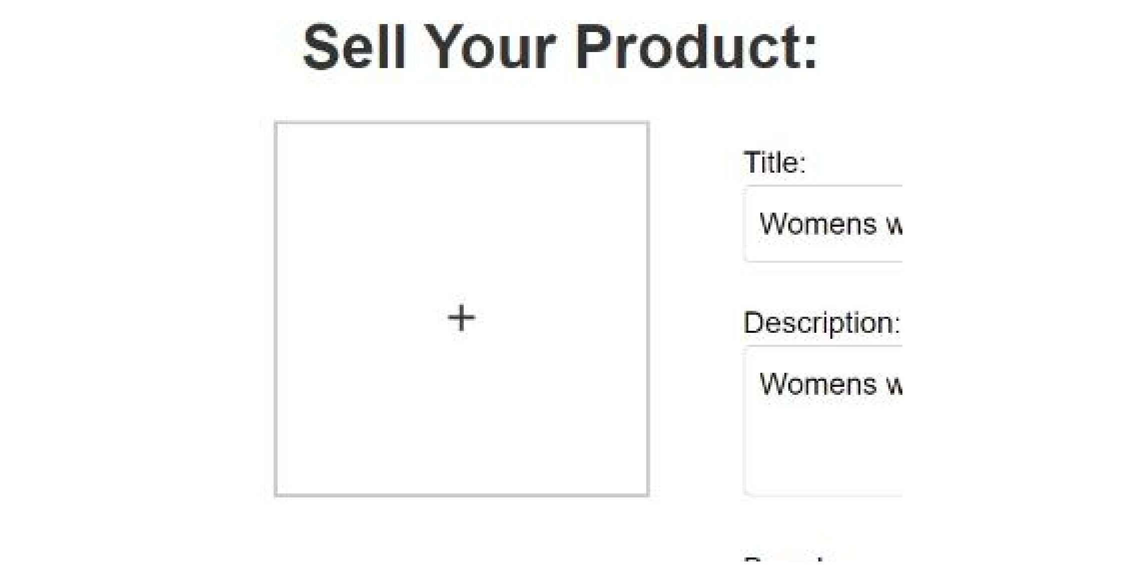
White women's top.
Clean and barely worn.
Brand: INSIGHT
Colour: white
Size: XS
Price: ₹250
Top Tier Category: Women
Second Tier Category: Clothing
Third Tier Category: womens_tops


localhost:3000 says

Please fill in all fields

Size:

Price (₹):



Sell Your Product:

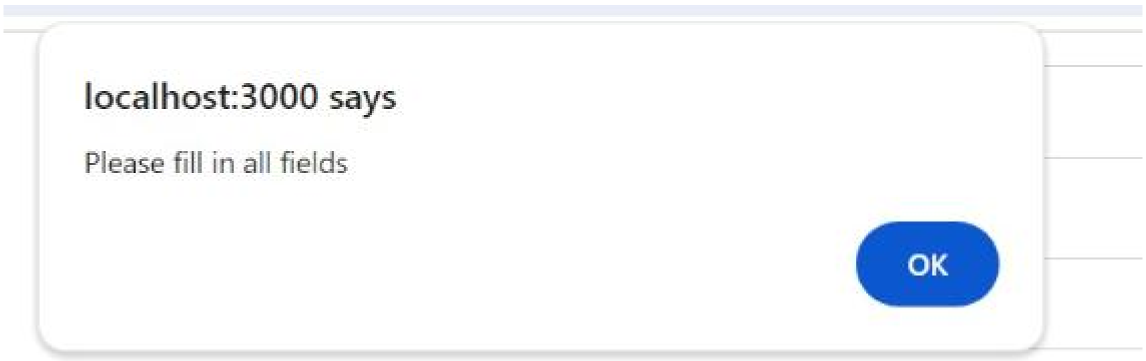
+

Title:

Womens w

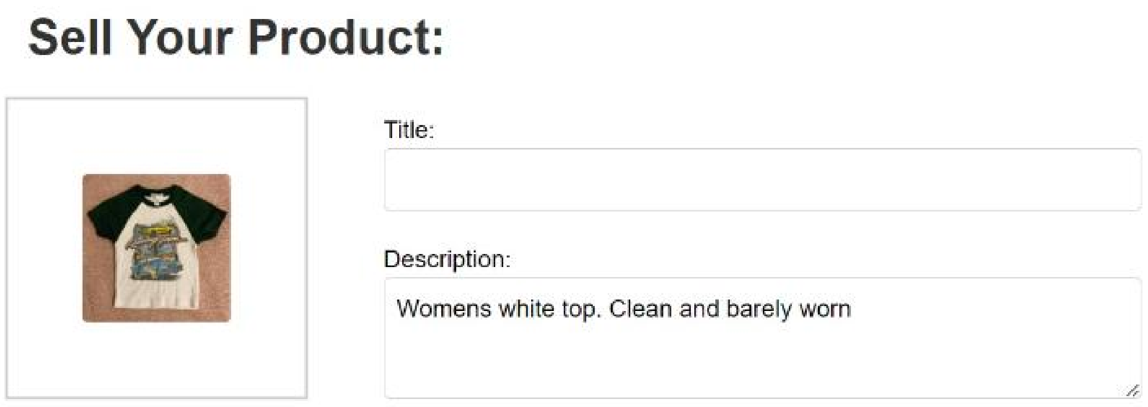
Description:

Womens w



localhost:3000 says

Please fill in all fields

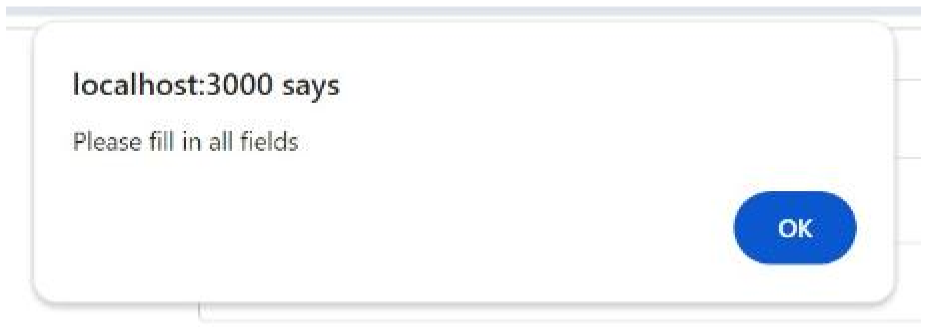


Sell Your Product:

Title:

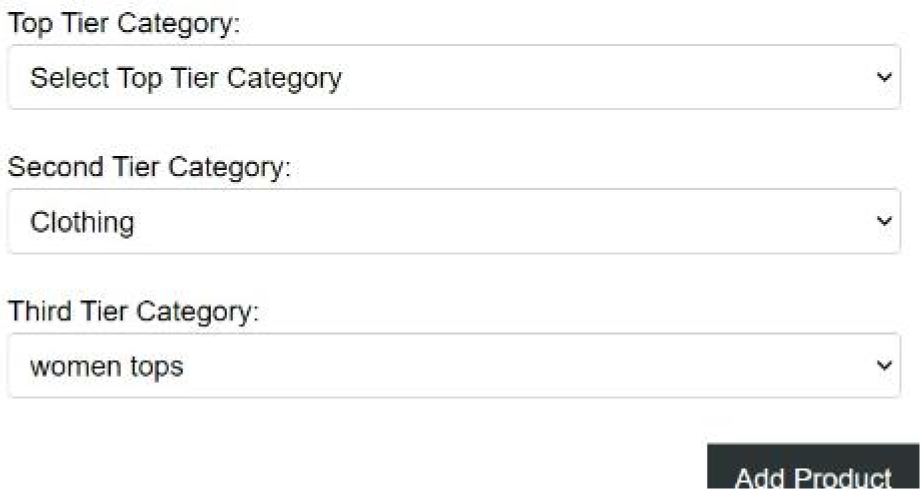
Description:

Womens white top. Clean and barely worn



localhost:3000 says

Please fill in all fields



Top Tier Category:

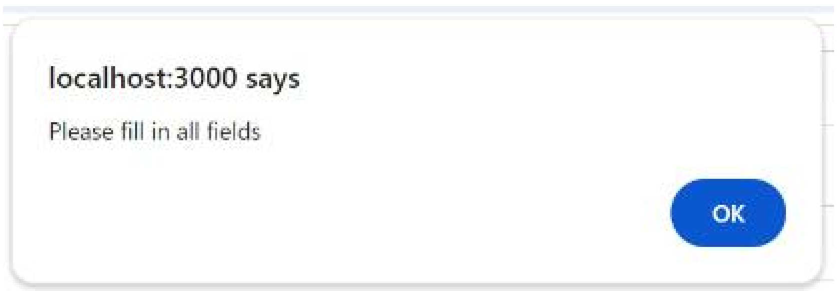
Select Top Tier Category

Second Tier Category:

Clothing

Third Tier Category:

women tops



localhost:3000 says

Please fill in all fields

12.5 INTEGRATION TESTING

Integration testing is a software testing technique that focuses on verifying the interaction between different modules or components of a software system. It aims to validate that the integrated components work together as expected and produce the desired outcomes.

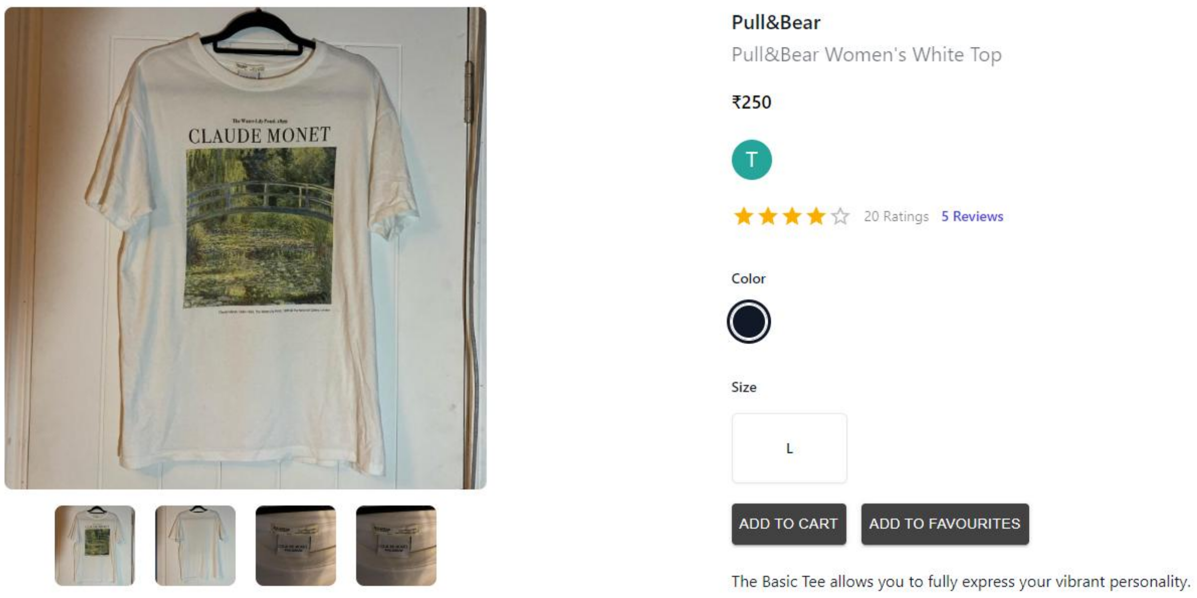
During integration testing, individual modules or units of code are combined and tested as a group to ensure that they interact correctly and exchange data or messages accurately. Integration testing helps identify defects or errors that may arise due to interactions between components, such as interface mismatches, data flow issues, or communication failures.

The primary goals of integration testing are to:

- Validate the correctness and completeness of the interactions between integrated components.
- Ensure that data flows smoothly between components and across subsystem boundaries.
- Identify and resolve defects or issues related to component integration early in the development process.

By performing integration testing, software development teams can ensure that the integrated software system functions as a cohesive unit and meets the specified requirements and quality standards.

Module	WHY	Desired Output	Observed Output
User	To test whether the product is added to the cart	The product should appear in the cart	The product has appeared in the cart


Pull&Bear

Pull&Bear Women's White Top

₹250

T

★★★★☆ 20 Ratings 5 Reviews

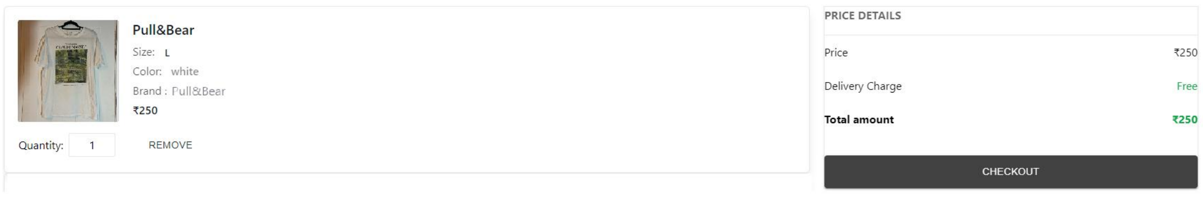
Color

Size

L

ADD TO CART ADD TO FAVOURITES

The Basic Tee allows you to fully express your vibrant personality.


Pull&Bear

Size: L

Color: white

Brand : Pull&Bear

₹250

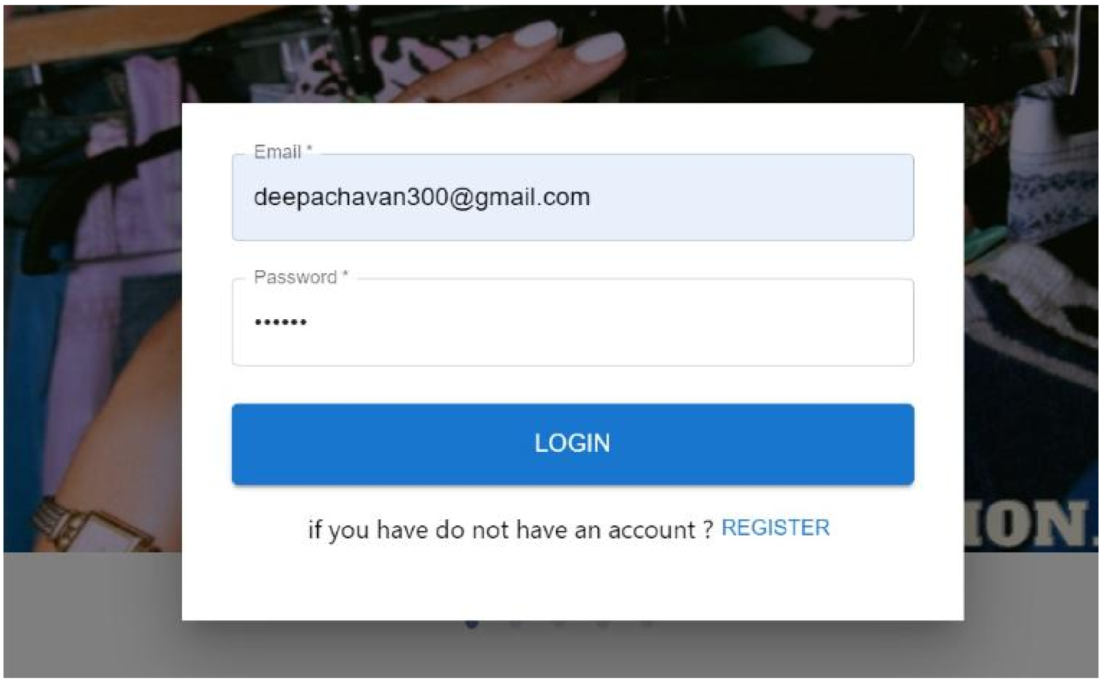
Quantity: 1 REMOVE

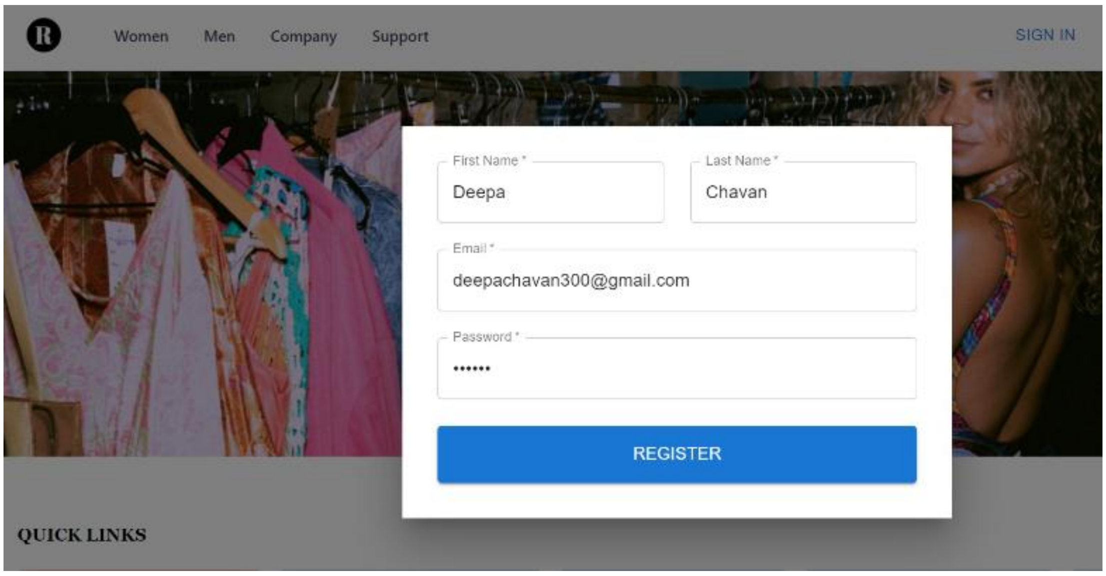
PRICE DETAILS

Price	₹250
Delivery Charge	Free
Total amount	₹250

CHECKOUT

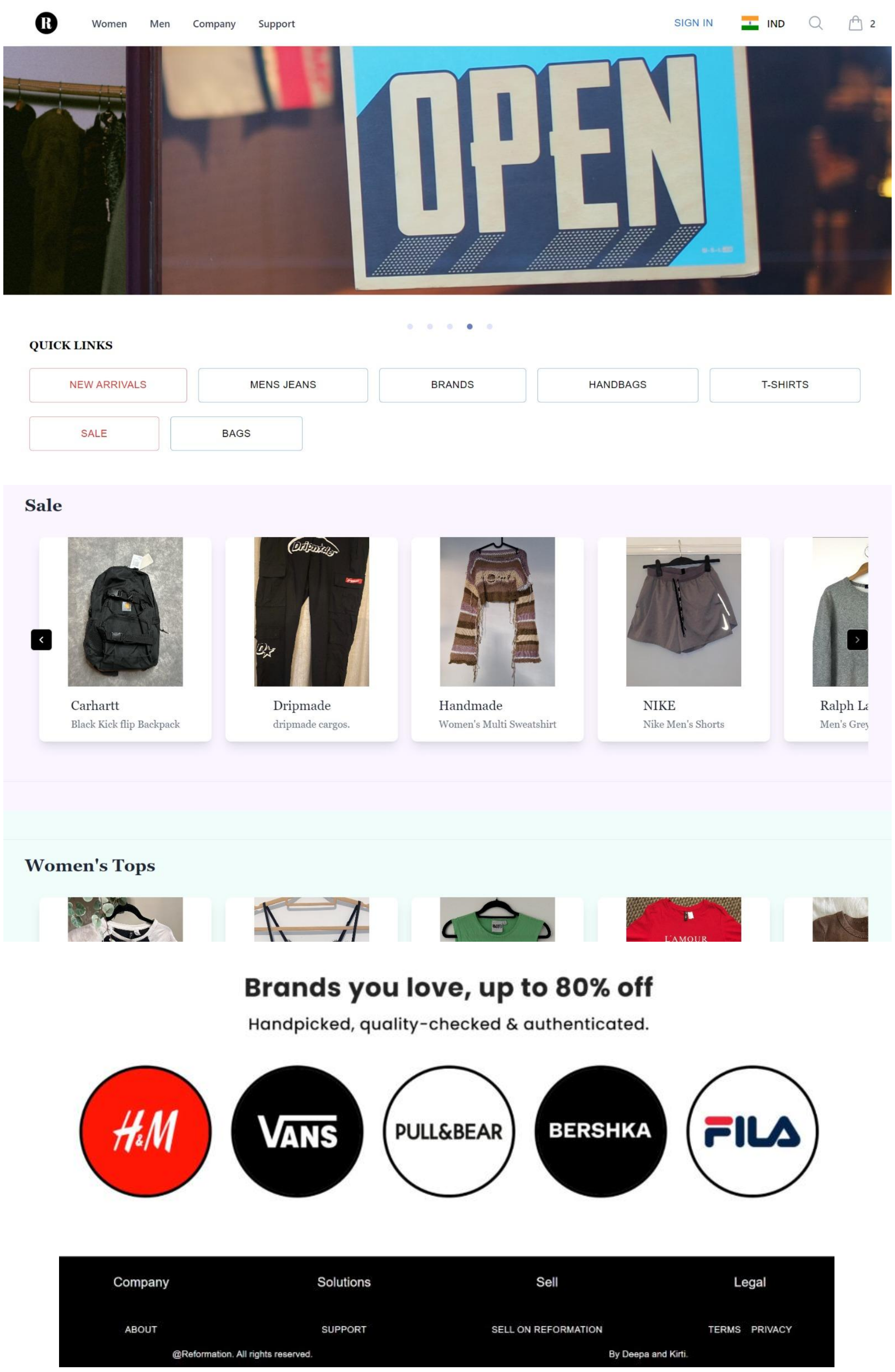
13. SCREENSHOTS


Email *
deepachavan300@gmail.com
Password *
••••••
LOGIN
if you have do not have an account ? REGISTER



First Name *
Deepa
Last Name *
Chavan
Email *
deepachavan300@gmail.com
Password *
••••••
REGISTER

HOMEPAGE


Women Men Company Support SIGN IN IND 2

QUICK LINKS

NEW ARRIVALS | MENS JEANS | BRANDS | HANDBAGS | T-SHIRTS | SALE | BAGS

Sale

Carhartt
Black Kick flip Backpack

Dripmade
dripmade cargos.

Handmade
Women's Multi Sweatshirt

NIKE
Nike Men's Shorts

Ralph La
Men's Grey

Women's Tops

Brands you love, up to 80% off

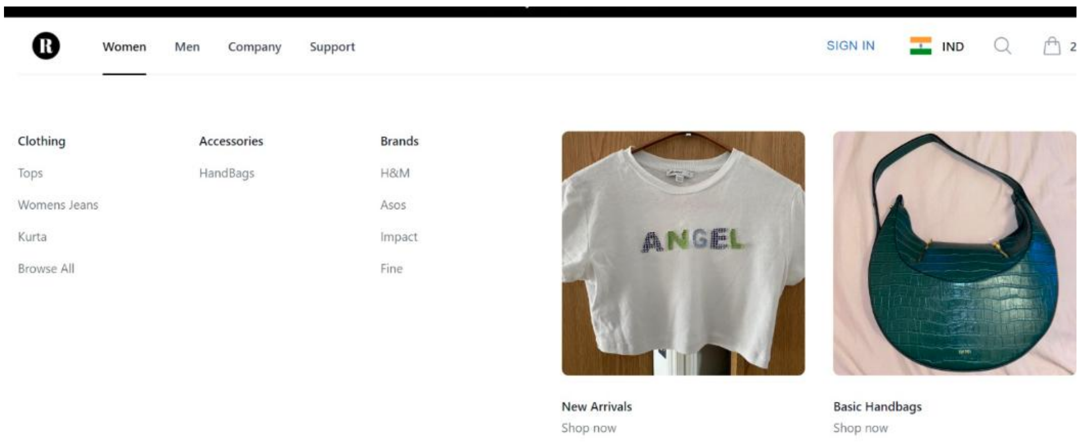
Handpicked, quality-checked & authenticated.

Company | Solutions | Sell | Legal

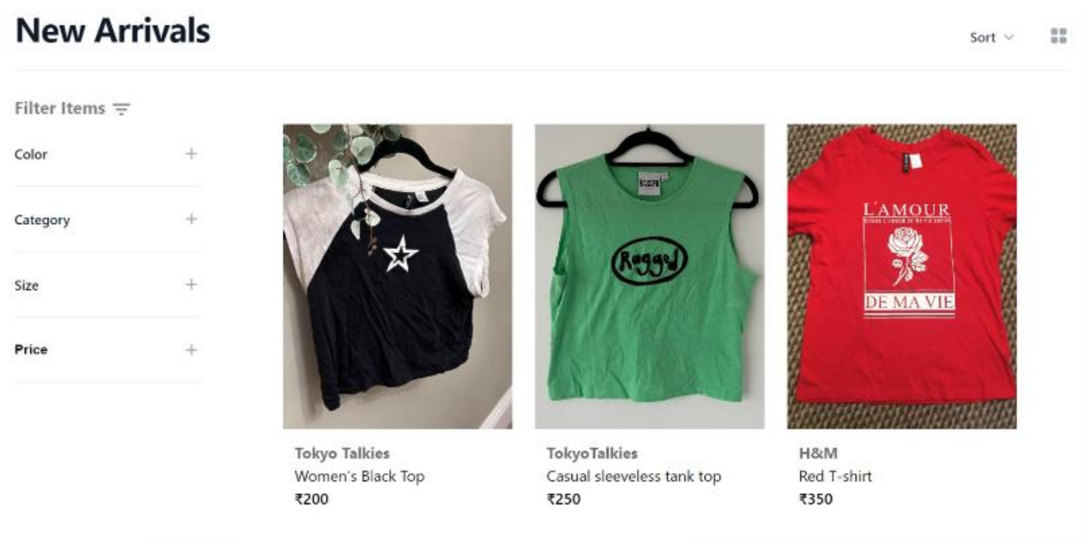
ABOUT | SUPPORT | SELL ON REFORMATION | TERMS PRIVACY

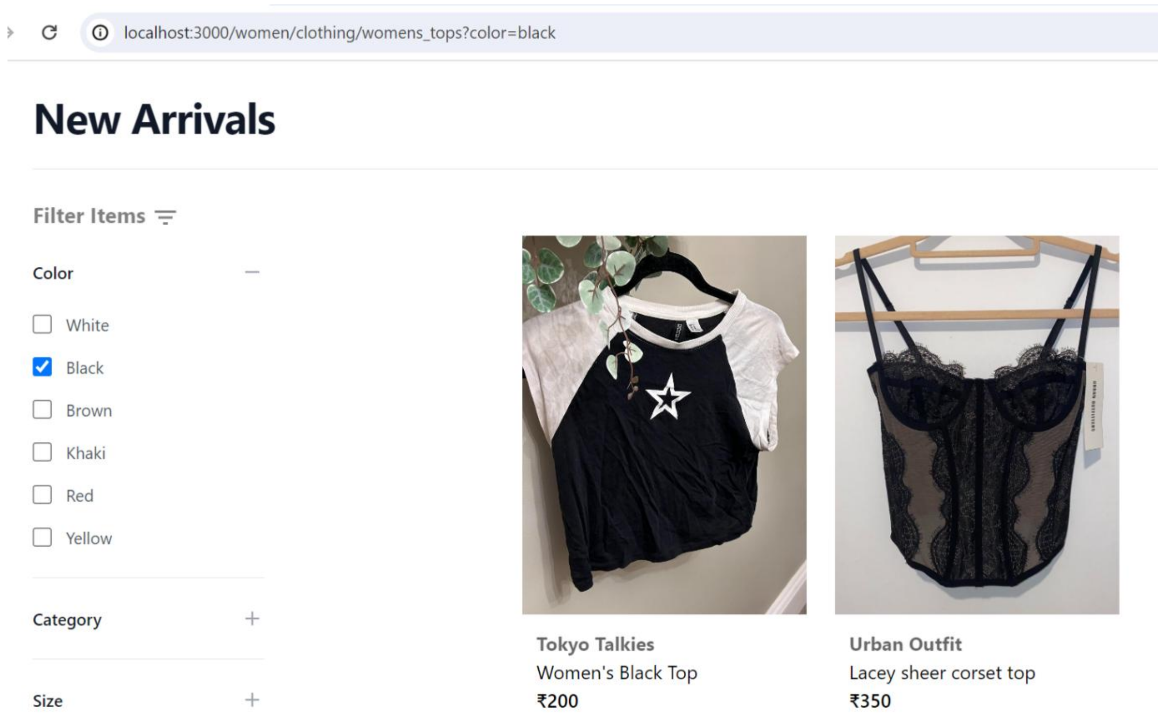
@Reformation. All rights reserved. By Deepa and Kirti.

NAVIGATION BAR

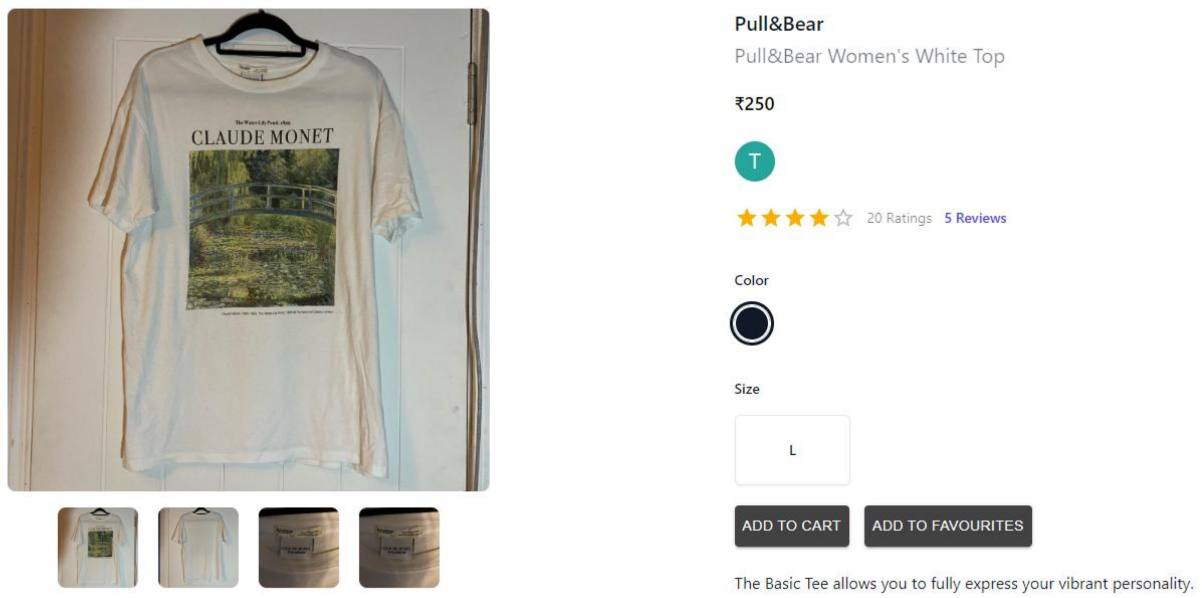


PRODUCT LIST PAGE





PRODUCT DETAIL PAGE



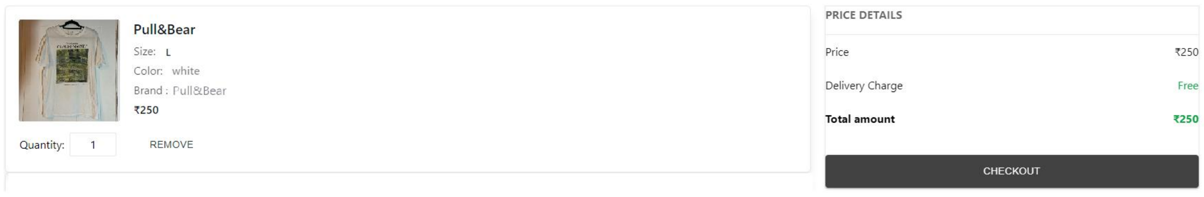
Pull&Bear

Pull&Bear Women's White Top

₹250

The Basic Tee allows you to fully express your vibrant personality.

CHECKOUT PAGE



Pull&Bear

Size: L

Color: white

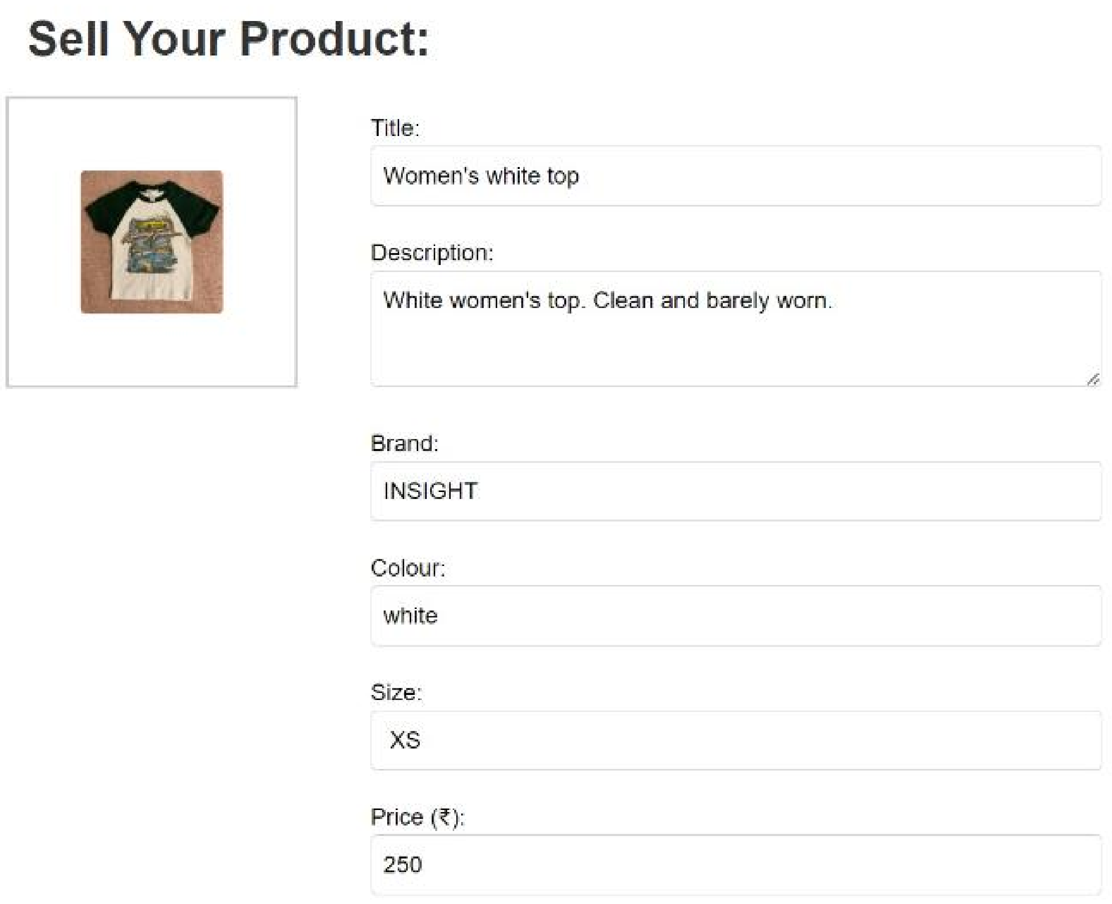
Brand : Pull&Bear

₹250

Quantity: 1

PRICE DETAILS

Price	₹250
Delivery Charge	Free
Total amount	**₹250**



Sell Your Product:

Title:
Women's white top

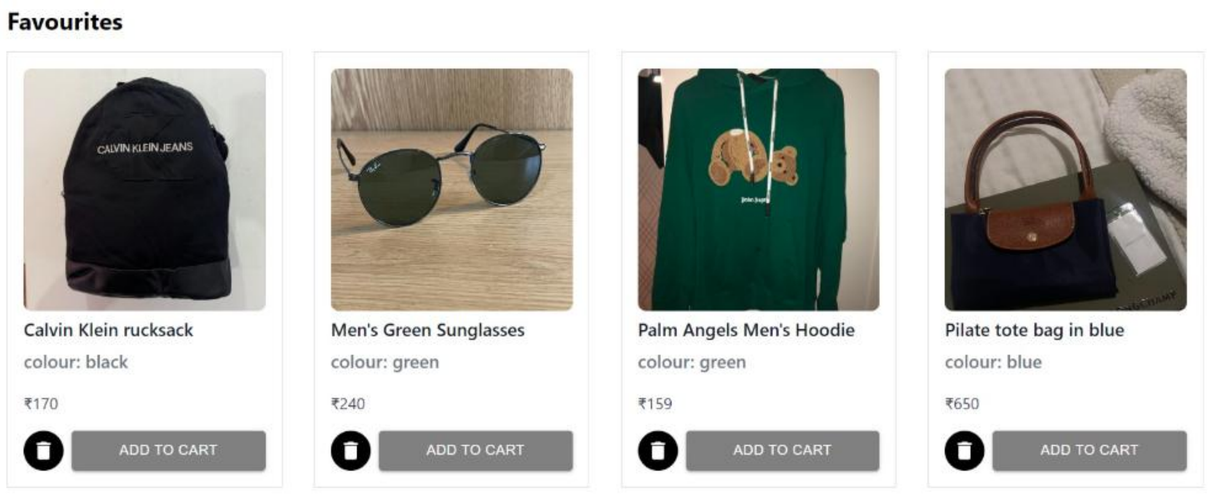
Description:
White women's top. Clean and barely worn.

Brand:
INSIGHT

Colour:
white

Size:
XS

Price (₹):
250



Favourites

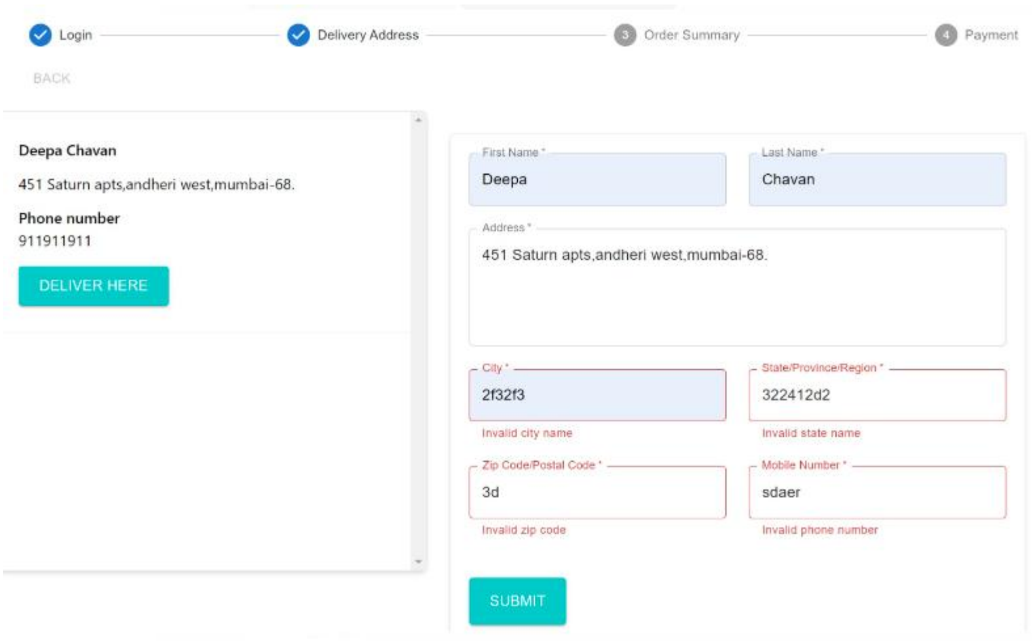
Calvin Klein rucksack
colour: black
₹170

Men's Green Sunglasses
colour: green
₹240

Palm Angels Men's Hoodie
colour: green
₹159

Pilate tote bag in blue
colour: blue
₹650

DELIVERY ADDRESS PAGE



Deepa Chavan

451 Saturn apts,andheri west,mumbai-68.

Phone number
911911911

First Name * Deepa

Last Name * Chavan

Address * 451 Saturn apts,andheri west,mumbai-68.

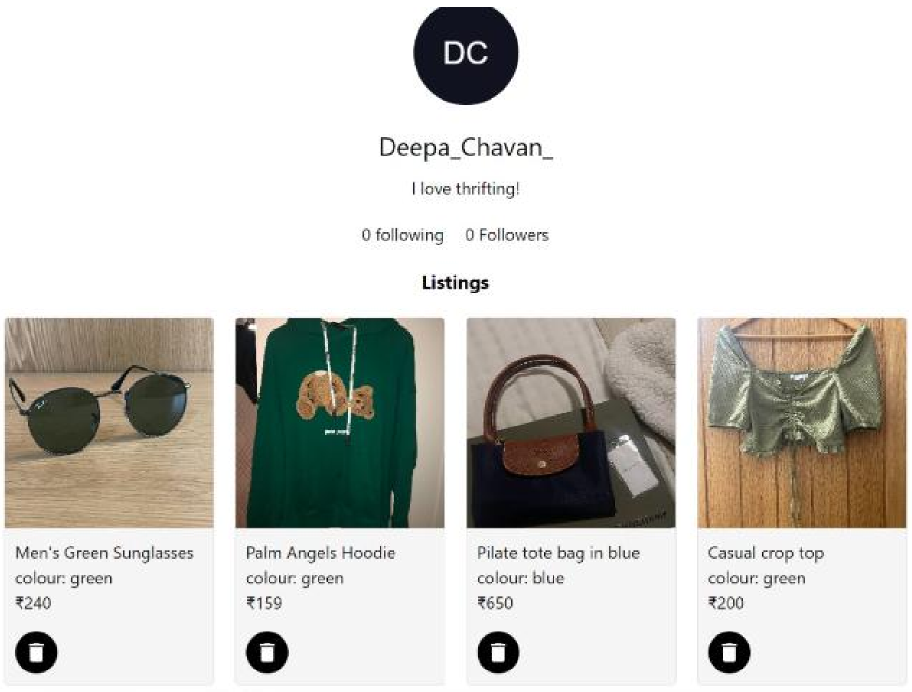
City * 2f32f3
Invalid city name

State/Province/Region * 322412d2
Invalid state name

Zip Code/Postal Code * 3d
Invalid zip code

Mobile Number * sdaer
Invalid phone number

PROFILE PAGE



Deepa_Chavan_

I love thrifting!

0 following 0 Followers

Listings

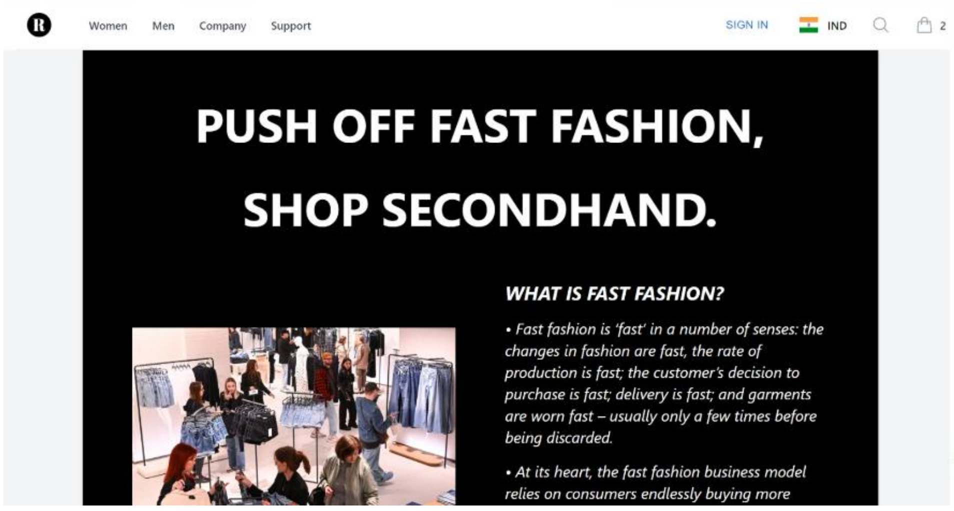
Men's Green Sunglasses
colour: green
₹240

Palm Angels Hoodie
colour: green
₹159

Pilate tote bag in blue
colour: blue
₹650

Casual crop top
colour: green
₹200

COMPANY PAGE



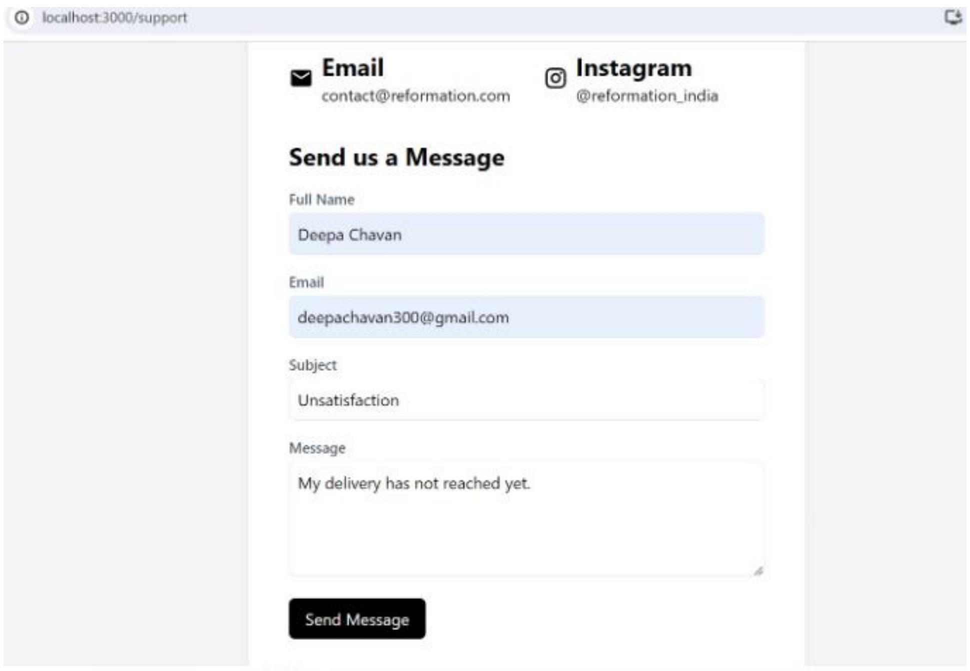
PUSH OFF FAST FASHION, SHOP SECONDHAND.

WHAT IS FAST FASHION?

• Fast fashion is 'fast' in a number of senses: the changes in fashion are fast, the rate of production is fast; the customer's decision to purchase is fast; delivery is fast; and garments are worn fast – usually only a few times before being discarded.

• At its heart, the fast fashion business model relies on consumers endlessly buying more

SUPPORT PAGE


localhost:3000/support

Email
contact@reformation.com

Instagram
@reformation_india

Send us a Message

Full Name: Deepa Chavan

Email: deepachavan300@gmail.com

Subject: Unsatisfaction

Message: My delivery has not reached yet.

14. LIMITATIONS AND FUTURE SCOPE

LIMITATIONS:

- **Limited Product Availability:** The availability of second-hand gadgets can vary relying on consumer listings, probably ensuing in constrained choice for positive regions or categories.
- **Quality Assurance:** Ensuring the first-class and authenticity of second-hand gadgets may be difficult, leading to capacity inconsistencies among product descriptions and the real situations.
- **Geographic Restrictions:** Some dealers may additionally best provide transport within a certain geographic area, proscribing availability of objects to users in other locations.
- **No Warranty:** Most of the items sold on this platform are second-hand, so you won't get any warranty or guarantee which could be trouble if the items are defective or not as described.
- **Security Concerns:** There is always a risk of possibly compromising user transactions, despite the platform's best efforts.

FUTURE SCOPE:

- A search function can be added to search for products based on title, brand, etc.
- An AI recommendation system can be added to enhance the user experience.
- A chatbot can be added to solve user queries right away.
- Minor customisation can be added for the admin module to give them more control over the application.
- Further developing and releasing a mobile application version of the platform to allow convenient access to users' smartphones and tablets, increasing convenience of use and expanding the user base.

15. CONCLUSION

CONCLUSION:

In conclusion, Reformation gives an exciting opportunity for users to participate in sustainable consumption practices while taking part in the advantages of online shopping. By imparting a user-friendly platform for buying and selling second hand items, the utility promotes environmental cognizance and fosters community engagement. Despite its limitations, together with confined product availability, the platform offers a precious opportunity to standard retail and contributes to the circular economic system. With continuous development and consumer feedback, Reformation has the capability to become the go-to spot for thrift shopping fans within the country, imparting a diverse range of pre-loved treasures and promoting a greater sustainable future for all.

16. REFFERENCES

REFFERENCES:

https://react.dev/
https://mui.com/
https://nodejs.org/api/all.html
https://heroicons.com/
https://www.mongodb.com/
https://www.postman.com/