Cloud Store

Project Report Submitted by

Ashish Sam T George

Reg. No.: AJC18MCA-I025

In Partial fulfillment for the Award of the Degree Of

INTEGRATED MASTER OF COMPUTER APPLICATIONS (INMCA)
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING KANJIRAPPALLY

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

2022 - 2023

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**Cloud Store"** is the bona-fide work of **ASHISH SAM T GEORGE (Regno: AJC18MCA-I025)** in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2022-23.

Navyamol K.T **Gloriya Mathew**

Internal Guide **Coordinator**

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report **"Cloud Store"** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University, during the academic year 2022-2023.

Date: **ASHISH SAM T GEORGE**

KANJIRAPPALLY **Reg:. AJC18MCA-I025**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to my family and to all who helped me in completing this project successfully. I wish to express my sincere gratitude to our manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

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

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project.

ASHISH SAM T GEORGE

ABSTRACT

E-commerce aggregation system provides a centralized access to all the products from the nearby shops of a customer. It enables fast delivery of the products. Customers don't have to worry about selecting the shops to buy the products from. The system takes care of choosing the products from the shops according to specific metrics like location and rating. Customers can choose custom locations to which the products can be delivered.

The system focuses on consumer products like stationary items, grocery and home appliances. Shop owners can easily add products by adding from the already existing list or by creating their own custom ones. When an order is made by the customer, a notification will be sent to all the nearby shops which have the specified products. The shops that accept the order will be selected on the basis of their rating. The selected shop will be notified with the order details.

Some of the core functionalities offered are:

- Shop Registration

  Shops are able to register themselves. They can add their products to the system. It is mandatory for every shop to have a location and it remains static unless changed by the shop themselves.

- Customer Registration

  Customers have the freedom to look at the various products but can only shop after logging in. Customers have the option to change their location anytime. Customers can add multiple products to their cart.

- Product Management

  Products are added by the shops. Each product is different. Even a small difference like color will classify the product as a separate one. Products are visible under categories, which will help the customers to find products.

- Admin Functionality

  Administrators do not require any location, to be registered on the system. Administrators can add, remove and modify the details of the shops and customer.

CONTENT

List of Abbreviation

Eg. IDE - Integrated Development Environment

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

'**Cloud Store**' is a web application that can be used to buy products which are available near the customer. Customers can easily buy products without the need to visit each shop. Since the products are available nearby, it will be delivered to the customer without significant delay. The customer can reduce the time, effort, and cost of buying quality products. Shops can register on the web application and can sell their products after getting verified. Shops have the option to upload their verification document. Once the documents are verified by the administrator, the shops can add products and can sell them.

Customers can only see products which are available near them. The main objective of this system is to strengthen the local shops by enhancing the shopping experience of the customers. It ensures that every shop has equal opportunity to thrive. The customers can have product-based purchasing, rather than shop-based purchasing. The website provides a user-friendly interface which makes both selling and purchasing of different kinds of products from different shops easy.

1.2 PROJECT SPECIFICATION

The proposed system is a web application through which shops can sell a variety of products and the customers can buy the available products. Some of the core modules are:

- Shop Registration

Shops are able to register themselves. They can add their products to the system. It is mandatory for every shop to have a location and it remains static unless changed by the shop themselves.

- Customer Registration

Customers have the freedom to look at the various products but can only shop after logging in. Customers have the option to change their location anytime. Customers can add multiple products to their cart.

- Product Management

Products are added by the shops. Each product is different. Even a small difference like color will classify the product as a separate one. Products are visible under categories, which will help the customers to find products easier.

- Admin Functionality

Administrators do not require any location, to be registered on the system. Administrators can add, remove and modify the details of the shops and customers.

- Location System

Location system is one of the core components of this aggregation system. Every shop and every customer will have a location associated with them at any point of time. As the location changes, the products shown to the customer, as well as the shops, changes. The products are shown to the customer according to the location of the customer.

- Payment System

Customers can pay the amount for the products through the payment gateway integrated in the system itself. The customers will get an email confirming the purchase of the products.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

System analysis is a review of a technological system, like a software package, for troubleshooting, development, or improvement purposes. Through in-depth analysis, analysts can uncover errors in code, accessibility issues for end-users or design incompatibilities. Performing an effective systems analysis often requires experts to have knowledge of a software product's or package's requirements so that they can approach their analysis effectively. Unlike systems administrators, who focus on day-to-day system maintenance, systems analysts consider the viability and effectiveness of a product overall. This allows them to suggest changes or make fixes that improve the system.

With a systems analysis, considering the goals of the system is important for solving problems and creating efficiencies. From there, dividing a system into components can make it easier to perform individual analyses that influence the complete system.

2.2 EXISTING SYSTEM

Zomato is an example of existing system. Zomato is a multinational restaurant aggregator and food delivery company. It connects the various restaurants and customers together. Customers can order food without going out of their home. Dine-in bookings are also provided for the customers. Customers can search and discover restaurants and food with respect to the location, post and see reviews, order food online, book tables and make payments on the go. In zomato, customers can search by food and restaurant. If they are making the search based on food, then they have to select the restaurant from which the food is to be ordered.

2.3 DRAWBACKS OF EXISTING SYSTEM

- The products are limited to food and beverages.
- Customers have to choose from the list of restaurants.
- Prices for the same product can be different in different shops.
- We cannot ensure that the reviews are not manipulated by the organization itself. Reviews greatly impact the decision of the customers.

PROPOSED SYSTEM

CloudStore is an ecommerce aggregation system. It connects the various shops and customers together. Customers can order the products anytime. Unlike zomato, the shopping experience will be product-based and not store-based. The product results will be shown based on the location of the customer. The system accepts both shop and product reviews from the customers.

2.4 ADVANTAGES OF PROPOSED SYSTEM

- The products are not limited to food and beverages.
- The system decides the shop to buy the products from, based on location and rating.
- Price of the product will always be near to the market price.
- The reviews cannot be manipulated, even by the organization.
- Can enhance the small-scale shops.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A feasibility study is a detailed analysis that considers all of the critical aspects of a proposed project in order to determine the likelihood of it success. A feasibility study is an assessment of the practicality of a proposed plan or project. A feasibility study analyzes the viability of a project to determine whether the project or venture is likely to succeed. The study is also designed to identify potential issues and problems that could arise while pursuing the project.

The document provides the feasibility of the proposed system that is being developed and three aspects of the feasibility of a project are considered such as technical, economical, and behavioral during the feasibility study.

3.1.1 Economical Feasibility

Economic feasibility analysis is the most commonly used method for determining the efficiency of a new project. It is also known as cost analysis. It helps in identifying profit against investment expected from a project. Cost and time are the most essential factors involved in this field of study.

The economic feasibility study for projects shows all the costs necessary for the project, which are the costs of establishing the project, as well as the size of the investment, estimating the size of the project's profits, and knowing the net profit during a specific period of time. and tools needed for the project.

The proposed system is developed as part of project work, thus there is no manual cost spent for the proposed system. All the required resources were already available, which shows that the system is economically feasible for development. The project was developed at a low cost as it is completely developed using open-source software.

1. Do the resources needed exist ?

   Yes.

2. Will the proposed application lead to better use of resources to improve health outcomes, when compared with other options ?

   Yes

3.1.2 Technical Feasibility

The technical feasibility study can be defined as the study related to all the technical aspects of the project. It includes defining the technical specifications of the product and the size of the project, and preparing the necessary labor schedules for production.

In technical feasibility the following issues are taken into consideration.

- Whether the required technology is available or not
- Whether the required resources are available

The proposed system is developed using Vue.js in frontend and Java in the backend along with MongoDB as the database and Spring boot as the framework of Java.

1. Do Stakeholders have the expertise needed ?

   Yes.

2. Are additional resources needed ?

   Yes. A smartphone and an internet connection is required.

3. Is the system ready in terms of the technology required ?

   Yes.

3.1.3 Behavioral Feasibility

Behavioral feasibility answers the following questions:

- Whether the proposed system will cause any harm to its users?
- Is this proposed system sufficient to support the users?

The project is behaviorally feasible because it satisfies all the objectives when developed and installed.

1. Do existing system procedures and protocols support the proposed service or initiative?

   Yes.

2. How will key collaborators be involved ?

   Shops can sell products through the web application by using smartphones. Customers can buy products from the web application by using smartphones.

3.1.4 Questionnaire collecting details about the project

1. Do you have a smartphone ?
   Yes.

2. Do you wish to sell products online ?
   Yes.

3. Is your shop open at night ?
   No. The shop closes by 10 P.M every night.

4. Can you sell the product at the average market price ?
   Yes.

5. What kind of products do you sell ?
   Stationary items, Grocery, Bakery Items.

6. Are you comfortable in accepting the orders received online ?
   Yes.

7. Specify the level of comfort for updating the stock when it is empty.
   Very comfortable, as it won't take much time to accept an order.

8. How to know when the stock is back ?
   It can be known when the stock is delivered from the vendors.

9. Is packaging the products possible by the shopkeeper ?
   Yes. We sell the products packaged even when selling in the conventional way.

10. Is there any return policy ?
    Yes. Especially for the bakery items.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	-	Intel core i5 8th Gen
RAM	-	8GB
Hard disk	-	1 TB
SSD	-	250 GB

3.2.2 Software Specification

Front End	-	Vue.js
Backend	-	Java, Spring boot, MongoDB
Client on PC	-	MX-Linux 21
Technologies used	-	JS, HTML5, CSS, Docker

3.3 SOFTWARE DESCRIPTION

3.3.1 Vue

Vue (also known as Vue.js or VueJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained a community of individual developers and companies. Vue can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Nuxt.js. However, vue.js is only concerned with state management and rendering that state to the DOM, so creating Vue applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

3.3.2 Spring

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup. Spring Framework also offers built-in support for typical tasks that an application needs to perform, such as data binding, type conversion, validation, exception handling, resource and event management and more. Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation.

Some of the advantages of Spring Boot are:

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

3.3.3 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL) which is deemed non-free by several distributions.

3.3.4 Docker

Docker is an open source platform for building, deploying, and managing containerized applications. It enables developers to build, deploy, run, update and manage *containers*—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify development and delivery of distributed applications. They have become increasingly popular as organizations shift to cloud-native development and hybrid multi-cloud environments. Docker makes containerization faster, easier and safer.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System Design is the process of designing the architecture, components, and interfaces for a system so that it meets the end-user requirements. system design ranges from discussing the system requirements to product development. System development creates or alters the system so that the processes, practices, and methodologies are changed to develop the system. Therefore, a systematic approach is needed to manage the system requirements and design methodology. It can be classified as logical design and physical design. The logical design represents the abstract dataflow, while the physical design represents the system's input and output processes. It specializes in developing great artwork by saving time and effort. This helps in creating plans for information systems. It is used to solve internal problems, boost efficiency, and broadcast opportunities. It also is the foundation of any business. It contributes a lot to successfully achieving the required results and makes working easier and simpler.

4.1.1 Natural System

Natural System refers to the practice of selling products and services within a single industry and in some cases, a specific geographic area. It relies on operating business hours during a specific period of time and requires housing inventory or occupying a retail store. Such systems handle advertising, inventory shipping and creation of products and services in-house with a staff of employees in close proximity. They do not typically share information with competitors. Natural systems often relies on face to face interaction with consumers and thrives based on word of mouth, networking and customer referrals for new and repeat business. Personal interaction is a key component of businesses such systems. Many businesses network within the community, establish rapport with city leaders and chambers of commerce and sponsor local events and sports teams to develop a relationship with the community to draw in business.

4.1.2 Designed System

Designed system is an integrated information system composed of various electronic technical methods that support business activities. It is a complex application system facing customers, provides various business management and administrative services to directly realize multi-functions of e-commerce. Usually, e-commerce can be conducted through designed system. Together with embedded computer applications, designed systems play an important role in the implementation of e-commerce.

4.2 UML DIAGRAM

The Unified Modeling Language (UML) is a general-purpose, developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

The creation of UML was originally motivated by the desire to standardize the disparate notational systems and approaches to software design. It is the general way to define the whole software architecture or structure.

In Object-Oriented Programming, we solve and interact with complex algorithms by considering themselves as objects or entities. These objects can be anything. It can be the bank or a bank manager too. The object can be a vehicle, animal, machine, etc. The thing is how we interact and manipulate them so that they can perform tasks.

The tasks can be interacting with other objects, transferring data from one object to another, manipulating other objects, etc. The single software could have hundreds or even thousands of objects. So, UML provides us a way to represent and track those objects in a diagram to become a blueprint of our software architecture.

UML includes the following nine diagrams.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

4.2.1 Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures. Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities. They also help identify any internal or external factors that may influence the system and should be taken into consideration. They provide a good high-level analysis from outside the system.

Following are the purposes of a use case diagram:

- It gathers the system's needs.
- It depicts the external view of the system.
- It recognizes the internal as well as external factors that influence the system.

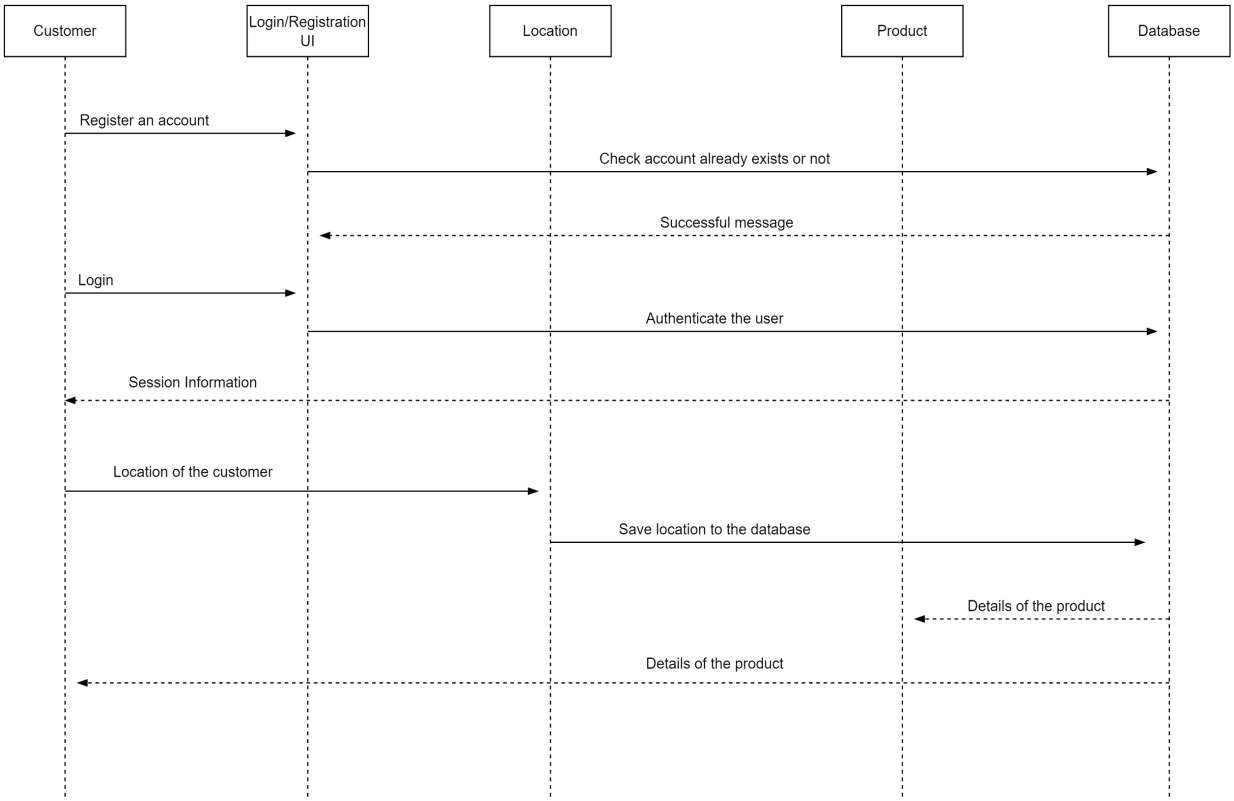

Shop Aggregation System
Registration & Login
Cart Management
Product Management
User Management
Shop Management
Customer
Shop
Admin

4.2.2 Sequence Diagram

A sequence diagram or system sequence diagram (SSD) shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Purpose of a Sequence Diagram:

- To model high-level interaction among active objects within a system.
- To model interaction among objects inside a collaboration realizing a use case.
- It either models generic interactions or some certain instances of interaction.

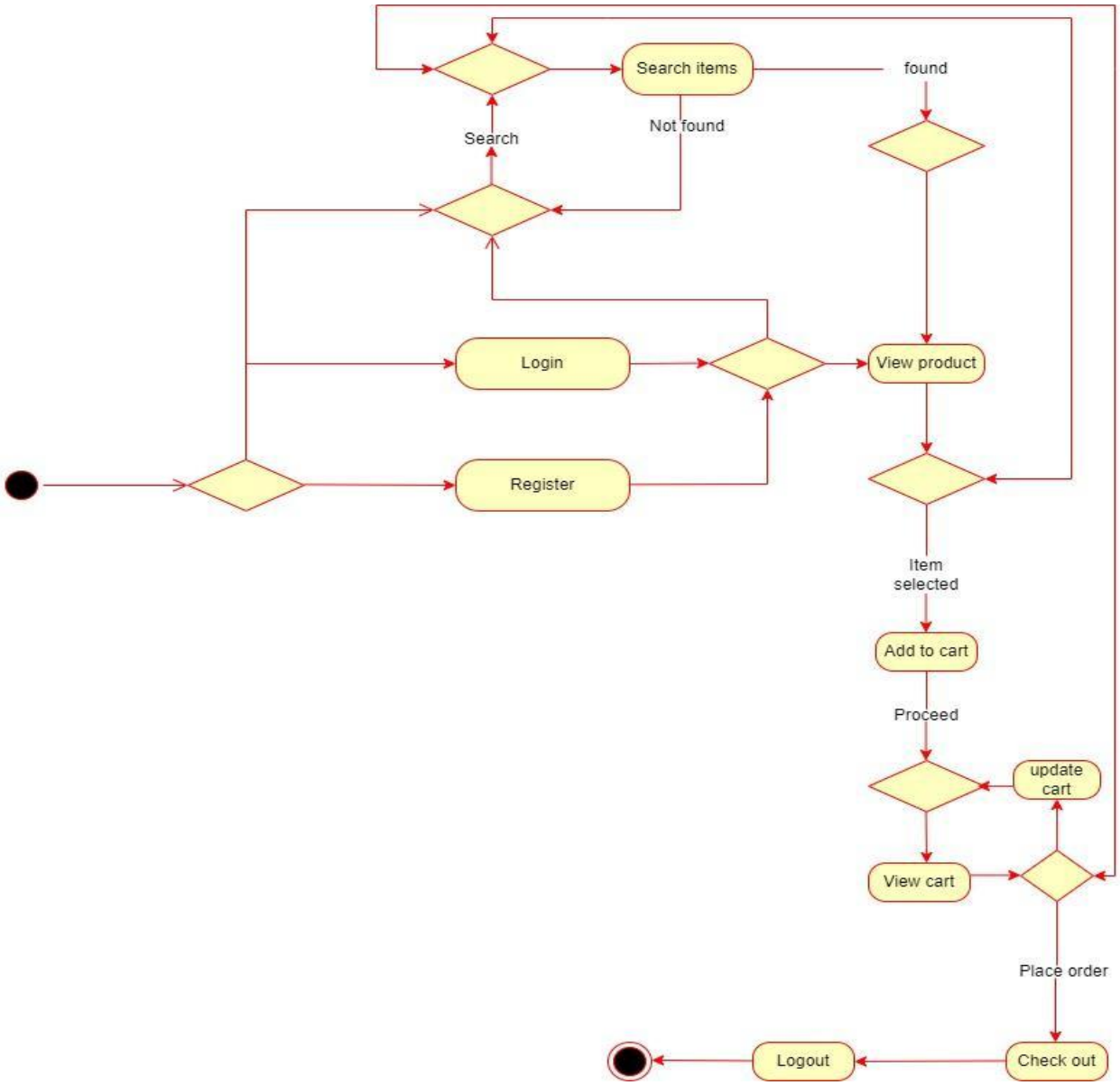


4.2.3 State Chart Diagram

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics. The Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events. Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of the Statechart diagram is to model the lifetime of an object from creation to termination. Statechart diagrams are also used for the forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

The main purposes of using Statechart diagrams are:

- To model the dynamic aspect of a system.
- To model the lifetime of a reactive system.
- To describe different states of an object during its lifetime.
- Define a state machine to model the states of an object.

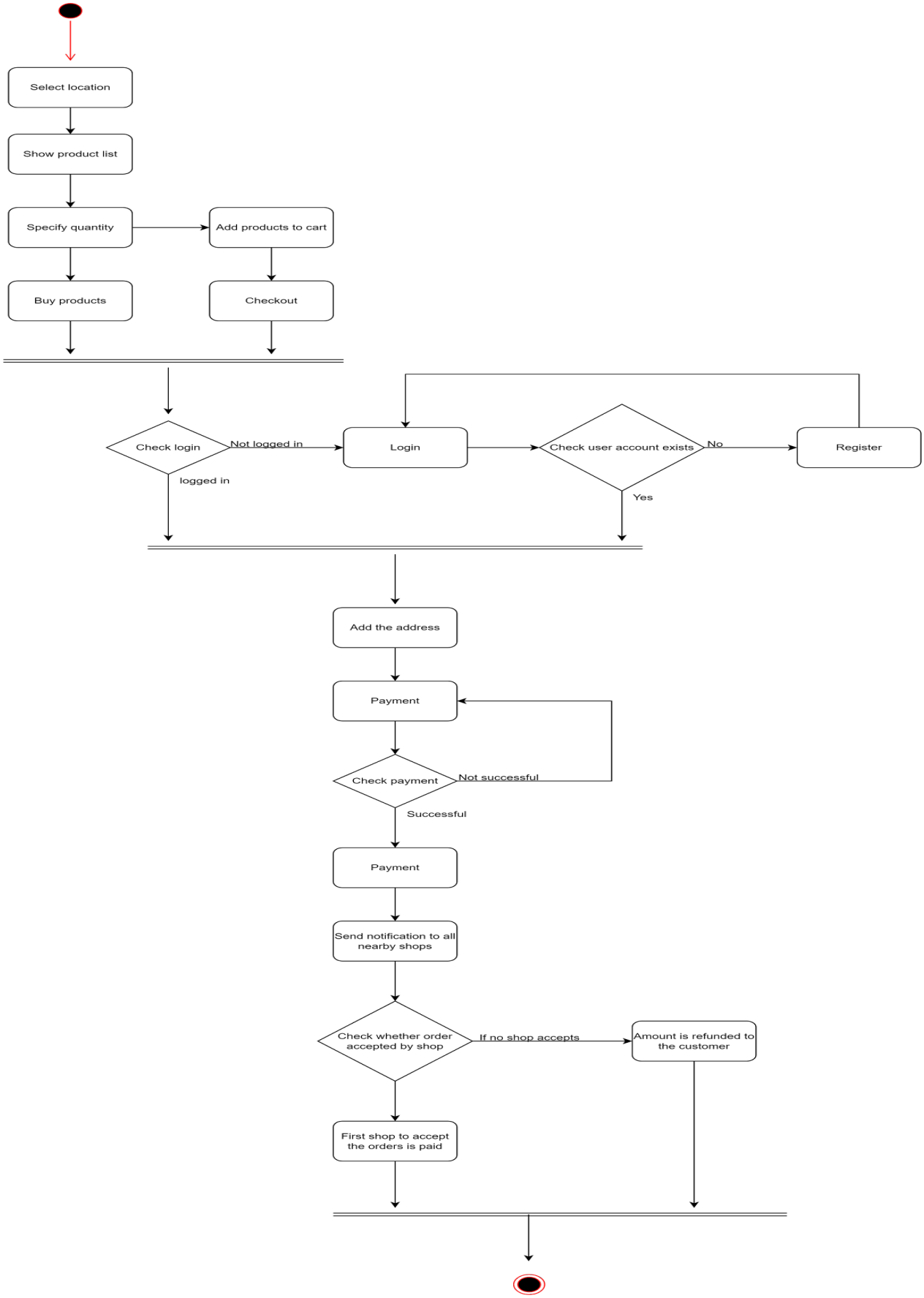

Search items
found
Not found
Search
Login
View product
Register
Item selected
Add to cart
Proceed
update cart
View cart
Place order
Logout
Check out

4.2.4 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores. Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types are:

- Ellipses represent actions
- Diamonds represent decisions.
- Bars represent the start (split) or end (join) of concurrent activities.
- Black circle represents the start (initial node) of the workflow.
- Encircled black circle represents the end (final node).

Arrows run from the start towards the end and represent the order in which activities happen. Activity diagrams can be regarded as a form of a structured flowchart combined with a traditional data flow diagram. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

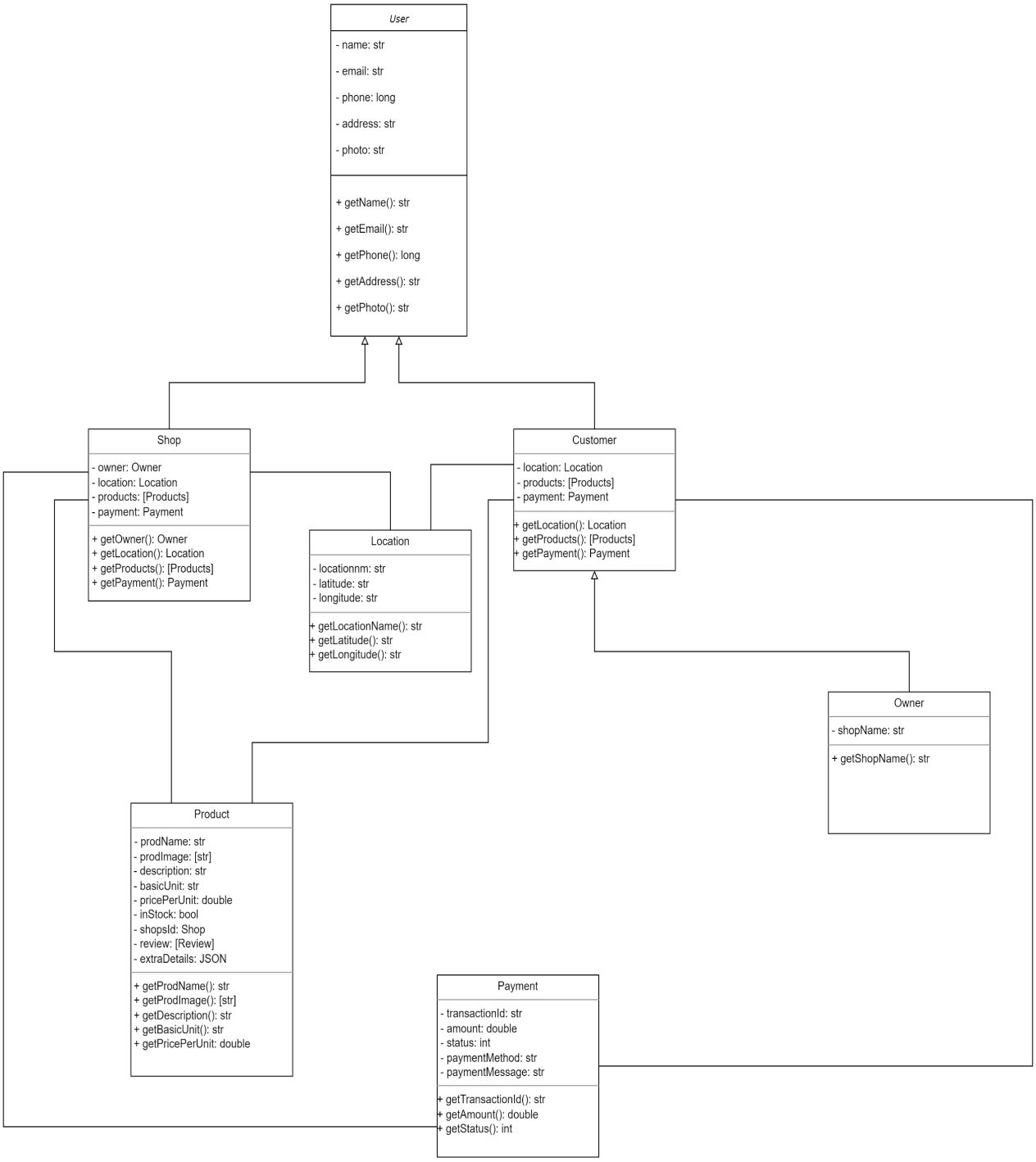

Select location
Show product list
Specify quantity
Add products to cart
Buy products
Checkout
Check login
Not logged in
logged in
Login
Check user account exists
No
Yes
Register
Add the address
Payment
Check payment
Not successful
Successful
Payment
Send notification to all nearby shops
Check whether order accepted by shop
If no shop accepts
Amount is refunded to the customer
First shop to accept the orders is paid

4.2.5 Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code. It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

The purpose of class diagrams given below:

- It analyses and designs a static view of an application.
- It describes the major responsibilities of a system.
- It is a base for component and deployment diagrams.
- It incorporates forward and reverse engineering.


User
- name: str
- email: str
- phone: long
- address: str
- photo: str
+ getName(): str
+ getEmail(): str
+ getPhone(): long
+ getAddress(): str
+ getPhoto(): str
Shop
- owner: Owner
- location: Location
- products: [Products]
- payment: Payment
+ getOwner(): Owner
+ getLocation(): Location
+ getProducts(): [Products]
+ getPayment(): Payment
Customer
- location: Location
- products: [Products]
- payment: Payment
+ getLocation(): Location
+ getProducts(): [Products]
+ getPayment(): Payment
Location
- locationnm: str
- latitude: str
- longitude: str
+ getLocationName(): str
+ getLatitude(): str
+ getLongitude(): str
Owner
- shopName: str
+ getShopName(): str
Product
- prodName: str
- prodImage: [str]
- description: str
- basicUnit: str
- pricePerUnit: double
- inStock: bool
- shopsId: Shop
- review: [Review]
- extraDetails: JSON
+ getProdName(): str
+ getProdImage(): [str]
+ getDescription(): str
+ getBasicUnit(): str
+ getPricePerUnit: double
Payment
- transactionId: str
- amount: double
- status: int
- paymentMethod: str
- paymentMessage: str
+ getTransactionId(): str
+ getAmount(): double
+ getStatus(): int

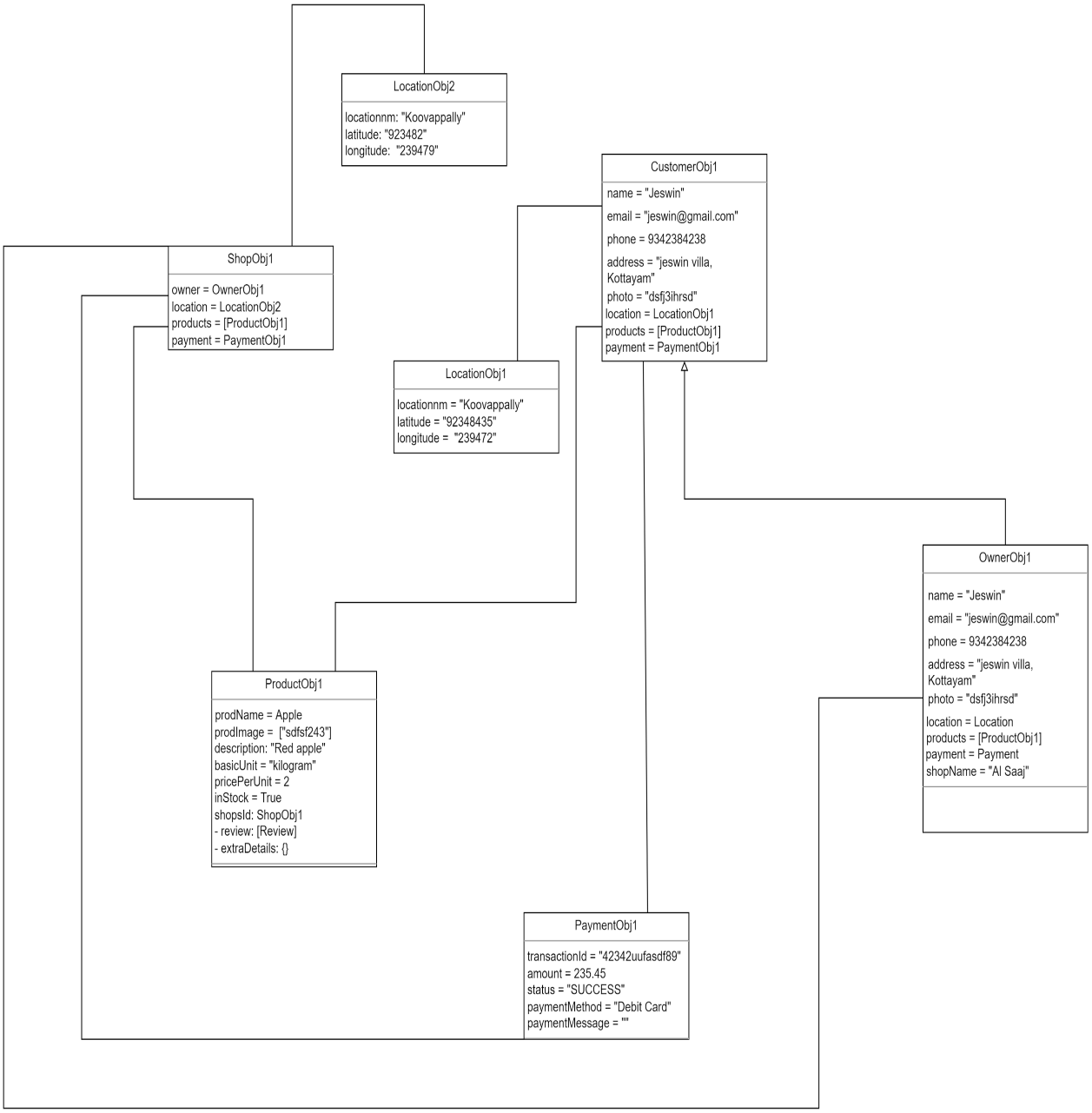
4.2.6 Object Diagram

Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.

Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system. The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view that comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

Purposes are enlisted below:

- It is used to perform forward and reverse engineering.
- It is used to understand object behavior and their relationships practically.
- It is used to get a static view of a system.
- It is used to represent an instance of a system.


LocationObj2
locationnm: "Koovappally"
latitude: "923482"
longitude: "239479"
CustomerObj1
name = "Jeswin"
email = "jeswin@gmail.com"
phone = 9342384238
address = "jeswin villa, Kottayam"
photo = "dsfj3ihrsd"
location = LocationObj1
products = [ProductObj1]
payment = PaymentObj1
ShopObj1
owner = OwnerObj1
location = LocationObj2
products = [ProductObj1]
payment = PaymentObj1
LocationObj1
locationnm = "Koovappally"
latitude = "92348435"
longitude = "239472"
OwnerObj1
name = "Jeswin"
email = "jeswin@gmail.com"
phone = 9342384238
address = "jeswin villa, Kottayam"
photo = "dsfj3ihrsd"
location = Location
products = [ProductObj1]
payment = Payment
shopName = "Al Saaj"
ProductObj1
prodName = Apple
prodImage = ["sdfsf243"]
description: "Red apple"
basicUnit = "kilogram"
pricePerUnit = 2
inStock = True
shopsId: ShopObj1
- review: [Review]
- extraDetails: {}
PaymentObj1
transactionId = "42342uufasdf89"
amount = 235.45
status = "SUCCESS"
paymentMethod = "Debit Card"
paymentMessage = ""

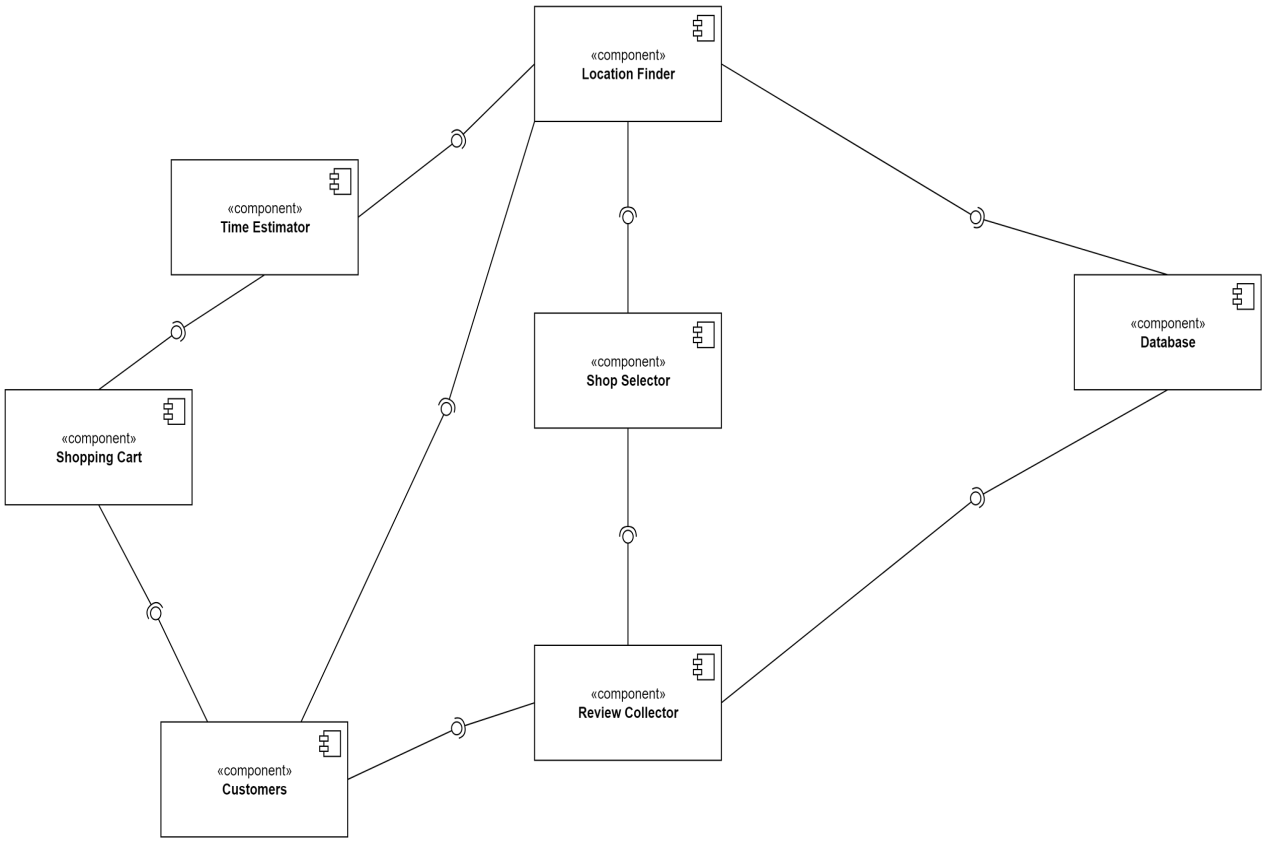
4.2.7 Component Diagram

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

The purpose of the component diagram are enlisted below:

- It envisions each component of a system.
- It constructs the executable by incorporating forward and reverse engineering.
- It depicts the relationships and organization of components.



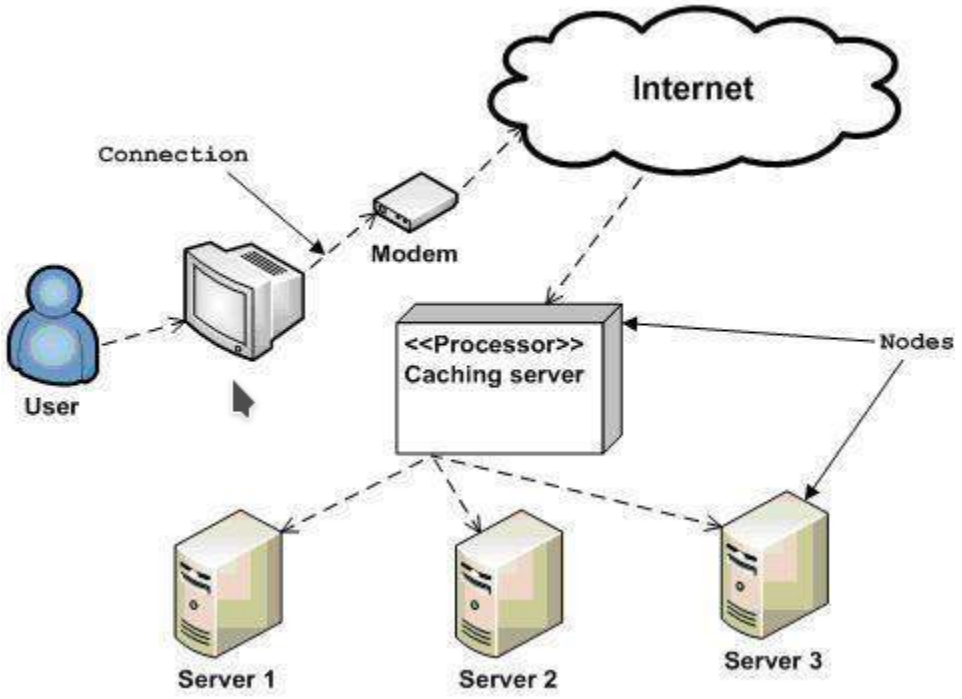
4.2.8 Deployment Diagram

The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.

It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths. The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.

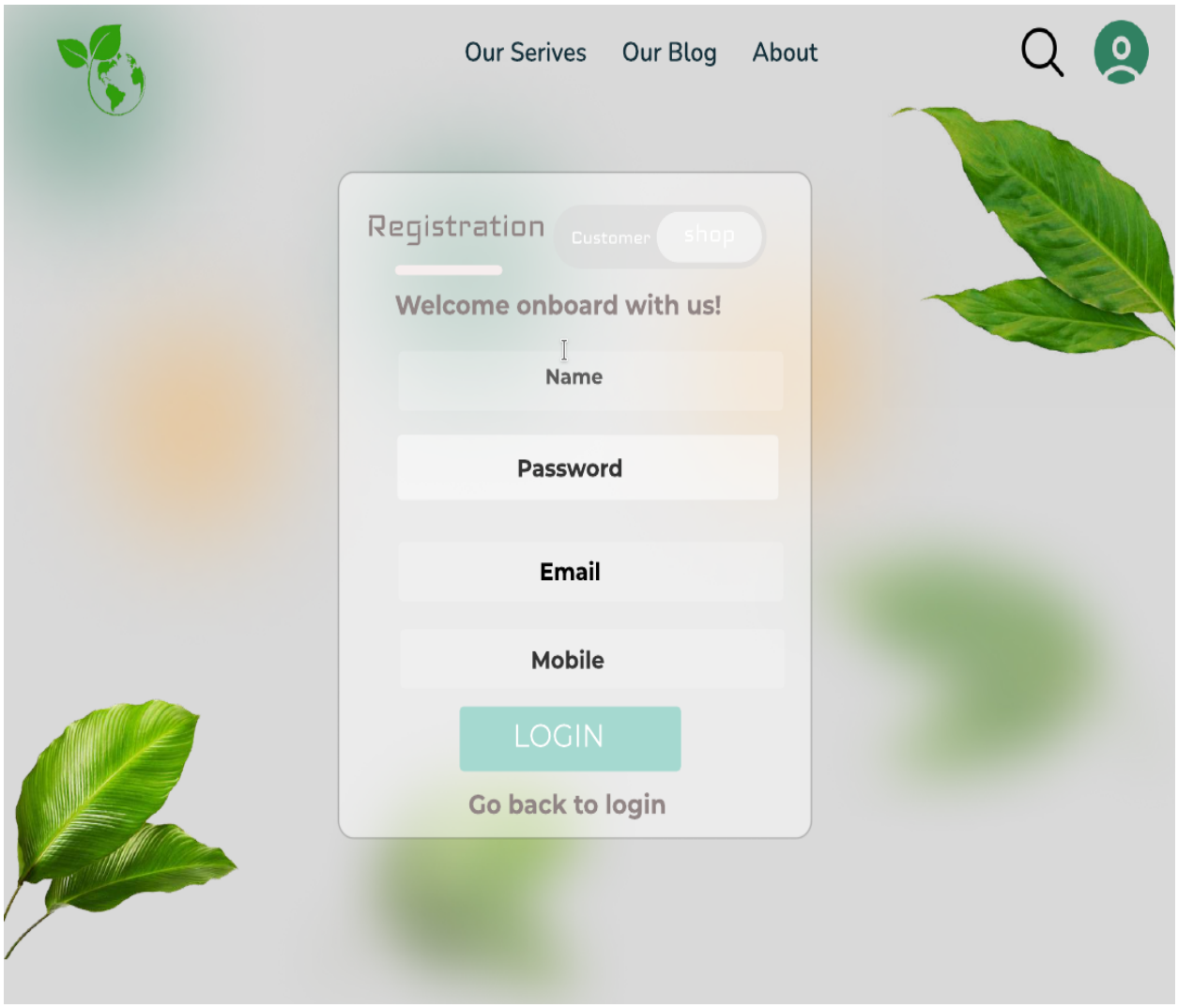
The purposes of deployment diagram enlisted below:

- To envision the hardware topology of the system.
- To represent the hardware components on which the software components are installed.
- To describe the processing of nodes at the runtime.

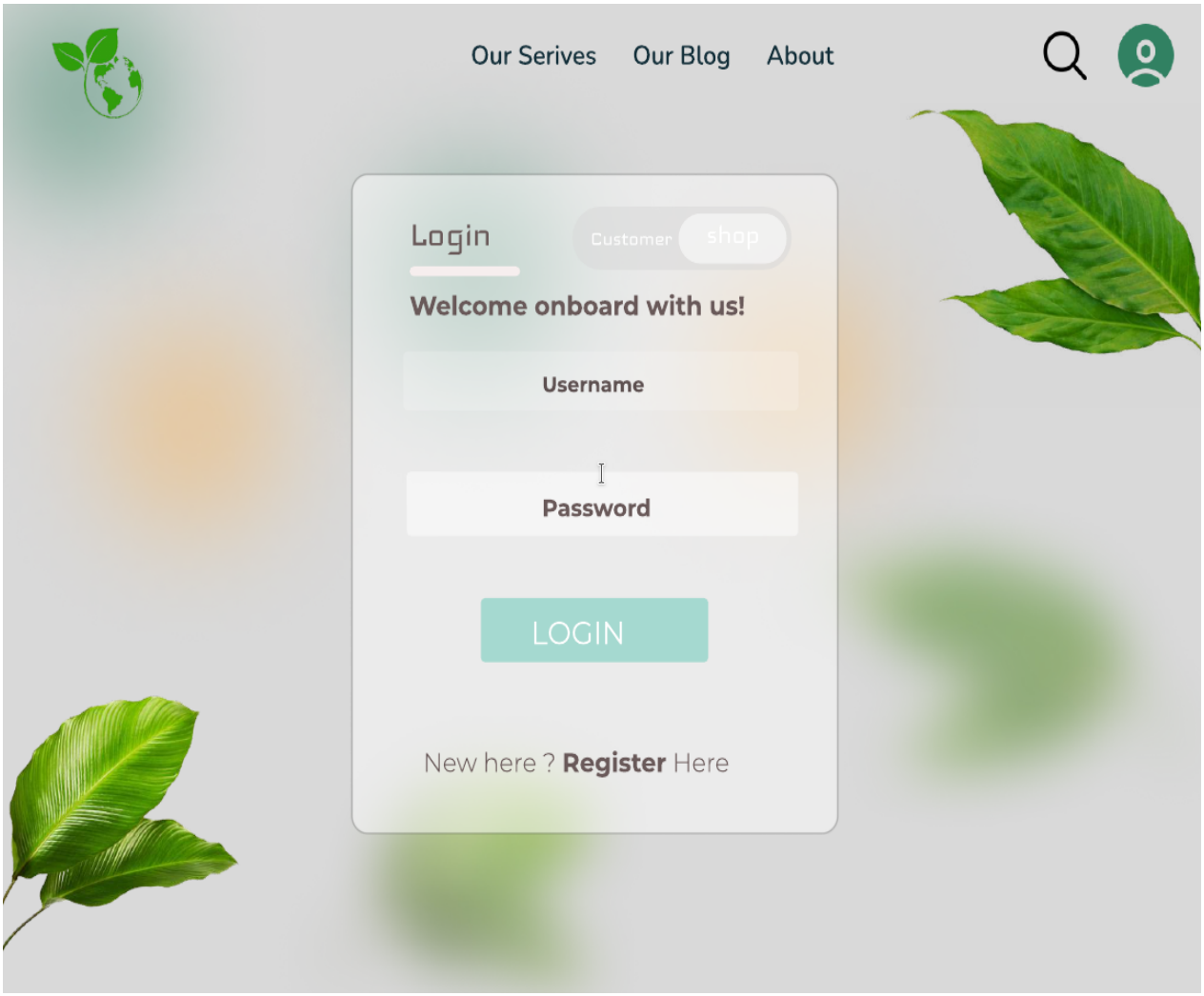

Internet
Connection
Modem
User
<<Processor>>
Caching server
Nodes
Server 1
Server 2
Server 3

4.3 USER INTERFACE DESIGN USING FIGMA

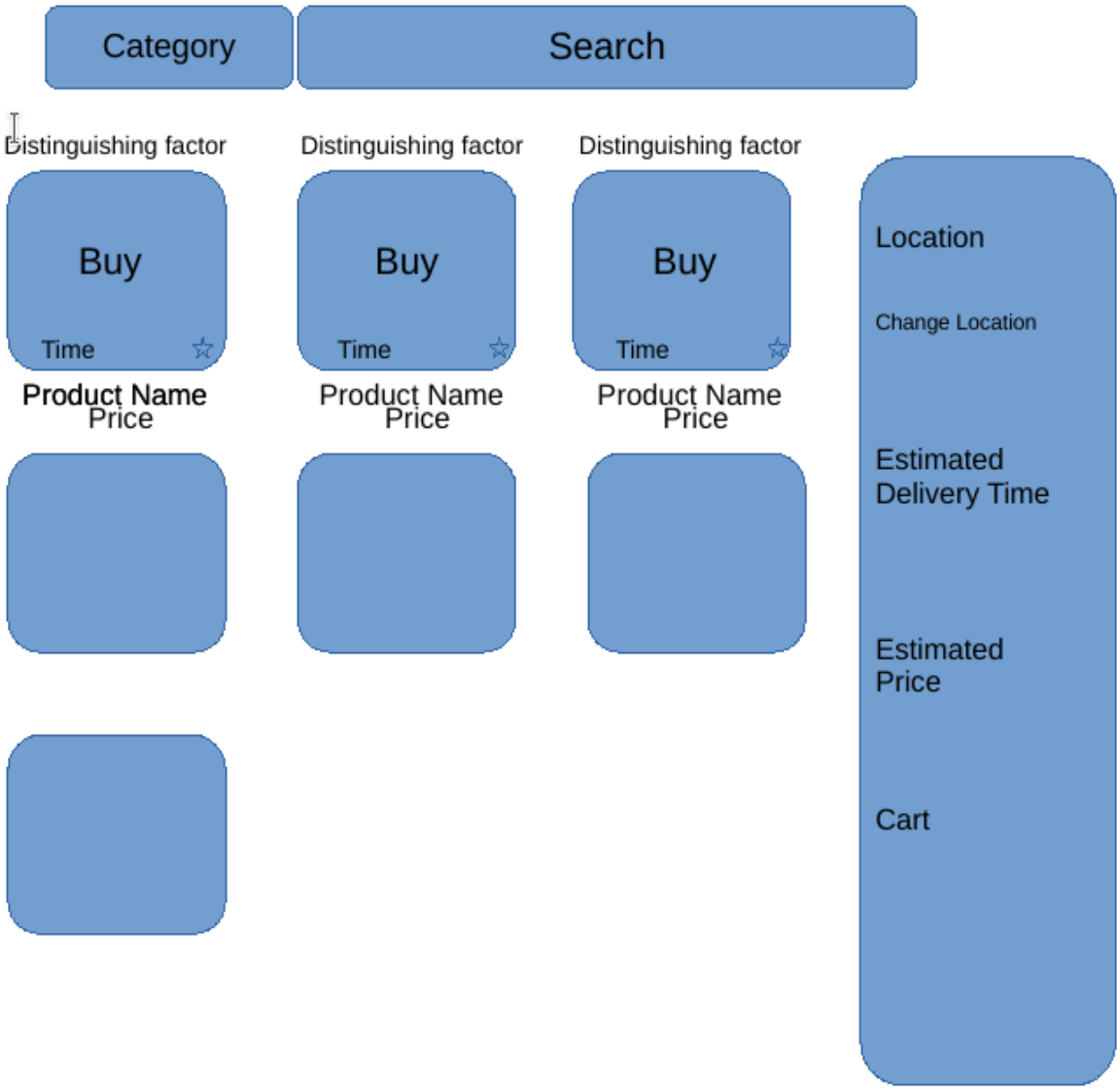
Form Name : User Registration



Form Name : User Login



Form Name : Product Display



4.4 DATABASE DESIGN

Database design can be generally defined as a collection of tasks or processes that enhance the designing, development, implementation, and maintenance of enterprise data management system. Designing a proper database reduces the maintenance cost thereby improving data consistency and the cost-effective measures are greatly influenced in terms of disk storage space. Therefore, there has to be a brilliant concept of designing a database. The designer should follow the constraints and decide how the elements correlate and what kind of data must be stored.

The main objectives behind database designing are to produce physical and logical design models of the proposed database system. To elaborate this, the logical model is primarily concentrated on the requirements of data and the considerations must be made in terms of monolithic considerations and hence the stored physical data must be stored independent of the physical conditions. On the other hand, the physical database design model includes a translation of the logical design model of the database by keep control of physical media using hardware resources and software systems such as Database Management System (DBMS).

4.4.1 Relational Database Management System (RDBMS)

RDBMS stands for Relational DataBase Management Systems. It is basically a program that allows us to create, delete, and update a relational database. Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of rows and columns. The major DBMS like SQL, My-SQL, ORACLE are all based on the principles of relational DBMS. Relational DBMS owes its foundation to the fact that the values of each table are related to others. It has the capability to handle larger magnitudes of data and simulate queries easily.

Relational Database Management Systems maintains data integrity by simulating the following features:

- **Entity Integrity**: No two records of the database table can be completely duplicate.
- **Referential Integrity**: Only the rows of those tables can be deleted which are not used by other tables. Otherwise, it may lead to data inconsistency.
- **User-defined Integrity**: Rules defined by the users based on confidentiality and access.
- **Domain integrity**: The columns of the database tables are enclosed within some structured limits, based on default values, type of data or ranges.

4.4.2 Normalization

Normalization is the process of organizing the data in the database. Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies. Normalization divides the larger table into smaller and links them using relationships. The normal form is used to reduce redundancy from the database table. The normalization in the DBMS can be defined as a technique to design the schema of a database and this is done by modifying the existing schema which also reduces the redundancy and dependency of the data. So with Normalization, the unwanted duplication in data is removed along with the anomalies. In insert anomaly, the values such as null are not allowed to be inserted for a column. The common normal forms are:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)

1. First Normal Form

   The table or relation is said to be in First Normal Form if it does not contain any multi-valued or composite attributes. So the table or relation should contain only single-valued attributes for fulfilling the condition for First Normal Form.

2. Second Normal Form

   A relation or table to be in Second Normal Form should be in First Normal Form and it should not hold any partial dependency. So in Second Normal Form, the table should not contain any non-prime attribute depending upon the proper subset of any candidate key.

3. Third Normal Form

   A table is in Third Normal Form if it is in Second Normal Form and there should not be any transitive dependency for the non-prime attributes. So for every non-trivial functional dependency A->B, if any of the two conditions is true from the below, the relation is said to be in the Third Normal Form.

   - A is a super key.
   - B is a prime attribute where each element of B is part of any candidate key.

4. Boyce-Codd Normal Form

   For a table to be in Boyce-Codd Normal Form, it should be in Third Normal Form and for every functional dependency A->B, A is the super key in the table.

4.4.3 Sanitization

Data sanitization involves purposely, permanently deleting, or destroying data from a storage device, to ensure it cannot be recovered.

Ordinarily, when data is deleted from storage media, the media is not really erased and can be recovered by an attacker who gains access to the device. This raises serious concerns for security and data privacy. With sanitization, storage media is cleansed so there is no leftover data on the device, and no data can be recovered, even with advanced forensic tools.

4.4.4 Indexing

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns. The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.

The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.

4.5 COLLECTION DESIGN

1. authentication_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of authentication collection
2	fullName	String	To store full name of the user
3	email	String	To store email of the user
4	password	String	To store password of the user
5	role	String	To store the type of role of the user
6	enableStatus	String	To store the status of the user

2. verification_token_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of verification token collection
2	token	String	To store the token used for verification
3	expirationTime	Date	To store the expiration time of the token
4	user	Sub Document	To store the details of the user to which the token was issued

3. customer_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of customer collection
2	fullName	String	To store full name of the customer
3	email	String	To store email of the customer
4	role	String	To store the role of the customer
5	mobile	String	To store mobile number of the customer
6	imageUrl	String	To store the url of the image of customer
7	enableStatus	String	To store the status of the customer
8	houseName	String	To store name of the house of the customer
9	streetName	String	To store name of the street of the customer
10	cityName	String	To store name of the city of the customer
11	pincode	String	To store pincode of the location of the customer

4. shop_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of shop collection
2	shopName	String	To store the name of the shop
3	email	String	To store the email of the shop
4	ownerFullName	String	To store full name of the owner of the shop
5	role	String	To store the role of the shop
6	mobile	String	To store the mobile number of the shop
7	imageUrl	String	To store the url of the image of shop
8	enableStatus	String	To store the status of the shop
9	verifyStatus	String	To store the verification status of the shop
10	documentUrl	String	To store the url of the document image of the shop
11	streetName	String	To store the name of street of shop
12	cityName	String	To store the name of city of shop
13	pincode	String	To store pincode of the location of the shop
14	productId	String[]	To store the list of Ids of products which the shop sells

5. product_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of product collection
2	shops	Sub Document	To store the list of shops that sell this product along with corresponding stock
3	pincode	String[]	To store the list of pincodes from which the product is available
4	prodName	String	To store the name of the product
5	category	String	To store the category of the product
6	imageUrl	String	To store url of the image of the product
7	mainUnit	String	To store the main unit in which the product is sold
8	saleUnit	String	To store the sale unit in which the product is sold
9	weight	double	To store the weight of the product
10	price	double	To store the price of the product
11	increment	int	To store the value in which the product is incremented
12	enabled	boolean	To store the status of the product

6. order_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of order collection
2	custId	String	To store the Id of the customer
3	products	Object[]	To store list of Ids of products along with the corresponding quantity and price
4	totalAmt	double	To store the total amount to be paid by the customer
5	time	long	To store the time in which the order was created
6	status	String	To store the status of the order
7	txnToken	String	To store the transaction token received from the payment gateway
8	addressId	String	To store the Id of the address to which the products are to be delivered

7. category_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of category collection
2	category	String	To store name of the category

8. address_collection

Primary key: _id

No	Field Key	Type	Description
1	_id	ObjectId	Primary key of address collection
2	fullName	String	To store full name of the customer
3	email	String	To store email of the customer
4	mobile	String	To store mobile number of the customer
5	houseName	String	To store name of house of the customer
6	place	String	To store the name of place of the customer
7	pincode	String	To store the pincode of the customer

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Some prefer saying Software testing definition as a White Box and Black Box Testing. In simple terms, Software Testing means the Verification of Application Under Test (AUT). This Software Testing course introduces testing software to the audience and justifies the importance of software testing. Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Benefits of using software testing:

- **Cost-Effective**: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security**: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
- **Product quality**: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction**: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

5.2 TEST PLAN

A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables.

The test plan is a base of every software's testing. It is the most crucial activity which ensures the availability of all the lists of planned activities in an appropriate sequence.

The test plan is a template for conducting software testing activities as a defined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager(20%), and by the test engineer(20%).

Types of test plan are as follows:

- **Master Test Plan**: Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.
- **Phase Test Plan:** A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.
- **Specific Test Plans:** Specific test plan designed for major types of testing like security testing, load testing, performance testing, etc. In other words, a specific test plan designed for non-functional testing.

5.2.1 Unit Testing

Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end. Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document. In order to execute Unit Tests, developers write a section of code to test a specific function in software application.

5.2.2 Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules. Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as integration testing.

5.2.3 Validation Testing or System Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

Validation Testing, carried out by QA professionals, is to determine if the system complies with the requirements and performs functions for which it is intended and meets the organization’s goals and user needs. This kind of testing is very important, as well as verification testing. Validation is done at the end of the development process and takes place after verification is completed.

Thus, to ensure customer satisfaction, developers apply validation testing. Its goal is to validate and be confident about the product or system and that it fulfils the requirements given by the customer. The acceptance of the software from the end customer is also its part.

When software is tested, the motive is to check the quality regarding the found defects and bugs. When defects and bugs are detected, developers fix them. After that, the software is checked again to make sure no bugs are left. In that way, the software product’s quality scales up.

5.2.4 Output Testing or User Acceptance Testing

User acceptance testing, also called end-user, user acceptability testing, or beta testing, is the process of testing software by the clients or users to see if the product is acceptable for release or not. The testers are familiar with the software's business requirements, so they can adequately gauge the product's readiness.

The software industry has its form of test audience screenings, and it's called user acceptance testing, or UAT for short. However, UAT is a lot more involved than showing a test audience a movie. That's why this article will introduce you to the world of user acceptance testing. We will answer questions like "What is user acceptance testing," "What are the prerequisites of user acceptance testing?" and "How do we make user acceptance testing more effective?" We will also touch upon the challenges of UATs, and how they differ from system testing.

User acceptance testing, also called end-user, user acceptability testing, or beta testing, is the process of testing software by the clients or users to see if the product is acceptable for release or not. The testers are familiar with the software's business requirements, so they can adequately gauge the product's readiness.

UAT is considered the last phase of the software testing process, conducted after the functional, system, and regression tests are complete. It's the final test run before the product goes live or before the client accepts delivery. Consider the UAT to be the bow sitting atop the testing package; it's not conducted until all other testing is done. Alpha, beta, and UAT all fall under the category of acceptance testing.

5.2.5 Automation Testing

Automation Testing, often known as Test Automation, is a software testing approach that involves the execution of a test case collection using particular automated testing software tools. On the other hand, manual testing is carried out by a person sitting in front of a computer, methodically carrying out the test processes.

In addition to entering test data into the System Under Test, the automated testing software may analyze predicted and actual outcomes and provide complete test reports. Software Test Automation necessitates significant financial and human resources. Continuous implementation of the same test suite will be required in subsequent development cycles. This test suite may be recorded and replayed as needed using a test automation tool. There is no need for human interaction after the test suite has been automated.

5.2.6 Selenium Testing

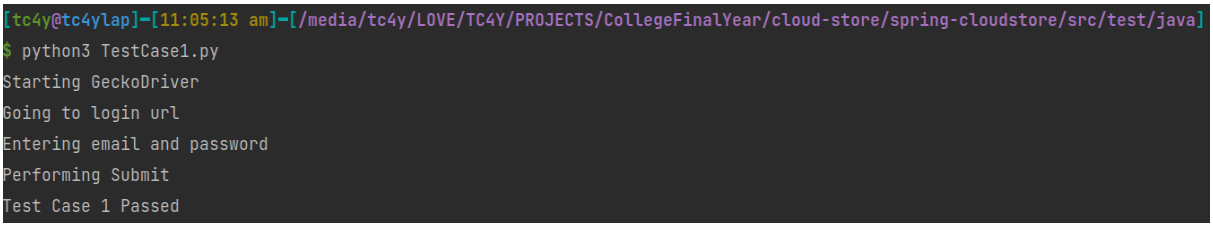
A curated set of various software tools, each striving with a distinct approach of assisting test automation, is known as Selenium Testing. This comprehensive suite of tools showcases the capabilities to satisfy the needs of testing of all types of web applications. Selenium Testing perfectly resembles flexibility. It plays a pivotal role in comparing expected test results against the actual behavior of an application. It allows various options in order to locate UI elements. In short, Selenium Testing is an important support system for the execution of tests on multiple browser platforms.

Test Case 1

Code

```
import selenium
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium import webdriver
expectedUrl = "http://localhost:3000"
print("Starting GeckoDriver")
driver = webdriver.Firefox()
print("Going to login url")
driver.get("http://localhost:3000/login")
print("Entering email and password")
driver.find_element(By.ID, "emailLogin")
.send_keys("astg4527customer1@gmail.com")
driver.find_element(By.ID,
"passLogin").send_keys("QWERTY@123qwerty")
print("Performing Submit")
driver.find_element(By.ID, "submitLogin").click()
driver.implicitly_wait(10)
currentUrl = driver.current_url
if expectedUrl == currentUrl:
 print("Test Case 1 Passed")
else:
 print("Test Case 1 Failed")
```

Screenshot



Test report

Test Case 1

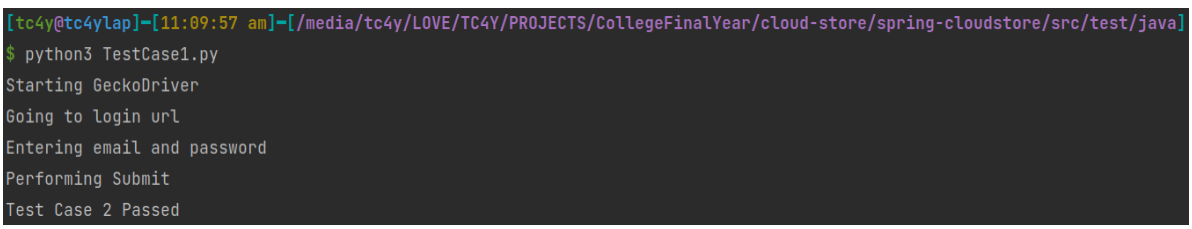
Project Name: Cloud Store					
Login Test Case					
Test Case ID: Test_1			**Test Designed By:** Ashish Sam T George		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 31-10-2022		
Module Name: Login			**Test Executed By :** Ms. Navyamol K.T		
Test Title : Login Test with valid email and password			**Test Execution Date:** 31-10-2022		
Description: Testing the login module					
Pre-Condition : User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status**
1	Navigation to Login Page		Login Page should be displayed	Login Page displayed	Pass
2	Provide valid username	Username: astg4527customer1@gmail.com	User should be able to login	User logged in and navigated to homepage	Pass
3	Provide valid password	Password: QWERTY@123qwerty			
4	Click on Submit button				
Post-Condition: User is validated with database and successfully logged in to the account					

Test Case 2

Code

```
import selenium
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium import webdriver
expectedUrl = "http://localhost:3000"
print("Starting GeckoDriver")
driver = webdriver.Firefox()
print("Going to login url")
driver.get("http://localhost:3000/login")
print("Entering email and password")
driver.find_element(By.ID, "emailLogin")
.send_keys("astg4527random@gmail.com")
driver.find_element(By.ID,
"passLogin").send_keys("QWERTY@98usdf3")
print("Performing Submit")
driver.find_element(By.ID, "submitLogin").click()
driver.implicitly_wait(10)
currentUrl = driver.current_url
if expectedUrl != currentUrl:
 print("Test Case 2 Passed")
else:
 print("Test Case 2 Failed")
```

Screenshot



Test report

Test Case 2

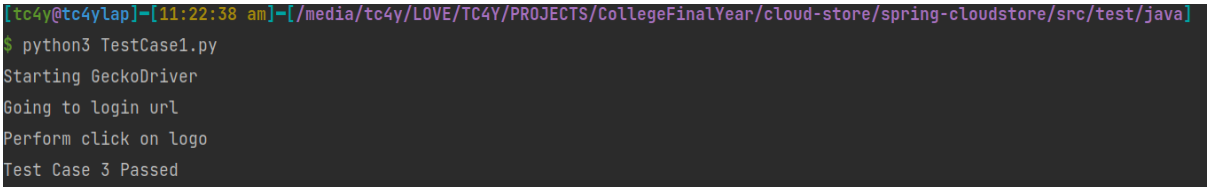
Project Name: Cloud Store					
Login Test Case					
Test Case ID: Test 2			**Test Designed By:** Ashish Sam T George		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 31-10-2022		
Module Name: Login			**Test Executed By :** Ms. Navyamol K.T		
Test Title : Login Test with invalid email and password			**Test Execution Date:** 31-10-2022		
Description: Testing the login module					
Pre-Condition : User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status**
1	Navigation to Login Page		Login Page should be displayed	Login Page displayed	Pass
2	Provide invalid username	Username: astgrandom@gmail.com	User should not be able to login	User not logged in and error message is shown	Pass
3	Provide invalid password	Password: QWERTY@sadfshf3uhh			
4	Click on Submit button				
Post-Condition: Credentials are validated with database and not logged in to the account					

Test Case 3

Code

```
import selenium
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium import webdriver

expectedUrl = "http://localhost:3000"

print("Starting GeckoDriver")
driver = webdriver.Firefox()
print("Going to login url")
driver.get("http://localhost:3000/login")
print("Perform click on logo")
driver.find_element(By.ID, "logoText") .click()
driver.implicitly_wait(10)

currentUrl = driver.current_url

if expectedUrl == currentUrl:
  print("Test Case 3 Passed")
else:
  print("Test Case 3 Failed")
```

Screenshot



Test report

Test Case 3

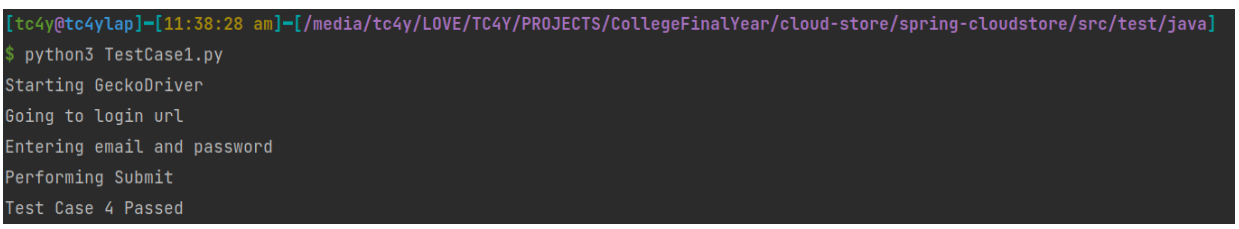
Project Name: Cloud Store					
Login Test Case					
Test Case ID: Test_3			**Test Designed By:** Ashish Sam T George		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 31-10-2022		
Module Name: LogoText Navigation			**Test Executed By :** Ms. Navyamol K.T		
Test Title : LogoText navigation when clicked			**Test Execution Date:** 31-10-2022		
Description: Testing the navigation when logo text is clicked					
Pre-Condition : Page is completely loaded					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status**
1	Navigation to Login Page		Login Page should be displayed	Login Page displayed	Pass
2	Click on logo text (CloudStore)		User should be navigated to the homepage showing products	User is navigated to the homepage showing products	Pass
Post-Condition: Credentials are validated with database and user is not logged in to the account					

Test Case 4

Code

```
import time
import selenium
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium import webdriver
expectedUrl = "http://localhost:3000"
print("Starting GeckoDriver")
driver = webdriver.Firefox()
print("Going to login url")
driver.get("http://localhost:3000/login")
print("Entering email and password")
driver.find_element(By.ID, "emailLogin")
.send_keys("astg4527customer1@gmail.com")
driver.find_element(By.ID,
"passLogin").send_keys("QWERTY@123qwerty")
print("Performing Submit")
driver.find_element(By.ID, "submitLogin").click()
time.sleep(5)
driver.find_element(By.ID, "logoutButton").click()
time.sleep(5)
currentUrl = driver.current_url
if expectedUrl == currentUrl:
 print("Test Case 4 Passed")
else:
 print("Test Case 4 Failed")
```

Screenshot



Test report

Test Case 4

Project Name: Cloud Store					
Login Test Case					
Test Case ID: Test_4			**Test Designed By:** Ashish Sam T George		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 31-10-2022		
Module Name: Logout			**Test Executed By :** Ms. Navyamol K.T		
Test Title : Login Test with logged in user account			**Test Execution Date:** 31-10-2022		
Description: Testing the logout by clicking on the Logout button					
Pre-Condition : User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status**
1	Navigation to Login Page		Login Page should be displayed	Login Page displayed	Pass
2	Provide valid username	Username: astg4527customer1@gmail.com	User should be able to login	User logged in and navigated to homepage	Pass
3	Provide valid password	Password: QWERTY@123qwerty			
4	Click on Submit button				
5	Click on Logout button		User should be logged out of the account	User is logged out of the account	Pass
Post-Condition: User is successfully logged out of the account					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the execution or practice of a plan, a method or any design, idea, model, specification, standard or policy for doing something. As such, implementation is the action that must follow any preliminary thinking for something to actually happen. Implementation is often used in the tech world to describe the interactions of elements in programming languages. To implement a project means to carry out activities proposed in the application form with the aim to achieve project objectives and deliver results and outputs. Implementation is the process that turns strategies and plans into actions in order to accomplish strategic objectives and goals. Implementing your strategic plan is as important, or even more important, than your strategy.

Project Implementation is the last stage before the analysis of the outcome where all such actions take place. This stage is the most challenging one in terms of implementation, details, and high team coordination requirements. The Project Implementation phase has two essential functions: execution of the work and proper delivery. The resources used in the implementation have to be accurate.

Based on the implementation, the project's fate is decided. The success and effectiveness can only be known after proper monitoring and feedback. This phase of delivering provides the client with an outlook of the project. The key to a successful implementation is proper structuring and coordination at the managerial level.

6.2 IMPLEMENTATION PROCEDURES

For the successful implementation of a project, you need to:

- Execute the planned details into a full-proof action plan
- Document every little detail in this stage, from decisions to results at every step
- Have an efficient line of communication in place across the hierarchy
- Take quick decisions if the need for a change of plan felt instantly.
- Form a consensus about the changes and implement immediately

There are three broad areas of concern in this stage of the project implementation. The first one is Project Scope Management, Project Time Management, and finally, Project Cost Management. These areas help resolve constraints in terms of the scope, budget, and time required for its implementation and success. It helps arrive at the uniformity of decisions regarding the same.

6.2.1 User Training

User Training is a term business executives hear quite often when they are in the process of implementing a new software system, however, more often than not, many choose to opt-out of receiving this training for their staff. As it is an added expense and many people nowadays are tech-savvy, business managers decide their employees can learn how to use the new system on their own. Unfortunately, this way of thinking can cause more problems during and after the implementation process.

Implementing a new system at a company is a big change for many employees. Many times a new or upgraded solution will include new sections, tools, buttons and improved processes that will allow businesses to be more productive in the long run and reduce human errors. However, because of all these changes and new processes, it is quite unlikely all users (employees) will be on the same page when learning the new system on

their own, which can prevent the company from seeing all the efficiencies the IT provider promised with the upgraded system. This is usually the moment where executives blame the software system and /or provider for not performing as originally planned. What many seem to forget is that software solutions can only show all of its benefits if used properly. Therefore, user training can truly help with the implementation of a new system at a company and ensure maximum efficiency right from the start. The IT provider can train users on their software system in a few hours and can always provide support after implementation if employees have any questions at a later date.

6.2.2 System Maintenance

Maintenance means restoring something to its original conditions. Enhancement means adding, modifying the code to support the changes in the user specification. System maintenance conforms the system to its original requirements and enhancement adds to system capability by incorporating new requirements.

Thus, maintenance changes the existing system, enhancement adds features to the existing system, and development replaces the existing system. It is an important part of system development that includes the activities which corrects errors in system design and implementation, updates the documents, and tests the data.

System maintenance can be classified into three types −

Corrective Maintenance − Enables user to carry out the repairing and correcting leftover problems.

Adaptive Maintenance − Enables user to replace the functions of the programs.

Perfective Maintenance − Enables user to modify or enhance the programs according to the users' requirements and changing needs

6.2.3 Training on the Application Software

Software training is important for a number of reasons, including that it can increase a staff's confidence and productivity rates, facilitate collaboration efforts, reduce errors, and even allow employees to perform duties outside of their job descriptions if needed. While some might assume that software training refers to standalone applications, the phrase can also encompass social media platforms and tools. Learning how to work with social media can support marketing efforts by reaching more prospects and increasing profits. In addition, employees who can work with content management systems or blogs can regularly offer valuable information to the public, and this can help establish your business as an authority within your industry. The result can be a raised profile and higher credibility.

There can be some unexpected software training benefits as well. For instance, training your employees on how to competently use software can indirectly act as an extra layer of cybersecurity for your business. If your employees understand a program's basic features and how it works with the Internet, they may learn to spot potential security breaches that could affect your business's systems. Proper software training can help protect your consumers' data and corporate networks. In a world where big data is one of the most in-demand commodities, this type of training can pay dividends.

Software training can also help your business stay competitive. New software is released every day. Certain pieces of software or applications can become instantly relevant or outdated, necessitating new training. For example, the shift to remote work has made video conferencing software much more popular, allowing a nationwide workforce to easily collaborate online. Taking full advantage of video conferencing software's advanced features can take some training, but it can be worthwhile if the work produced doesn't disrupt an established workflow and meets or exceeds an employer's expectations.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The project entitled '**Cloud Store**' have been developed to help people buy products with ease from nearby local shops. Cloud Store is a web application that can be used to buy products which are available near the customer. Customers can easily buy products without the need to visit each shop. Since the products are available nearby, it will be delivered to the customer without significant delay. Using Vue as the front end and Spring as back end, along with MongoDB as the database, the system was developed and tested with all possible samples of data. The performance of the system is provided to be efficient and can meet all the requirements of the user. It provides a fast and efficient web application that can be used to buy various kinds of nearby products with ease.

7.2 FUTURE SCOPE

- Shop Selection module can be integrated with the web application. It will analyse the location of the customer and the reviews on the shop in order to select the most suitable shop for that customer. The product will be selected from that shop. The customer always has the option to choose the shop according to their wish.
- Review module can be integrated with the web application. It can directly and indirectly gather reviews from the customer in various ways.
- Cryptocurrency payment module can be integrated in the web application which will enable the web application to accept cryptocurrency as payments. It will provide a bridge between decentralized and traditional commerce.
- The web application can be migrated to blockchain technology so that it can work as a decentralized application.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- Saurabh Pal, "*System Analysis and Design*", 2017

- R.L. Glass, I. Vessey, V. Ramesh, "*Research in software engineering: an analysis of the literature*", 2002

- Engström, Emelie & Storey, Margaret-Anne & Runeson, Per & Höst, Martin & Baldassarre, Maria, "*A review of software engineering research from a design science perspective*", 2019

WEBSITES:

- https://www.javatpoint.com/java-tutorial
- https://www.w3schools.com/Js/default.asp
- https://docs.spring.io/spring-security/reference/getting-spring-security.html
- https://vuejs.org/guide/introduction.html

CHAPTER 9

APPENDIX

9.1 Sample Code

Signup.vue

```
<template>
  <div class="page-sign-up">
  <div class="columns">
  <div class="column is-4 is-offset-4">
        <h1 class="title">Sign up</h1>

        <form @submit.prevent="submitForm">
        <div class="field">
        <label class="is-size-5">Full Name</label>
        <div class="control">
              <input type="text" class="input" v-model="state.fullName" />
              <span v-if="v$.fullName.$error" class="has-text-danger">
              {{ v$.fullName.$errors[0].$message }}
              </span>
        </div>
        </div>

        <div class="field">
        <label class="is-size-5">Email</label>
        <div class="control">
              <input type="text" class="input" v-model="state.email" />
              <span v-if="v$.email.$error" class="has-text-danger">
              {{ v$.email.$errors[0].$message }}
              </span>
        </div>
        </div>

        <div class="field">
        <label class="is-size-5">Password</label>
        <div class="control">
              <input type="password" class="input" v-model="state.password"
/>
              <span v-if="v$.password.$error" class="has-text-danger">
              {{ v$.password.$errors[0].$message }}
              </span>
        </div>
        </div>

        <div class="field">
```

```
<label class="is-size-5">Confirm password</label>
        <div class="control">
              <input type="password" class="input" v-model="state.password2"
/>
              <span v-if="v$.password2.$error" class="has-text-danger">
              {{ v$.password2.$errors[0].$message }}
              </span>
        </div>
        </div>

        <div class="field">
        <label class="is-size-5">Role</label>
        <div class="control">
              <div class="select">
              <select class="is-hovered" v-model="state.role">
              <option selected value="CUSTOMER">Customer</option>
              <option value="SHOP">Shop</option>
              </select>
              <span v-if="v$.role.$error" class="has-text-danger">
              {{ v$.role.$errors[0].$message }}
              </span>
              </div>
        </div>
        </div>

        <!-- <div class="notification is-danger" v-if="errors.length">
        <p v-for="error in errors" v-bind:key="error">{{ error }}</p>
        </div> -->

        <div class="field mt-6">
        <div class="control">
              <button class="button is-dark">Sign up</button>
        </div>
        </div>

        <hr />

        Already have an Account? <router-link to="/login">Log
in</router-link>
        </form>
  </div>
  </div>
  </div>
</template>
```

```
<script>
import axios from "axios";
import { toast } from "bulma-toast";

import useVuelidate from "@vuelidate/core";
import {
  required,
  email,
  sameAs,
  minLength,
  maxLength,
  helpers,

} from "@vuelidate/validators";
import { reactive, computed } from "vue";

export default {
  name: "SignUp",
  setup() {
  const state = reactive({
  fullName: "",
  email: "",
  password: "",
  password2: "",
  role: "",
  });

  const rules = computed(() => {
  return {
        fullName: { required },
        email: { required, email },
        password: {
        required,
        minLength: minLength(8),
        maxLength: maxLength(131),
        containsPasswordRequirement: helpers.withMessage(
        () =>
              `The password requires an uppercase, lowercase, number and
special character`,
        (value) =>

/(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#\$%\^&\*])/.test(value)
        ),
        },
```

```
password2: { required, sameAs: sameAs(state.password) },
        role: { required },
  };
  });

  const v$ = useVuelidate(rules, state);

  return {
  state,
  v$,
  };
  },
  mounted() {
  document.title = "Sign Up | CloudStore";
  },
  methods: {
  submitForm() {
  this.v$.$validate();
  if (!this.v$.$error) {
        const formData = {
        fullName: this.state.fullName,
        email: this.state.email,
        password: this.state.password,
        role: this.state.role,
        };

        this.$store.commit("setIsLoading", true);

        axios
        .post("/user/register", formData)
        .then((response) => {
        toast({
              message: "Account created, please log in!",
              type: "is-success",
              dismissible: true,
              pauseOnHover: true,
              duration: 2000,
              position: "bottom-right",
        });

        this.$router.push("/login");
        })
        .catch((error) => {
        toast({
```

```
message: "Account not created, Try again",
              type: "is-success",
              dismissible: true,
              pauseOnHover: true,
              duration: 2000,
              position: "bottom-right",
        });
        });
        this.$store.commit("setIsLoading", false);
  }
  },
  },
};
</script>
```

Login.vue

```
<template>
  <div class="page-log-in">
  <div class="columns">
  <div class="column is-4 is-offset-4">
        <h1 class="title">Log in</h1>

        <form @submit.prevent="submitForm">
        <div class="field">
        <label>Email</label>
        <div class="control">
              <input type="text" class="input" v-model="state.email"
id="emailLogin" />
              <span v-if="v$.email.$error" class="has-text-danger">
              {{ v$.email.$errors[0].$message }}
              </span>
        </div>
        </div>

        <div class="field">
        <label>Password</label>
        <div class="control">
              <input type="password" class="input" v-model="state.password"
id="passLogin" />
              <span v-if="v$.password.$error" class="has-text-danger">
              {{ v$.password.$errors[0].$message }}
```

```
</span>
        </div>
        </div>

        <!-- <div class="notification is-danger" v-if="errors.length">
        <p v-for="error in errors" v-bind:key="error">{{ error }}</p>
        </div> -->

        <div class="field">
        <div class="control">
              <button class="button is-dark" id="submitLogin">Log in</button>
        </div>
        </div>

        <hr />

        Don't have an Account? <router-link to="/signup">Sign
Up</router-link>
        </form>
  </div>
  </div>
  </div>
</template>

<script>
import axios from "axios";

import { toast } from "bulma-toast";

import useValidate from "@vuelidate/core";
import {
  required,
  email,
  sameAs,
  minLength,
  maxLength,
  helpers,
} from "@vuelidate/validators";
import { reactive, computed } from "vue";

export default {
  name: "LogIn",

  setup() {
```

```
const state = reactive({
  email: "",
  password: "",
  });

  const rules = computed(() => {
  return {
        email: { required, email },
        password: {
        required,
        minLength: minLength(8),
        maxLength: maxLength(131),
        containsPasswordRequirement: helpers.withMessage(
        () =>
              `The password requires an uppercase, lowercase, number and
special character`,
        (value) =>

/(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#\$%\^&\*])/.test(value)
        ),
        },
  };
  });

  const v$ = useValidate(rules, state);

  return {
  state,
  v$,
  };
  },

  mounted() {
  document.title = "Log In | CloudStore";
  },
  methods: {
  async submitForm() {
  this.v$.$validate();

  if (!this.v$.$error) {
        axios.defaults.headers.common["Authorization"] = "";

        localStorage.removeItem("token");
```

```
const formData = {
        email: this.state.email,
        password: this.state.password,
        };

        this.$store.commit("setIsLoading", true);

        await axios
        .post("/user/login", formData)
        .then((response) => {

        const token = response.data;

        this.$store.commit("setToken", token);

        axios.defaults.headers.common["Authorization"] = "Bearer " + token;

        localStorage.setItem("token", token);

        var base64Url = token.split(".")[1];
        var base64 = base64Url.replace(/-/g, "+").replace(/_/g, "/");
        var jsonPayload = decodeURIComponent(
              window
              .atob(base64)
              .split("")
              .map(function(c) {
              return "%" + ("00" + c.charCodeAt(0).toString(16)).slice(-2);
              })
              .join("")
        );


        let decoded = JSON.parse(jsonPayload);
        let decodedRole = decoded.authorities[0].authority;
        let decodedId = decoded.id

        this.$store.commit("setUserRole", decodedRole);
        this.$store.commit("setUserId", decodedId)

        const toPath = this.$route.query.to || "/";

        this.$router.push(toPath);
        })
        .catch((error) => {
```

```
toast({
              message: "Not Logged In, Try again",
              type: "is-success",
              dismissible: true,
              pauseOnHover: true,
              duration: 2000,
              position: "bottom-right",
        });
        });

        this.$store.commit("setIsLoading", false);
  }
  },
  },
};
</script>
```

ProductController.java

```
package com.cloudstore.service.admin;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.cloudstore.entity.ProductCategoryEntity;
import com.cloudstore.entity.ProductEntity;
import com.cloudstore.model.ProductModel;
import com.cloudstore.service.ShopServiceInterface;

@RestController
public class ProductController {

    @Autowired
    private ShopServiceInterface shopService;
```

```
 @CrossOrigin("http://localhost:3000")
    @GetMapping("/products")
    public List<ProductEntity> listProducts() {
          List<ProductEntity> products = shopService.findAllProducts();
          return products;
    }

    @CrossOrigin("http://localhost:3000")
    @GetMapping("/productsbyquery")
    public List<ProductEntity> listProductsByQuery(@RequestParam String
query){
          List<ProductEntity> products = shopService.similarProducts(query);
          return products;
    }

    @CrossOrigin("http://localhost:3000")
    @GetMapping("/productsbypin")
    public List<ProductEntity> listProductsByPincode(@RequestParam String
pincode){
          List<ProductEntity> products =
shopService.findAllProductsByPincode(pincode);
          return products;
    }

    @CrossOrigin("http://localhost:3000")
    @GetMapping("/productsbycategory")
    public List<ProductEntity> listProductsByCategory(@RequestParam String
category){
          List<ProductEntity> products =
shopService.findAllProductsByCategory(category);
          return products;
    }

    @CrossOrigin("http://localhost:3000")
    @GetMapping("/product")
    public ProductEntity listProducts(@RequestParam String prodId) {
          ProductEntity product = shopService.findProductById(prodId);
          return product;
    }


    @CrossOrigin("http://localhost:3000")
    @DeleteMapping("/product")
    public String deleteProduct(@RequestParam String prodId) {
```

```
shopService.deleteProduct(prodId);
          return "SUCCESSFULLY DELETED";
    }

    @CrossOrigin("http://localhost:3000")
    @PostMapping("/admin/product")
    public ProductEntity addProduct(@RequestBody ProductModel productModel) {
          ProductEntity product = shopService.addProduct(productModel);
          return product;
    }

    @CrossOrigin("http://localhost:3000")
    @PostMapping("/admin/product/category")
    public ProductCategoryEntity addCategory(@RequestParam String category) {
          ProductCategoryEntity savedCategory =
shopService.addCategory(category);
          return savedCategory;
    }

}
```

AdminShopServiceImpl.java

```
package com.cloudstore.service.admin;


import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import com.cloudstore.entity.CustomerEntity;
import com.cloudstore.entity.EnableStatusEnum;
import com.cloudstore.entity.ShopEntity;
import com.cloudstore.entity.UserAuthenticationEntity;
import com.cloudstore.model.EditShopModel;
import com.cloudstore.repository.ShopRepository;
import com.cloudstore.repository.UserAuthenticationRepository;
import com.cloudstore.service.authentication.UserLoginServiceInterface;
```

```
@Service
public class AdminShopServiceImpl implements AdminShopServiceInterface {

    @Autowired
    private ShopRepository shopRepository;

    @Autowired
    private UserLoginServiceInterface userLoginService;

    @Autowired
    private UserAuthenticationRepository userAuthenticationRepository;

    @Override
    public List<ShopEntity> findAllShops() {
         List<ShopEntity> shops = shopRepository.findAll();
         return shops;
    }

    @Override
    public void disableShops(String emails) {
         ShopEntity shops = shopRepository.findByEmail(emails);
         shops.setEnableStatus(EnableStatusEnum.ADMIN_DISABLED);
         shopRepository.save(shops);

         userLoginService.disableUsers(emails);
    }

    @Override
    public void enableShops(String emails) {
         ShopEntity shops = shopRepository.findByEmail(emails);
         shops.setEnableStatus(EnableStatusEnum.ENABLED);
         shopRepository.save(shops);

         userLoginService.enableUsers(emails);

    }

    @Override
    public ShopEntity editShops(EditShopModel editModel) {
         ShopEntity shop = shopRepository.findByEmail(editModel.getEmail());
         if(shop != null) {

               shop.setOwnerFullName(editModel.getOwnerFullName());
               shop.setMobile(editModel.getMobile());
```

```
shop.setImageUrl(editModel.getImageUrl());
                shop.setDocumentUrl(editModel.getDocumentUrl());
                shop.setStreetName(editModel.getStreetName());
                shop.setCityName(editModel.getCityName());
                shop.setPincode(editModel.getPincode());

                shopRepository.save(shop);

          }
          else {
                throw new UsernameNotFoundException("Customer with that email
does not exist");
          }
          return shop;
     }

}
```

PaymentController.java

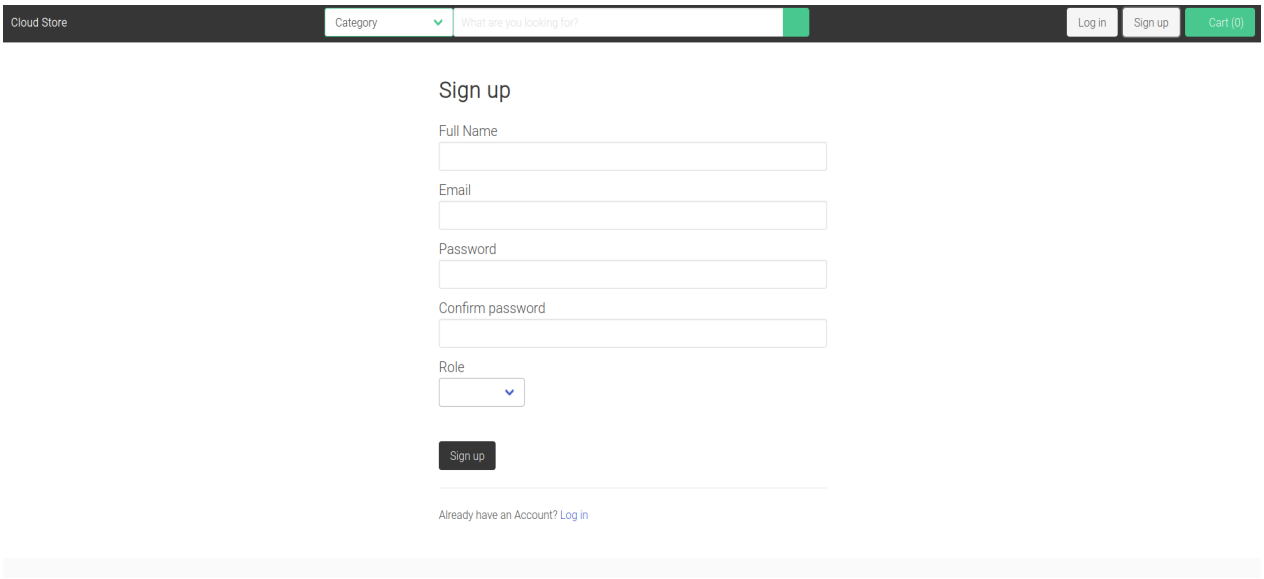
```
package com.cloudstore.controller;

import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.cloudstore.entity.CustomerEntity;
import com.cloudstore.entity.OrderEntity;
import com.cloudstore.entity.OrderProductList;
import com.cloudstore.entity.ProductEntity;
import com.cloudstore.model.AddressModel;
import com.cloudstore.model.OrderModel;
import com.cloudstore.model.OrderProductListModel;
```

```
import com.cloudstore.service.CustomerServiceInterface;
import com.cloudstore.service.ShopServiceInterface;
import com.cloudstore.utility.PaytmInitiateTransaction;
import io.jsonwebtoken.io.IOException;

@RestController
public class PaymentController {

    @Autowired
    private CustomerServiceInterface customerService;

    @Autowired
    private ShopServiceInterface shopService;

    @Autowired
    private PaytmInitiateTransaction paytmInitiate;


    @CrossOrigin("http://localhost:3000")
    @PostMapping("/user/customer/payment")
    public JSONObject createOrder(@RequestBody OrderModel orderModel) throws
Exception {
         Authentication usernamePasswordAuthenticationToken =
                     SecurityContextHolder.getContext().getAuthentication();
         String email = usernamePasswordAuthenticationToken.getName();
         CustomerEntity customer = customerService.customerInfo(email);

         System.out.println("Address Received: " +
orderModel.getAddressId());

         String[] orderIdAndAmount = shopService.createOrder(customer,
orderModel);

         String response =
paytmInitiate.initiateTransaction(orderIdAndAmount[0], orderIdAndAmount[1]);

         JSONParser jsonParser = new JSONParser();
         JSONObject responseJSON = (JSONObject) jsonParser.parse(response);
         responseJSON.put("orderId", orderIdAndAmount[0]);
         return responseJSON;
    }

    @CrossOrigin("http://localhost:3000")
    @PostMapping("/user/customer/deliveryaddress")
```

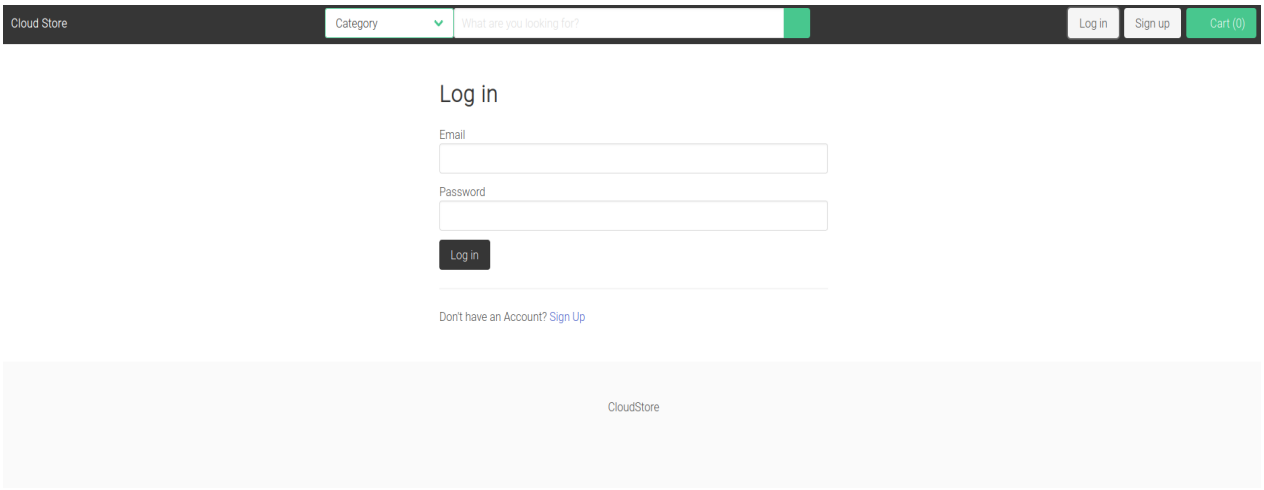
```
public String storeDeliveryAddress(@RequestBody AddressModel address) {

        String addressId = customerService.storeDeliveryAddress(address);
        return addressId;
    }

    @CrossOrigin("http://localhost:3000")
    @PostMapping("/user/customer/paymentResponse")
    public void paymentResponse(HttpServletRequest request,
HttpServletResponse response) throws java.io.IOException {
        String responseMsg = request.getParameter("RESPMSG");
        String responseCode = request.getParameter("RESPCODE");
        String orderId = request.getParameter("ORDERID");

        if(responseCode.equalsIgnoreCase("01")) {

response.sendRedirect("http://localhost:3000/cart/success?orderId=" +
orderId);
        }
        else {

response.sendRedirect("http://localhost:3000/cart/failure?errorMessage=" +
responseMsg);
        }
    }

}
```

9.2 Screen Shots

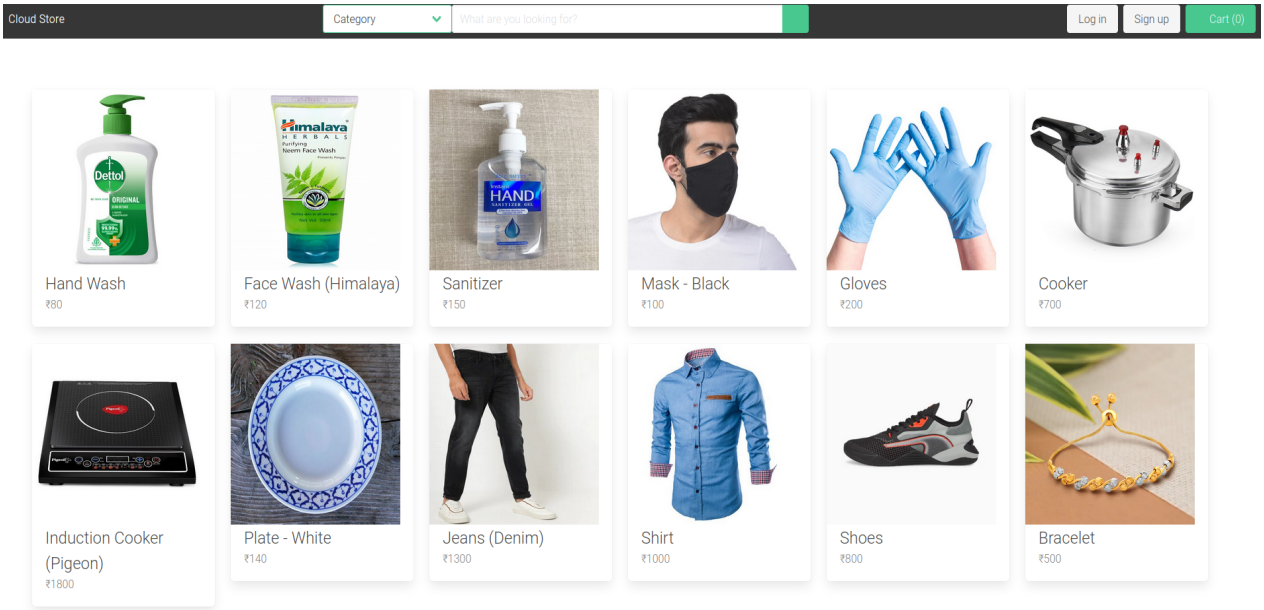
Signup



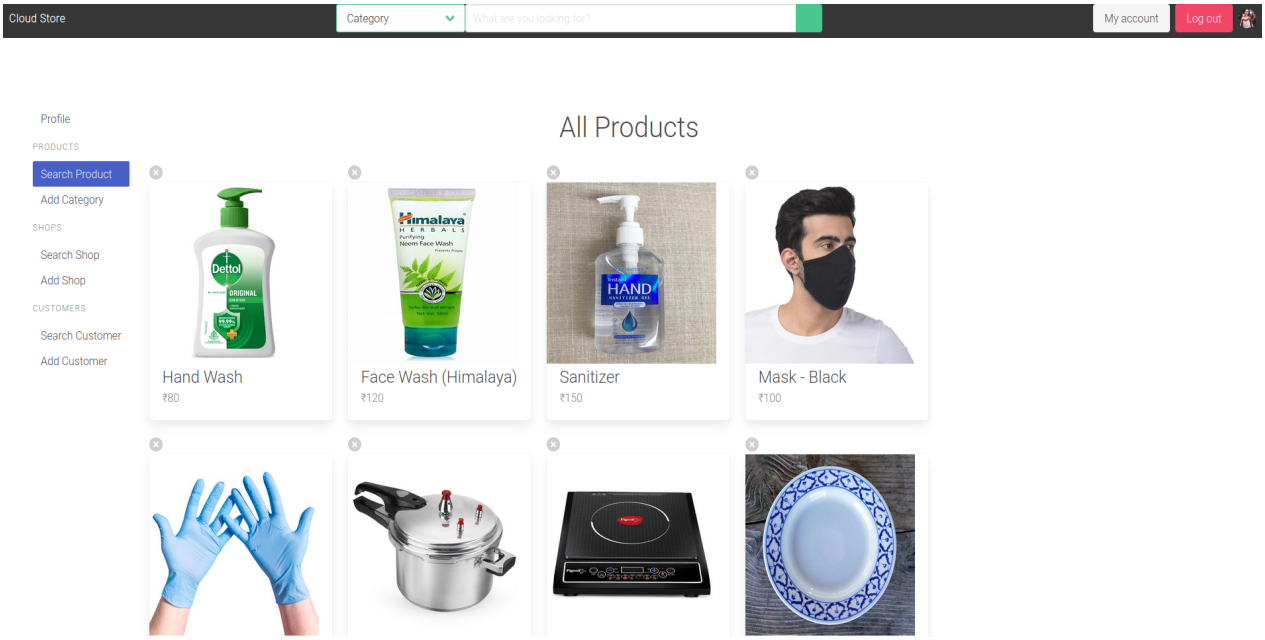
Login



HomePage



Admin Product Page



Payment Page