AURORA'S PG COLLEGE (MCA)
(Autonomous)
Accredited by NAAC with A+ Grade
Ramanthapur, Hyderabad, Telangana – 500013

PROJECT REPORT ON

Connecting Social Media To E-Commerce

AT

IntelliCloud Apps

Project Report submitted in partial fulfilment of the requirements for the award of the Degree of

MASTER O`F COMPUTER APPLICATIONS

Submitted by
Bhukya Veeranna
(H.T.No. 1325-23-862-074)

Under the Supervision of **Mr.**
Md. Ismail

2023 – 2025 Batch

AURORA'S POST -GRADUATE COLLEGE (M CA)
Ramanthapur, Hyderabad - 500013

CERTIFICATE

This is to certify that the project work entitled

Connecting Social Media To E-Commerce

Is a bonafide work done by

Bhukya Veeranna
1325-23-862-074

as part of the curriculum in the

DEPARTMENT OF COMPUTER SCIENCE
AURORA'S POST GRADUATE COLLEGE (MCA) Ramanthapur, Hyderabad – 500013

In partial fulfillment of requirement for award of

Master of Computer Applications

Osmania University, Hyderabad

This work has been carried out under our guidance.

Internal Guide | Head of Department | Principal

Internal Examiner | External Examiner

CERTIFICATE

23/06/2025

PROJECT COMPLETION CERTIFICATE

This is to certify that Mr. BHUKYA VEERANNA, studying in MCA at Aurora's Pg College (MCA), Ramanthapur. with HT.NO. 1325-23-862-074 has successfully completed the project work titled Connecting Social Media To E Commerce .

He has done this project using Python technology during the period from 03/03/2025 to 23/06/2025 under the guidance and supervision of Mr. Sateesh Kumar , Sr . Software Developer, Intellicloud Apps Pvt Ltd, Hyderabad .

He has completed the assigned project well within the time frame and h is sincere, hardworking and his conduct during this project is commendable.

We wish all the best to his future endeavors.

For IntelliCloud Apps Private Limited

INTELLICLOUD APPS PVT. LTD. HYDERABAD

DECLARATION

I, Bhukya Veeranna hereby declare that the project entitled **"Connecting Social Media To E-Commerce"** has been carried out by me in the IntellicloudApps. This project is submitted to Osmania University Hyderabad in partial fulfillment of requirements for the award of the degree of "MASTERS OF COMPUTER APPLICATIONS". The results embodied in this dissertation have not been submitted to any other University or institution for the award of Degree or Diploma.

(Signature of the student)

Bhukya Veeranna
1325-23-862-074

ACKNOWLEDGEMENT

I would like to express deep gratitude and respect to all those people behind the scene who guided, inspired in the completion of this project work.

I wish to convey my sincere thanks to Principal and Head of Department Computer Science, our project in charge **Mr.MOHAMMED ISMAIL**and project guide **Mr.LAXMI NARAYANA** for giving me the required guidance during this project work.

Last but not least I am very thankful to the faculty members of my college and friends for their suggestions and help me successfully completing this project.

Bhukya Veeranna

1325-23-862-074

TABLE OF CONTENTS

ORGANIZATION

Organization Profile

Founded in 2014, INTELLICLOUD APPS PVT. LTD. located at Kphb, Hyderabad, has a rich background in developing Mobile Applications and IOT projects, especially in solving latest problems, Software Development and continues its entire attention on achieving transcending excellence in the Development and Maintenance of Software Projects and Products in Many Areas.

In today's modern technological competitive environment, students in computer science stream want to ensure that they are getting guidance in an organization that can meet their professional needs. With our well-equipped team of solid information systems professionals, who study, design, develop, enhance, customize, implement, maintain, and support various aspects of information technology, students can be sure.

We understand the customer needs and develop their quality of professional life by simply making the technology readily usable for them. We practice exclusively in software development, network simulation, search engine optimization, customization, and system integration. Our project methodology includes techniques for initiating a project, developing the requirements, making clear assignments to the project team, developing a dynamic schedule, reporting status to executives and problem solving.

About The People:

As a team we have the clear vision and realize it too. As a statistical evaluation, the team has more than 40,000 hours of expertise in providing real-time solutions in the fields of Mobile Apps Development in Android, IOS, and Flutter, Web Designing, Web Development, Cloud Computing, Image Processing and Implementation, client Server Technologies in Java, (J2EE), ANDROID, Python, PHP, Oracle.

Our Vision:

“Impossible as Possible” this is our vision; we work according to our vision.

ABSTRACT

Abstract

The **Connecting Social Media to E-Commerce** system is a hybrid platform that integrates social networking features with online shopping functionalities. This system allows users (customers) to post updates, share preferences, interact with friends, and simultaneously shop for products, view orders, and receive notifications—all from a single platform. It also enables vendors to manage their products and orders, while admins oversee platform operations, manage users, and control content. The project combines the community engagement of social platforms with the transactional features of e-commerce systems to increase user interaction, personalization, and sales. The modules included are Customer, Vendor, and Admin, each with role-specific functionalities to streamline communication, content sharing, and product management.

SYSTEM ANALYSIS

Scope and Objective

Scope:

This system provides a dual-purpose platform where users can interact socially and engage in e-commerce activities. It covers user profile management, friend requests, content posting, personalized product recommendations, vendor product uploads, and administrative control.

Objective:

- To blend social media functionalities with e-commerce to increase customer engagement.
- To allow customers to interact, post, shop, and share preferences.
- To enable vendors to manage products and track customer orders.
- To provide administrators with complete control over platform content and users.

Literature Survey

Several platforms such as Facebook Marketplace, Instagram Shopping, and Pinterest have experimented with integrating shopping into their social media interfaces. While effective, these platforms still rely heavily on external links for purchases, and they often lack internal order or product management systems. Studies show that social proof—such as likes, shares, and friend recommendations—can increase customer trust and conversion rates in e-commerce.

Academic research also supports this trend. A study published in the *Journal of Retailing and Consumer Services* (2021) indicates that social interaction significantly boosts user retention and product interest on e-commerce sites. However, very few systems fully integrate both user-to-user social interaction and seamless shopping within one environment.

This system is designed to fill that gap by allowing customers to engage in social interaction while actively participating in a shopping ecosystem. It supports vendor control, customer personalization, and admin governance, making it a full-stack platform that unites two major online domains—social networking and commerce.

Existing System

In most current systems, social interaction and online shopping are conducted on separate platforms. Users may see ads or product suggestions on social media, but the purchase process requires redirecting to an external site. This often leads to user drop-off, poor engagement, and lack of trust.

Existing e-commerce platforms like Amazon or Flipkart focus solely on transactions without integrating user-generated content, personal networks, or social recommendations that influence purchasing behavior.

Drawbacks of Existing System

- Lack of direct social interaction or user networking within e-commerce platforms.
- Limited ability to customize product suggestions based on social preferences.
- No direct post-to-purchase integration.
- Reduced engagement due to absence of social proof (likes, friend suggestions, etc.).
- Fragmented experience requiring users to switch between apps.

Proposed System

The proposed **Connecting Social Media to E-Commerce** system addresses the above issues by merging social media features with e-commerce functionalities. Customers can create and view posts, interact with friends, add preferences, and shop—all in one platform. Vendors can upload products and track orders efficiently. Admins control notifications, content, and users. The system promotes community-driven purchasing behavior and enhances the user experience through personalization and social engagement.

Advantages of Proposed System

- Unified platform for both social and shopping activities.
- Increased customer engagement through social features.
- Personalized product suggestions based on preferences and network activity.
- Direct communication between vendors and customers.
- Centralized admin control over all users and content.
- Seamless user experience without needing to switch platforms.

Modules

CustomerModule:

After login, customers can manage their profiles, add posts, view others' posts, and maintain a friend list through request and approval features. They can also add product preferences, view and buy products, manage a shopping cart, and view their order history. Notifications keep them updated about orders and social activities.

VendorModule:

Vendors can log in to view orders, add and manage products, and access customer information. They have profile management and password change options, providing a secure and manageable system to maintain their online business operations.

AdminModule:

Admins have complete control over the system. They can view all users, vendors, and posts. They can manage categories, add/view notifications, and handle user contacts. Admins are also responsible for platform maintenance, ensuring smooth and secure operation of the system.

Software Requirements

- Operating System : Windows
- User Interface : HTML, CSS , JS and Bootstrap
- Programming Language : Python
- Web Applications : Django
- IDE/Workbench : Pycharm
- Database : MYSQL, SQL, SQLYOG
- Server Deployment : Apache Tomcat

Hardware Requirements

- Processor : Intel I3 Processor
- RAM : 4 GB
- Monitor : 15 inch color monitor or LED
- Hard disk : 256 GB
- Key board : Standard 102 keys
- Mouse : Optical

Functional Requirements:
Home
About
Contact

CUSTOMER:

- ❖ After Login:
  - Home
  - Add Post

- View My post
- View All Posts
- View Cart
- View Products
- View Notifications
- View My Orders
- Add Preference
- My Preference
- All Users
- My Friends
- Requests
- My Profile
- Change Password
- Logout

ADMIN:

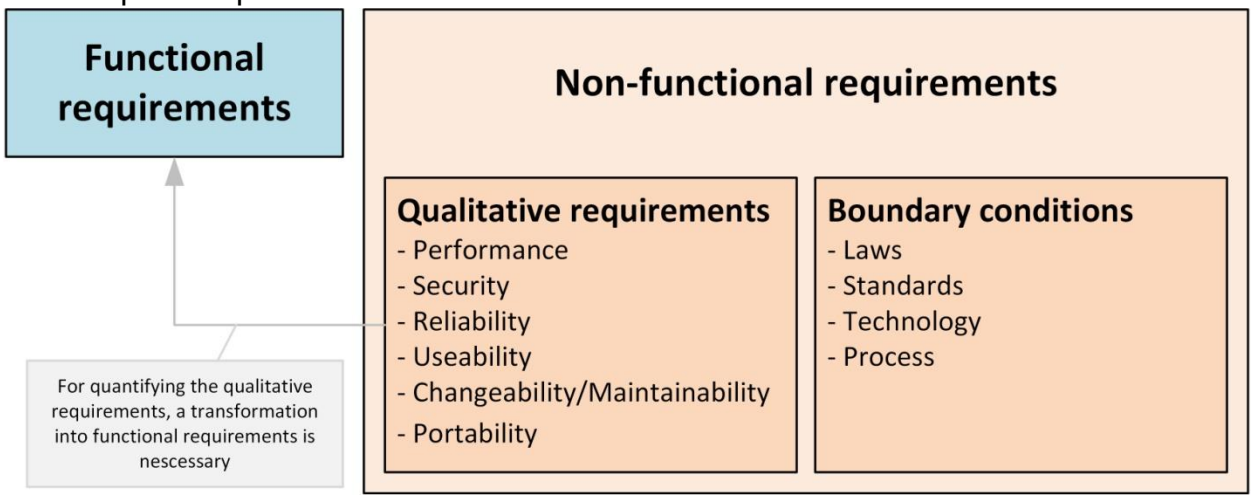
- After Login
  - Home
  - Add Notifications
  - View Notifications
  - Add Category
  - View Category
  - View Contacts
  - View Customers
  - View Vendors
  - View All Posts
  - Change Password
  - Logout

VENDOR

- After Login
  - Home
  - View Orders
  - Add Product
  - View Product
  - View Customers
  - My Profile
  - Change Password
  - Logout

Non Functional Requirements:

Non functional testing is a type of software testing that verifies non functional aspects of the product, such as performance, stability, and usability. Whereas functional testing verifies whether or not the product does what it is supposed to, non functional testing verifies how well the product performs.



The system should shows below non functional requirements

1. High Security
2. Multi browser and operating system compatibility
3. Mobile responsiveness
4. High quality and performance
5. Memory optimized and User Friendliness
6. Easy maintenance and scalable capability
7. It's should support good accessibility and usability
8. It should follow regulatory and compliance policies

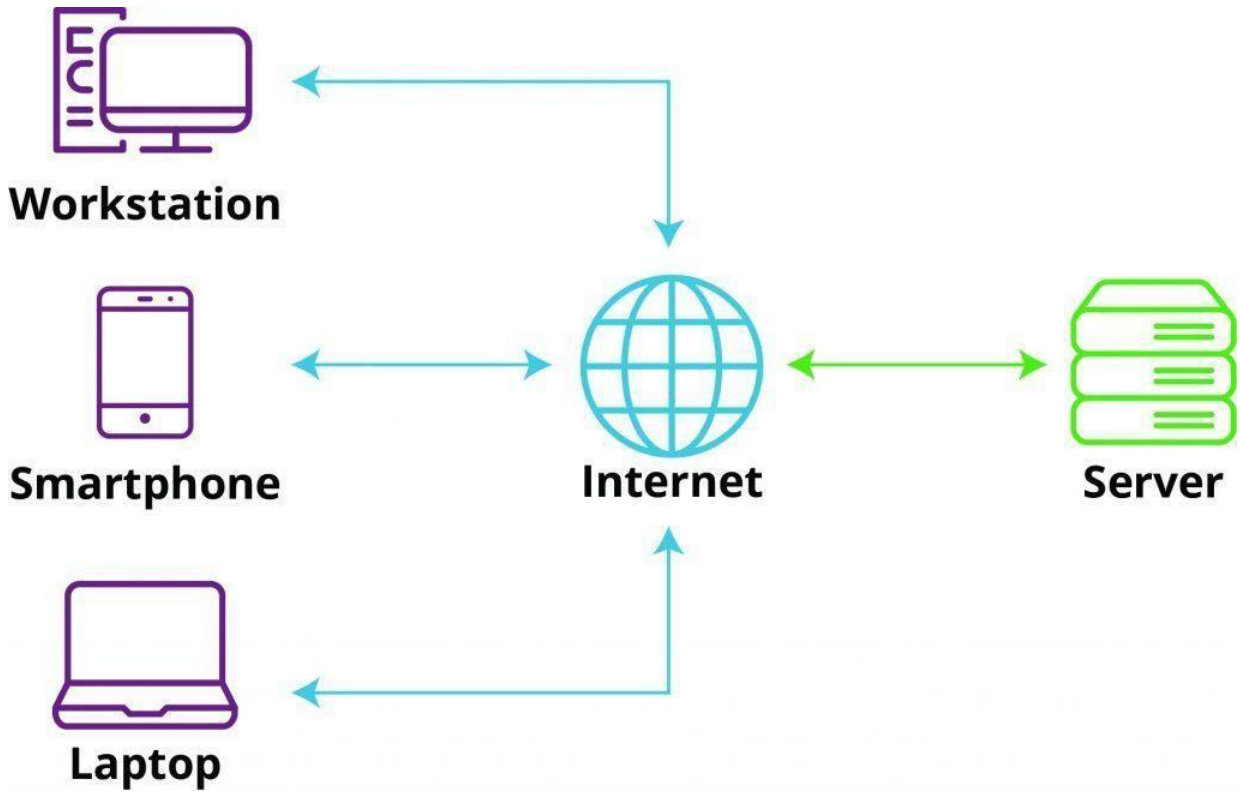
Feasibility Study:

The possibility of the undertaking is broke down during this stage and strategic plan is advanced with an exceptionally broad arrangement for the task and a few quotes. During project examination the likelihood investigation of the proposed project is to be completed. this is often to ensure that the proposed project isn't a weight to the organization. For practicality examination, some comprehension of the many necessities for the project is prime.

After the approval of the request to the organization and project guide, with an investigation being considered, the project request must be examined to determine precisely what the system requires.
Not all request projects are desirable or feasible. Some organization receives so many project requests from client users that only few of them are pursued. However, those projects that are both feasible and desirable should be put into schedule. After a project request is approved, it cost, priority, completion time and personnel requirement is estimated and used to determine where to add it to any project list. Truly speaking, the approval of those above factors, development works can be launched
An important outcome of preliminary investigation is the determination that the system request is feasible. This is possible only if it is feasible within limited resource and time. The different feasibilities that have to be analyzed are.

ECONOMICAL FEASIBILITY

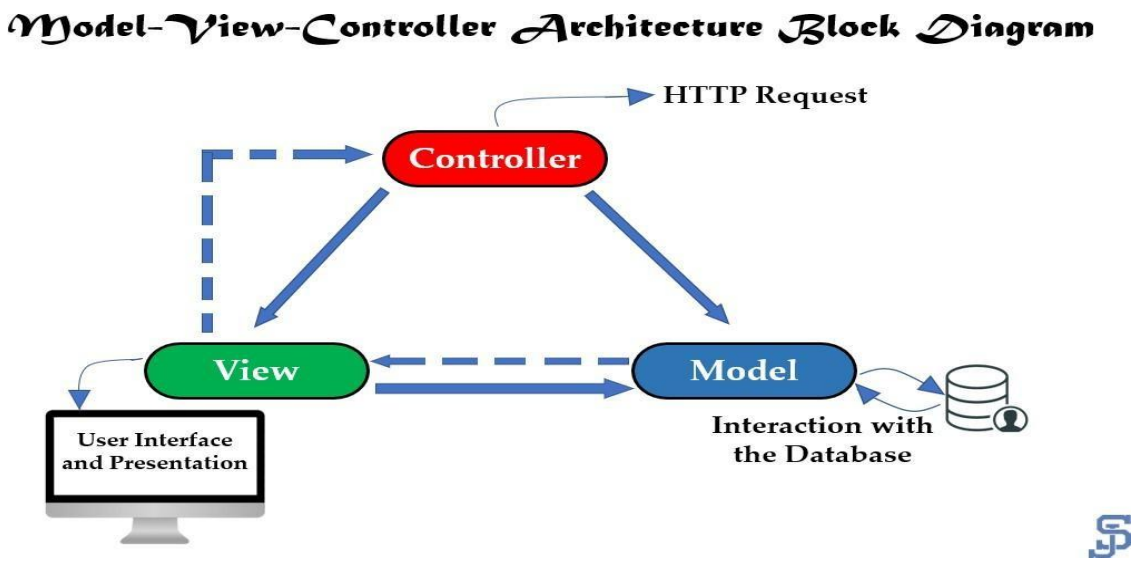
This examination is completed to see the financial effect that the framework will wear the association. The measure of store that the organization can fill the innovative work of the framework is constrained. The consumptions must be advocated. Accordingly, the created framework also inside the budget and this was accomplished in light of the very fact that an outsized portion of the advancements utilized are uninhibitedly accessible. Just the modified items must be bought. Economic Feasibility or Cost-benefit is an assessment of the economic justification for a computer based project. As hardware was installed from the beginning & for lots of purposes thus the cost on project of hardware is low. And also in this project work all software is free and open source. It can be developed using our personal machines. So its economically its feasible to complete the project.



Web Application MVC Architecture

The Model-View-Controller (MVC) is a well-known design pattern in the web development field. It is way to organize our code. It specifies that a program or application shall consist of data model, presentation information and control information. The MVC pattern needs all these components to be separated as different objects.
The MVC pattern architecture consists of three layers:

- **Model:** It represents the business layer of application. It is an object to carry the data that can also contain the logic to update controller if data is changed.
- **View:** It represents the presentation layer of application. It is used to visualize the data that the model contains.
- **Controller:** It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.



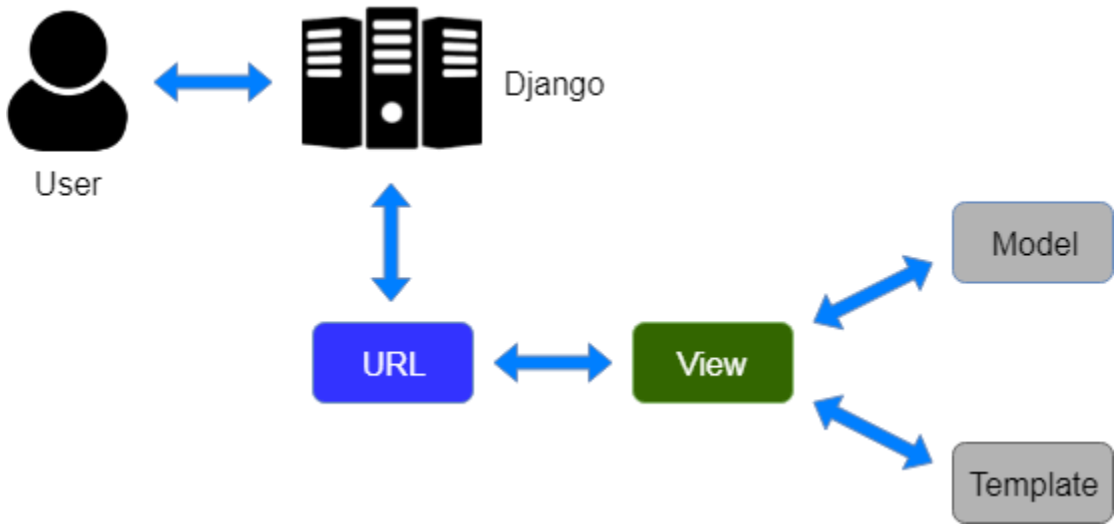
Web Application MVT Architecture

The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data.

The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template.

The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).

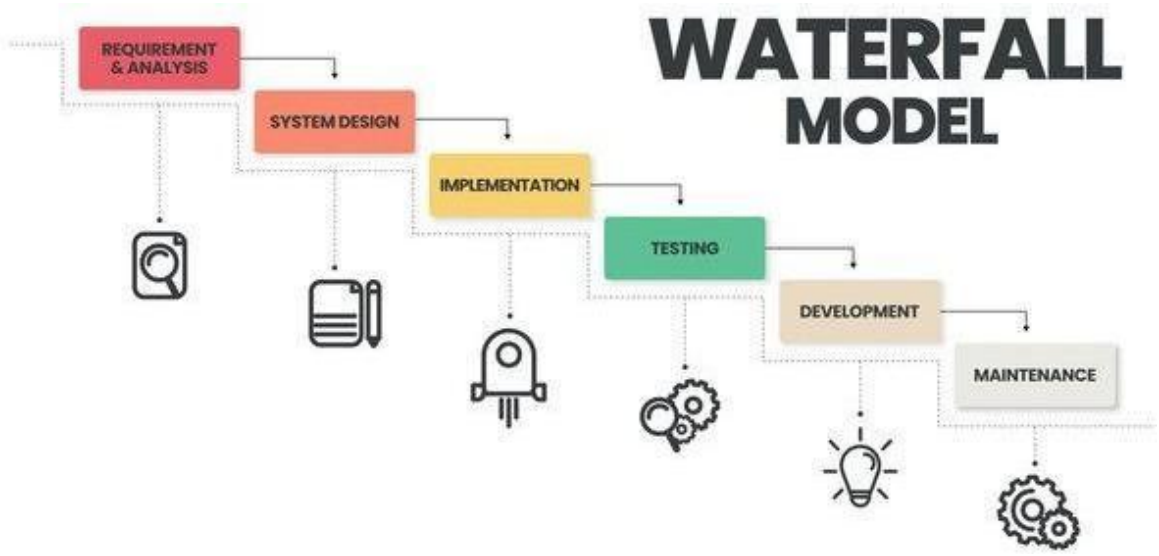
The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.



Process Models

1. **Waterfall Model:**

Waterfall model indicates that all the SDLC steps will be going on step by step. In this model once one stage completed then only it will start the another process. This will be implemented when the requirements well known and clearly defined.
In this model we can implement all the steps one by one. First we will do requirements gathering i.e in this step business analyst will collect the all functional requirements from the client and will discuss with the manager and development team. Second step is analysis where people can analyses the requirements so that they will do risk basement, risk mitigation, risk controlling steps. And also they will do the cost-time analysis to estimate the cost. They will also do the detail feasible study over the requirements of project.



In third phase called design page generally people will design the project i.e architect will prepare the UML diagrams and prototypes for project. And then developers will start programming or coding based on design . Finally testing team will test the content to identify and fix the bugs. After successful user acceptance testing application will be deployed and maintained in the client machine.

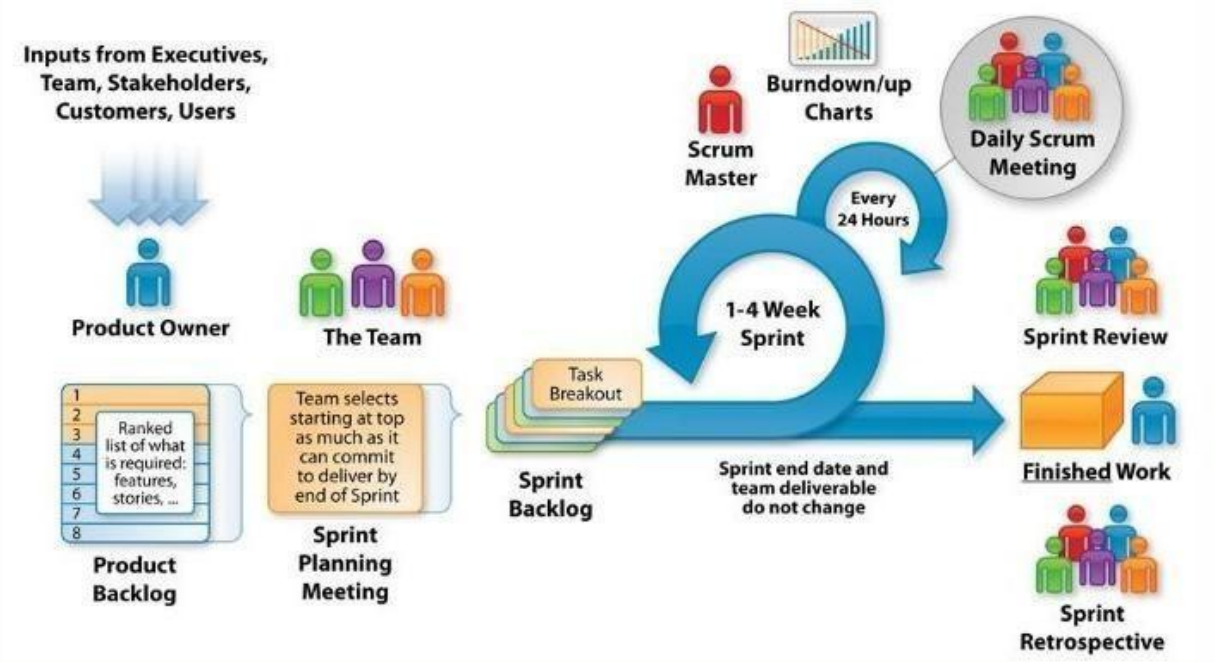
The waterfall model was selected as the SDLC model due to the following reasons:

- When the requirements are very clear and not changed.
- Available technology is enough to implement.
- Project is easy and simple to understand to finish.
- And if we don’t have any confusion in requirements.
- We can easily manage the review process and task management.
- Every stage defined clearly and with timelines

2. Agile Scrum Model:

Agile scrum methodology is the combination of the agile philosophy and the scrum framework. Agile means “incremental, allowing teams to develop projects in small increments. Scrum is one of the many types of agile methodology, known for breaking

projects down into sizable chunks called “sprints.” Agile scrum methodology is good for businesses that need to finish specific projects quickly.



Agile scrum methodology is a project management system that relies on incremental development. Each iteration consists of two- to four-week sprints, where the goal of each sprint is to build the most important features first and come out with a potentially deliverable product. More features are built into the product in subsequent sprints and are adjusted based on stakeholder and customer feedback between sprints. Whereas other project management methods emphasize building an entire product in one operation from start to finish, agile scrum methodology focuses on delivering several iterations of a product to provide stakeholders with the highest business value in the least amount of time.

Agile scrum methodology has several benefits. First, it encourages products to be built faster, since each set of goals must be completed within each sprint’s time frame. It also requires frequent planning and goal setting, which helps the scrum team focus on the current sprint’s objectives and increase productivity.

The greatest benefit of agile scrum methodology is its flexibility. With the sprint-based model, the scrum team typically receives feedback from stakeholders after each sprint.

If there are any problems or changes, the scrum team can easily and quickly adjust product goals during future sprints to provide more valuable iterations. This way, stakeholders are happier because they get exactly what they want after being involved every step of the way.

SYSTEM DESIGN

DESIGN:

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES:

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN:

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the

information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2. Select methods for presenting information.

3. Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

System Design:

This section consists of the UML diagrams related to the modules developed such as Activity diagram, Sequence diagram and Use case diagram.

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

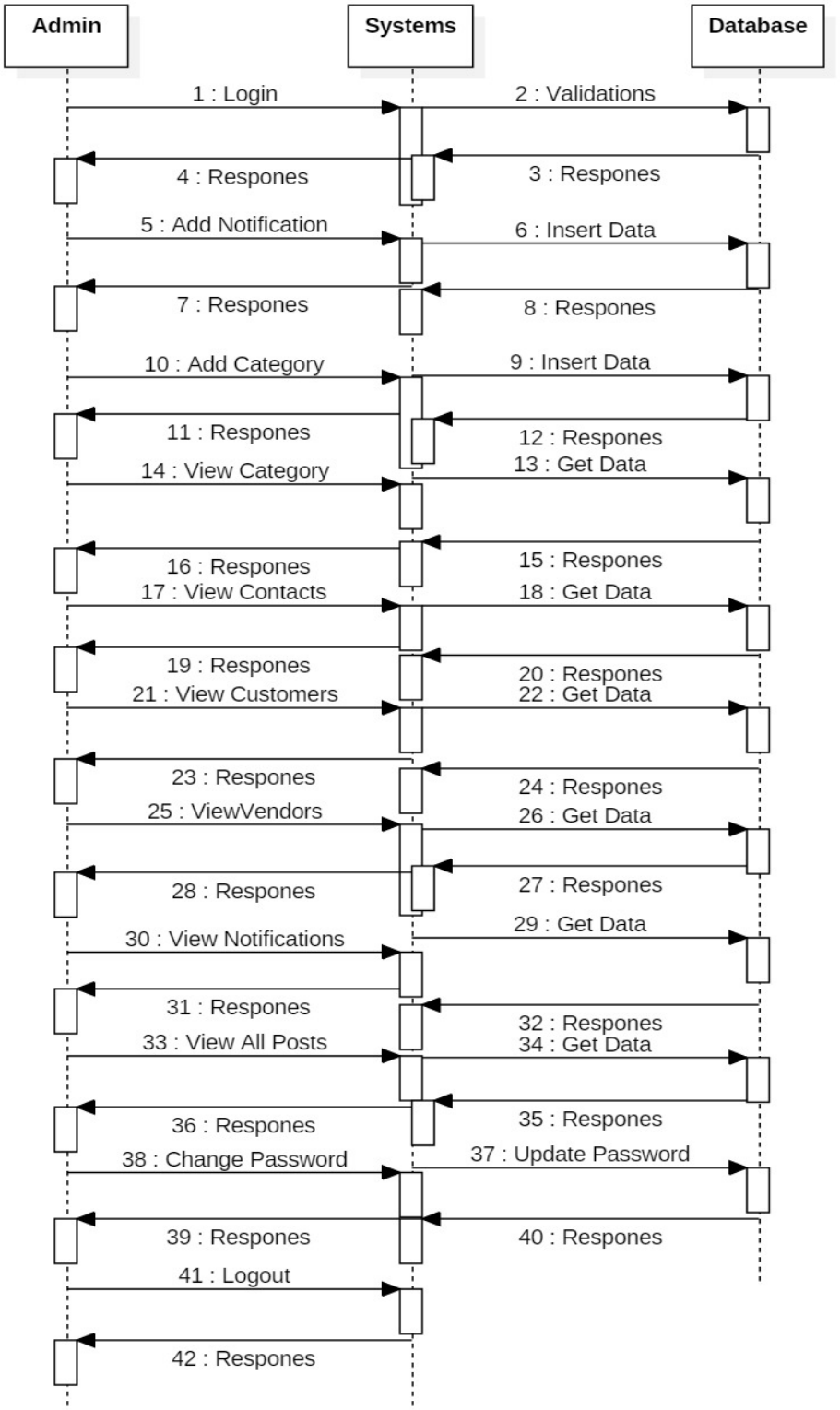
- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

UML Diagram

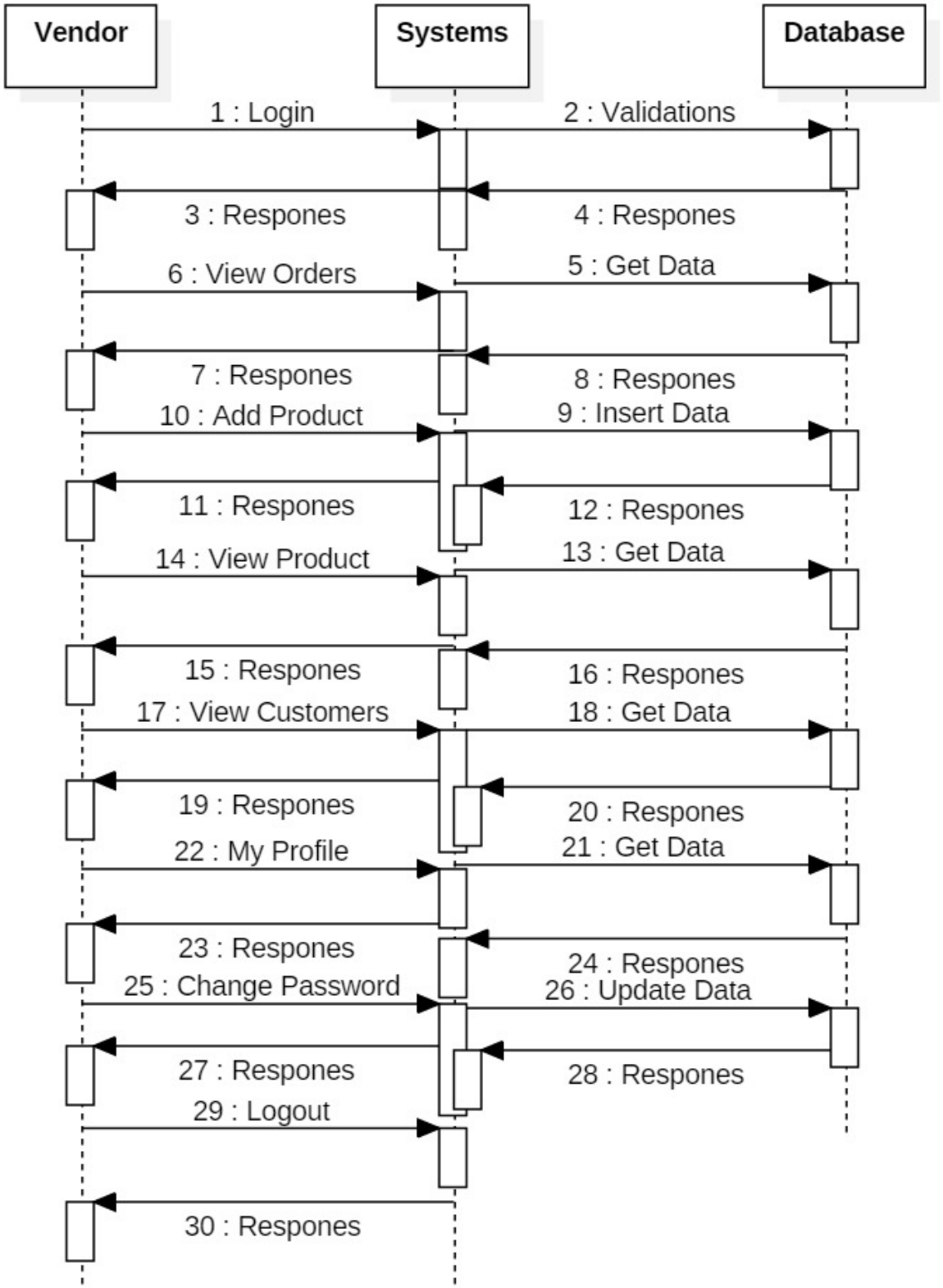
Sequence Diagram

A grouping outline in UML is a sort of association chart that shows how procedures work with each other and in what request. “It is a build of a Message Sequence Chart. Succession outlines are now and again called occasion charts, occasion situations, and timing graphs”.

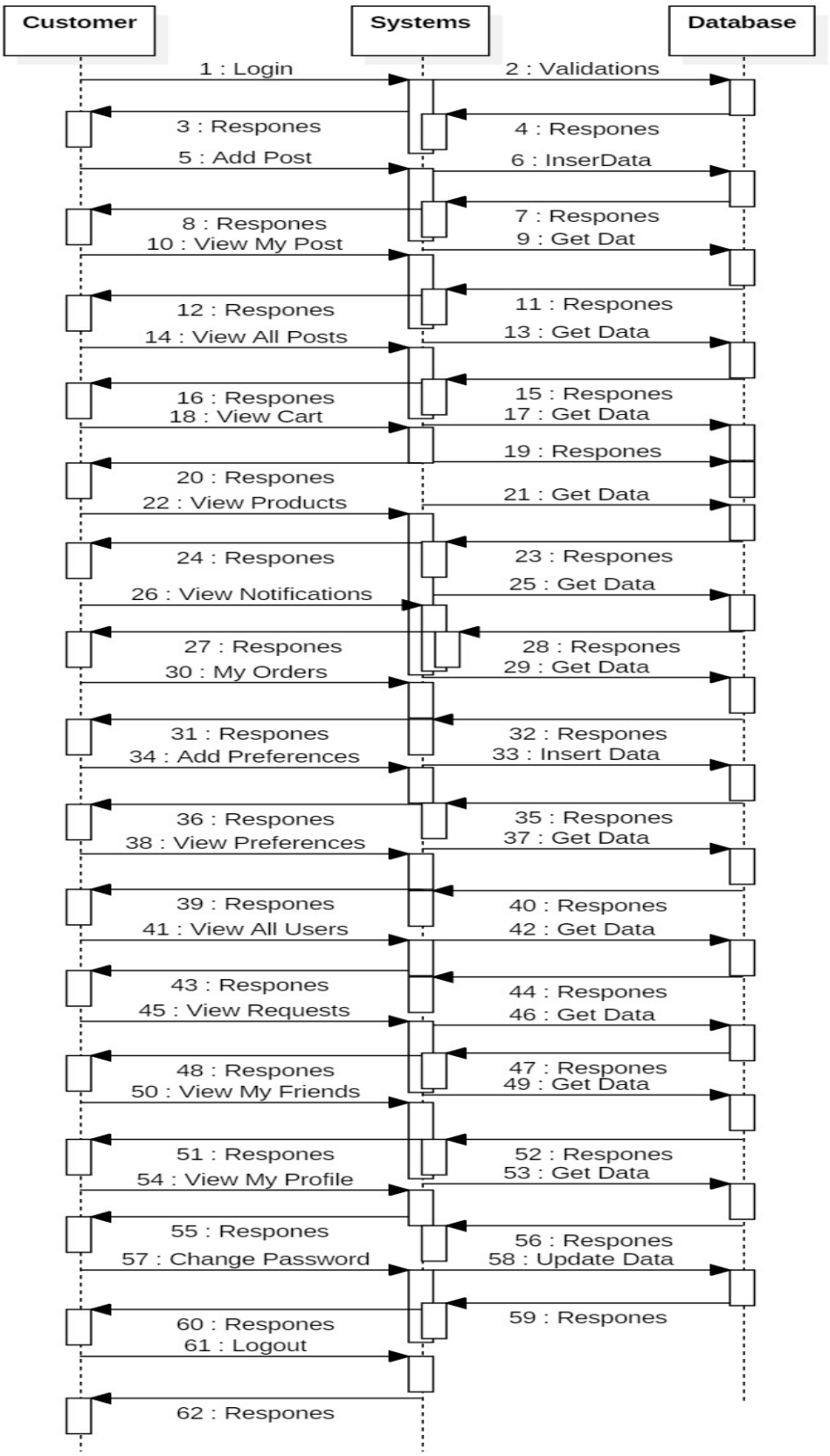
Admin-Sequence Diagram:



Vendor-Sequence Diagram:


Vendor
Systems
Database
1 : Login
2 : Validations
3 : Respones
4 : Respones
6 : View Orders
5 : Get Data
7 : Respones
8 : Respones
10 : Add Product
9 : Insert Data
11 : Respones
12 : Respones
14 : View Product
13 : Get Data
15 : Respones
16 : Respones
17 : View Customers
18 : Get Data
19 : Respones
20 : Respones
22 : My Profile
21 : Get Data
23 : Respones
24 : Respones
25 : Change Password
26 : Update Data
27 : Respones
28 : Respones
29 : Logout
30 : Respones

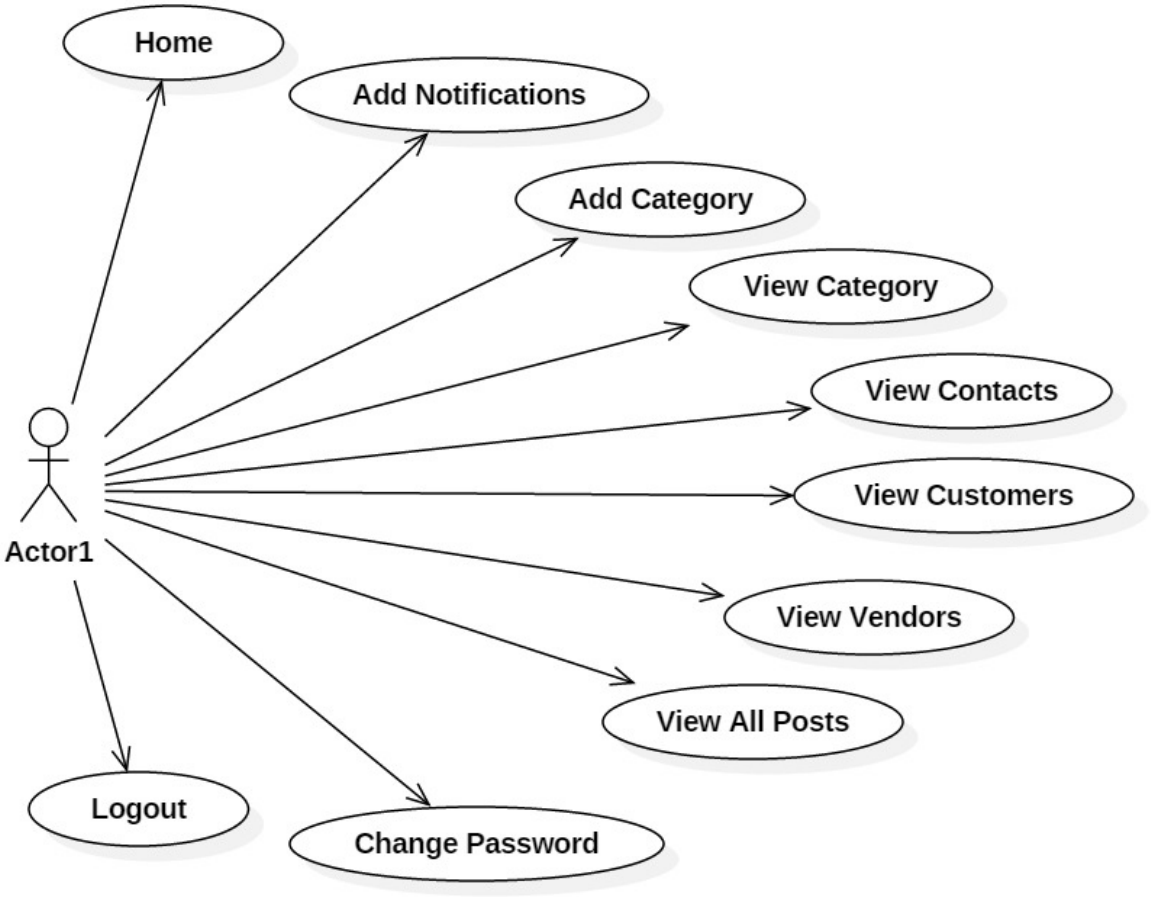
Customer-Sequence Diagram:


Customer
Systems
Database
1 : Login
2 : Validations
3 : Respones
4 : Respones
5 : Add Post
6 : InserData
8 : Respones
7 : Respones
10 : View My Post
9 : Get Dat
12 : Respones
11 : Respones
14 : View All Posts
13 : Get Data
16 : Respones
15 : Respones
18 : View Cart
17 : Get Data
19 : Respones
20 : Respones
22 : View Products
21 : Get Data
24 : Respones
23 : Respones
26 : View Notifications
25 : Get Data
27 : Respones
28 : Respones
30 : My Orders
29 : Get Data
31 : Respones
32 : Respones
34 : Add Preferences
33 : Insert Data
36 : Respones
35 : Respones
38 : View Preferences
37 : Get Data
39 : Respones
40 : Respones
41 : View All Users
42 : Get Data
43 : Respones
44 : Respones
45 : View Requests
46 : Get Data
48 : Respones
47 : Respones
50 : View My Friends
49 : Get Data
51 : Respones
52 : Respones
54 : View My Profile
53 : Get Data
55 : Respones
56 : Respones
57 : Change Password
58 : Update Data
60 : Respones
59 : Respones
61 : Logout
62 : Respones

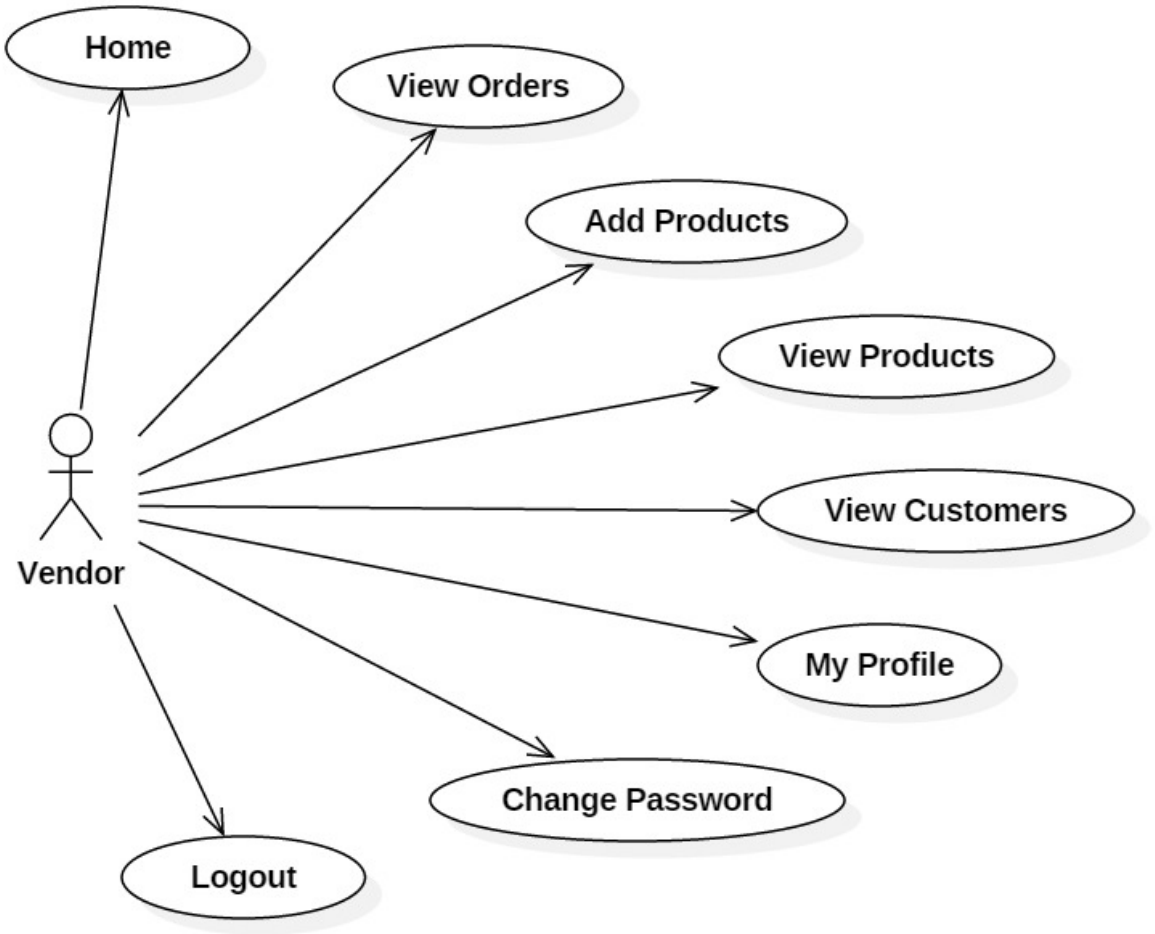
Use Case Diagram

An utilization case outline in the UML is a sort of conduct graph characterized by and made from a Use-case examination. Its motivation is to introduce a graphical diagram of the usefulness given by a framework regarding on-screen characters, their objectives (spoke to as use cases), and any conditions between those utilization cases. The fundamental motivation behind an utilization case outline is to indicate what framework capacities are performed for which on-screen character. Jobs of the on-screen characters in the framework can be delineated".

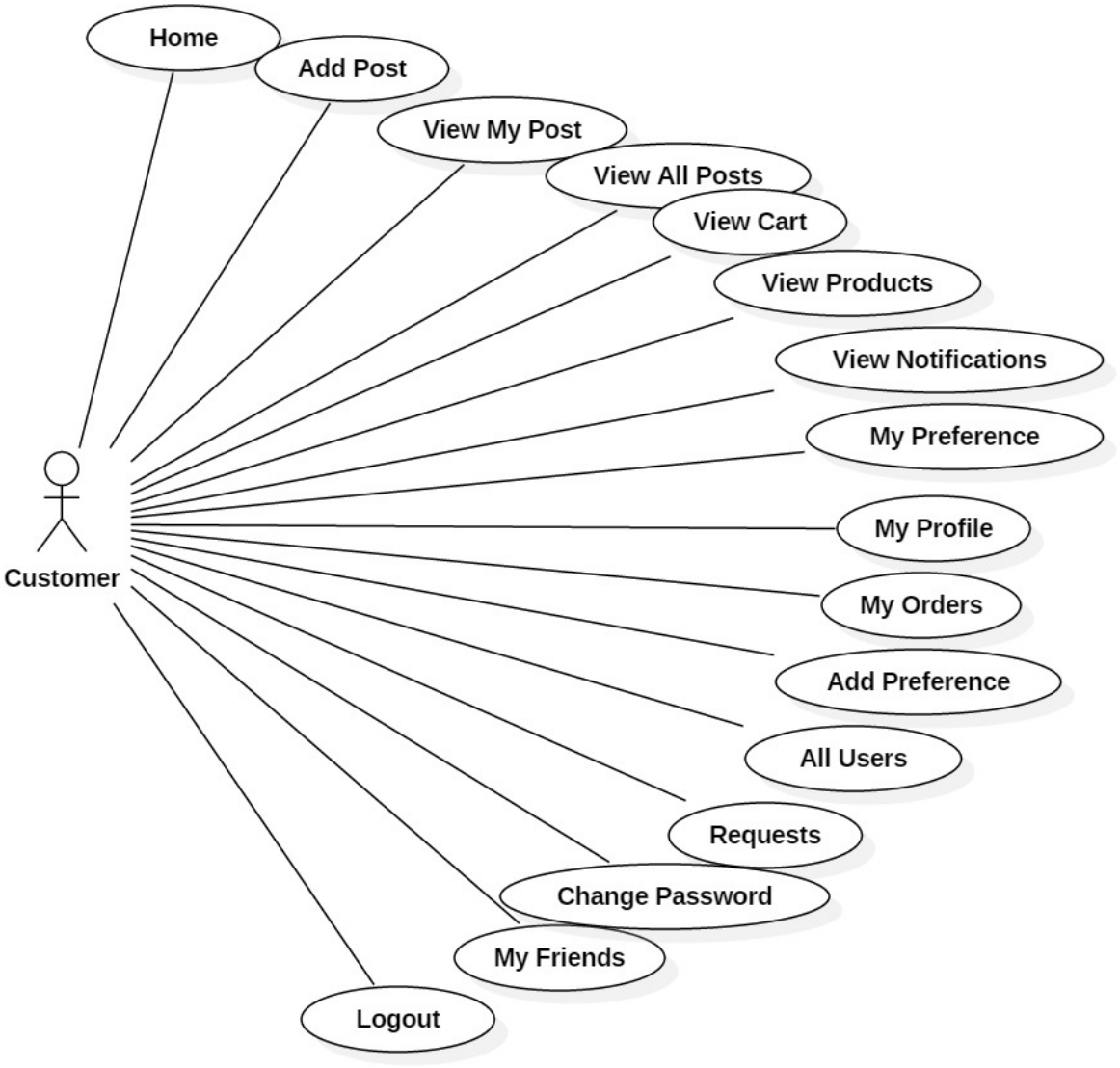
Admin-Use Case Diagram:



Vendor-Use Case Diagram:



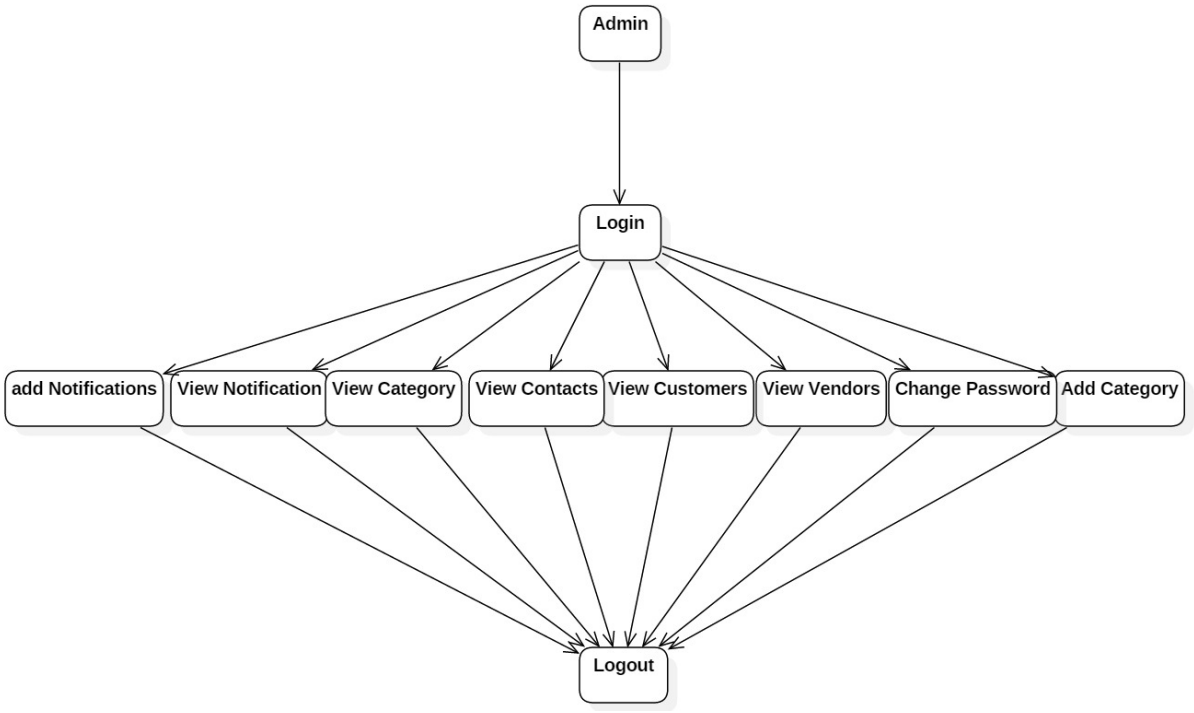
Customer-Use Case Diagram:



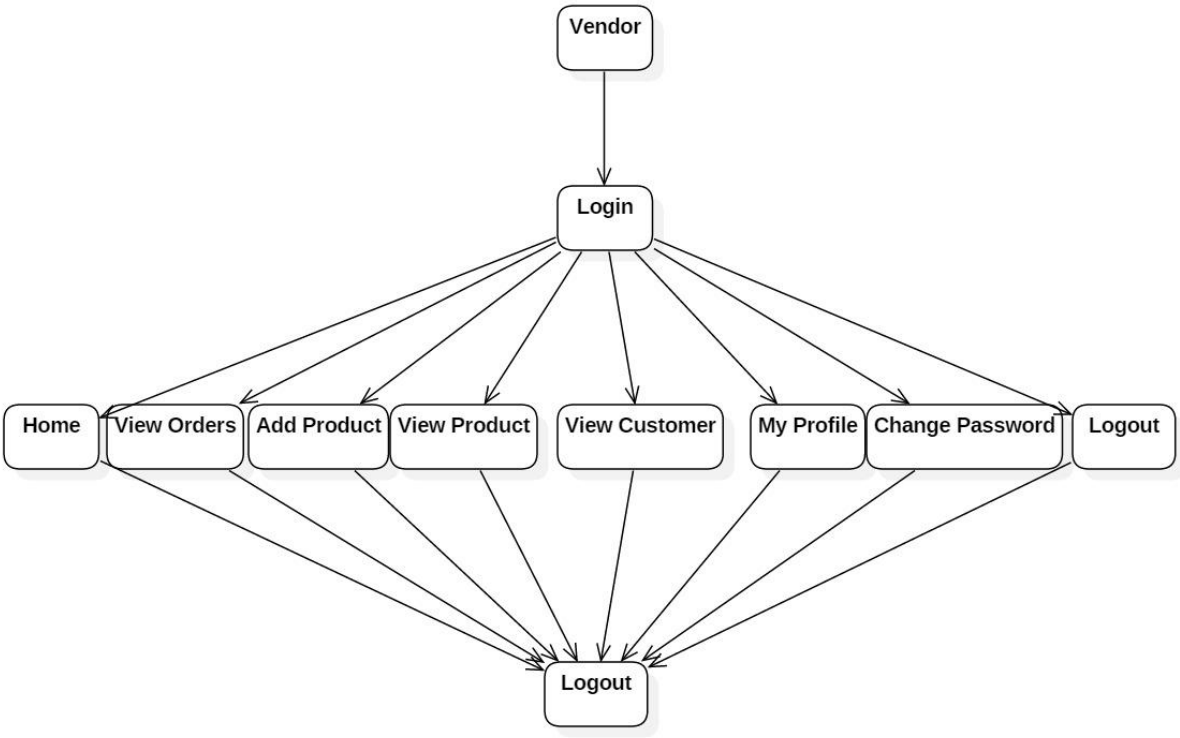
Activity Diagram

“Movement charts are graphical portrayals of work processes of stepwise exercises and activities with help for decision, emphasis and simultaneousness. In the Unified Modeling Language, movement graphs can be utilized to depict the business and operational bit by bit work processes of parts in a framework. A movement chart demonstrates the general progression of control”.

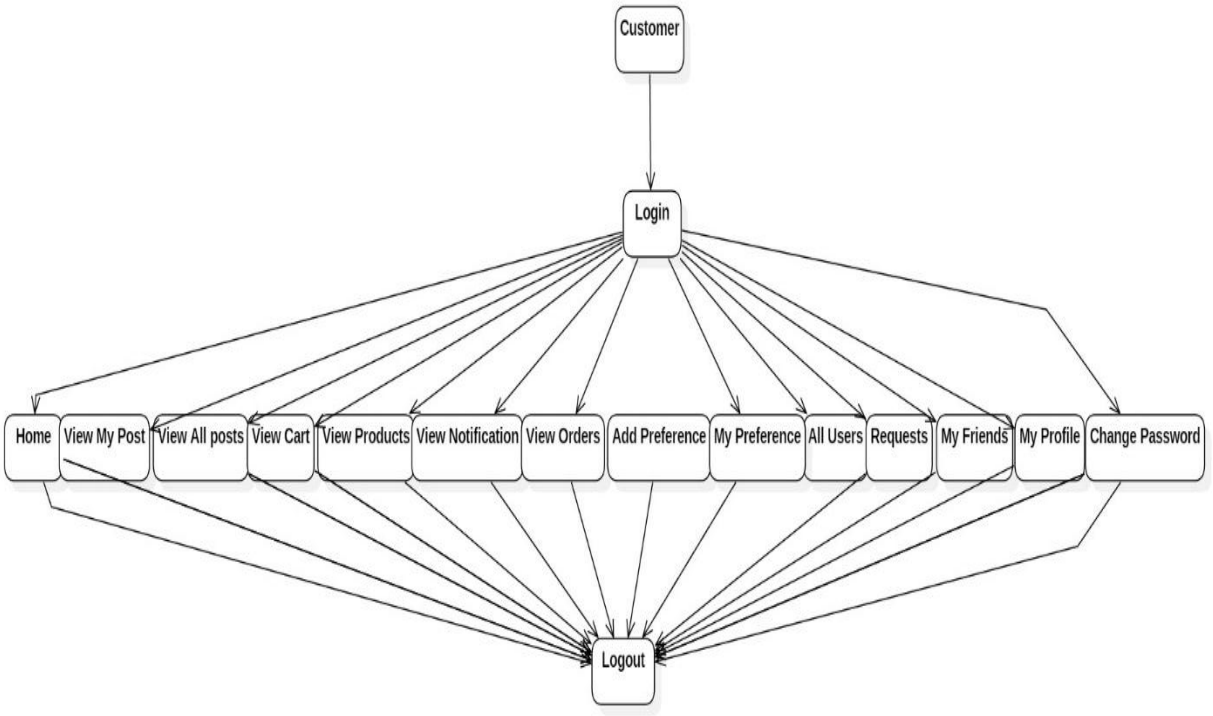
Admin-Activity Diagram:



Vendor-Activity Diagram:



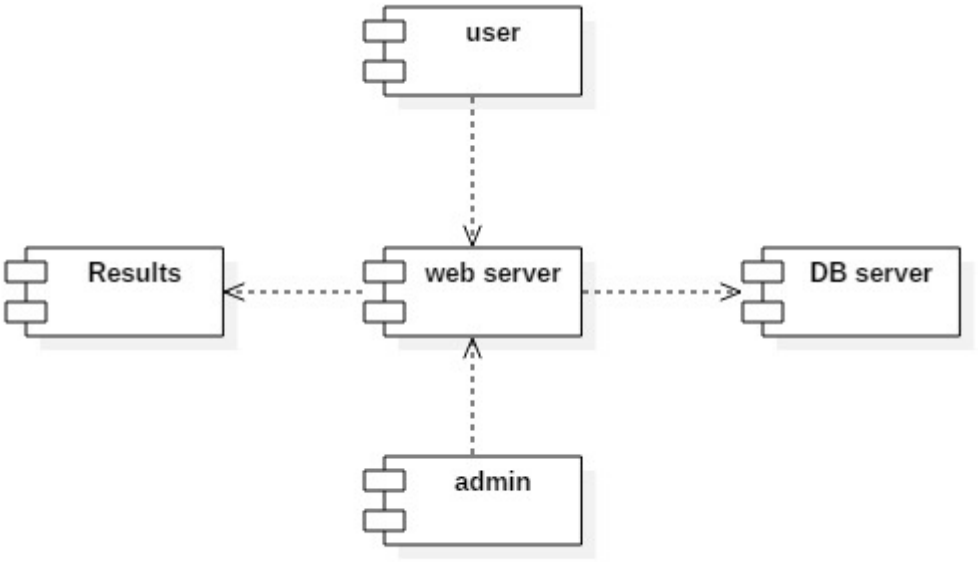
Customer-Activity Diagram:



Component Diagram

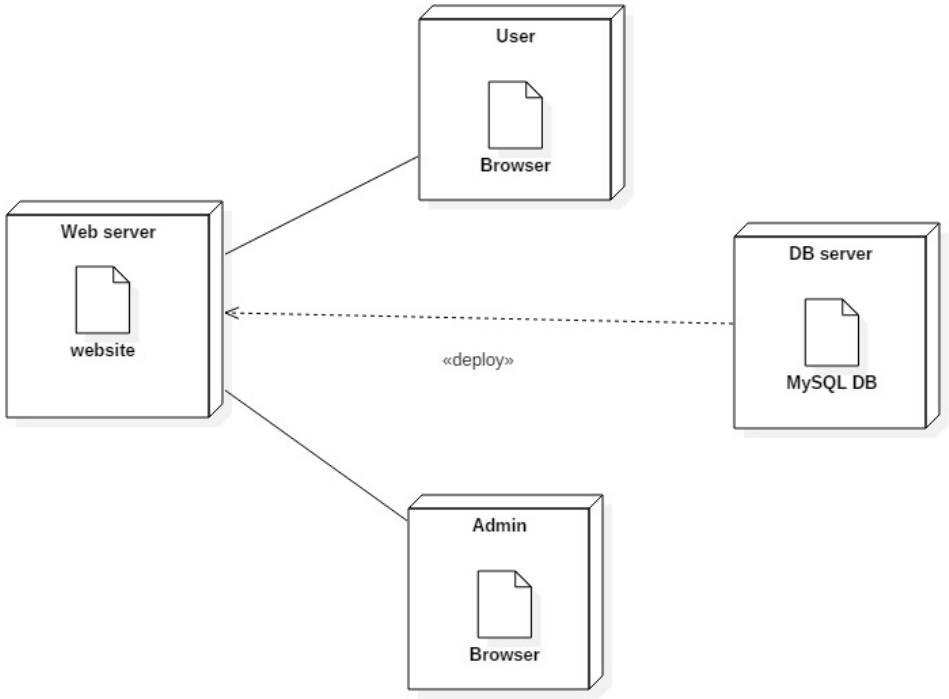
The component diagram is a special purpose diagram, which is used to visualize the static implementation view of a system. It represents the physical components of a system, or we can say it portrays the organization of the components inside a system. The components,

such as libraries, files, executables, etc.



Deployment Diagram

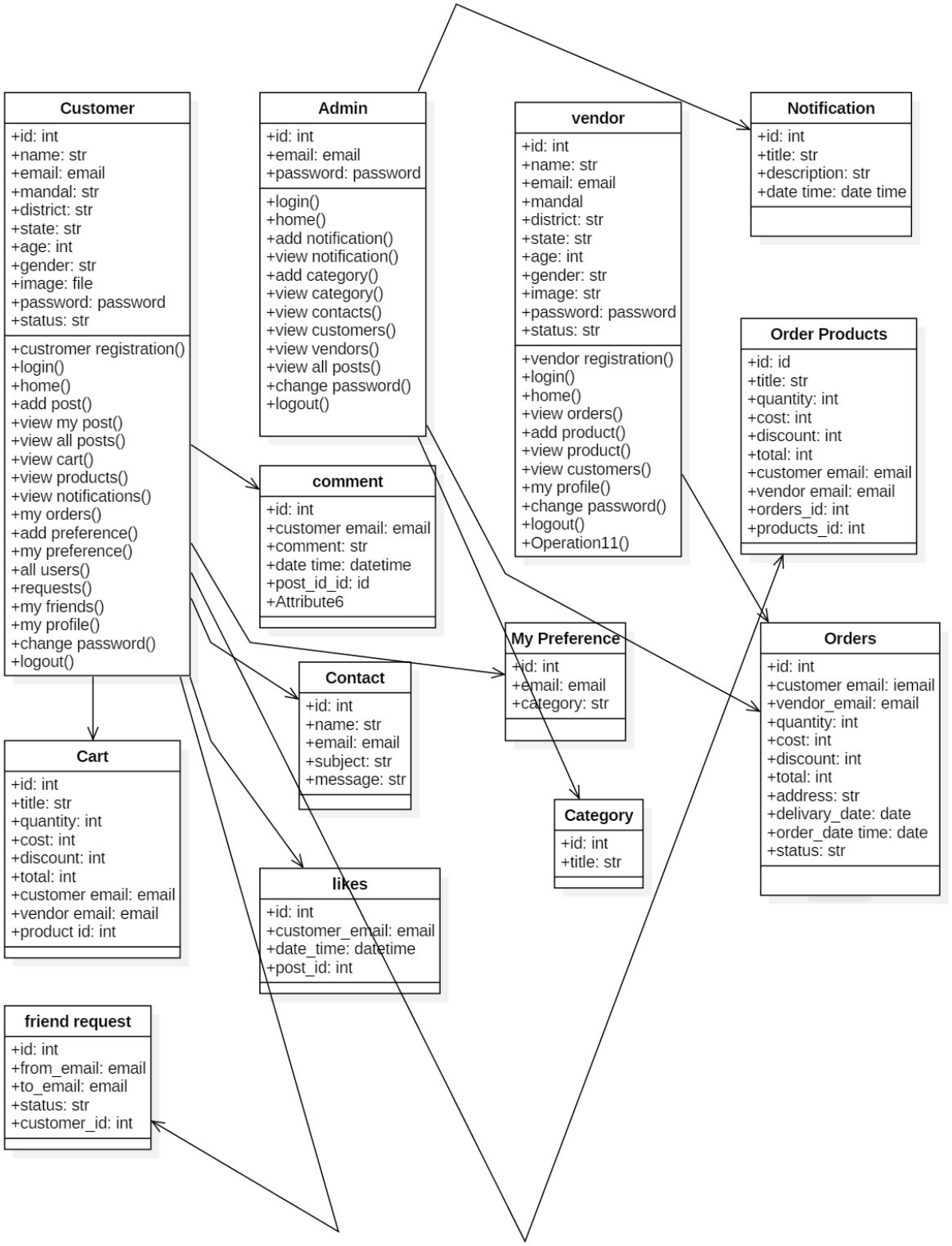
A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system



Class Diagram:

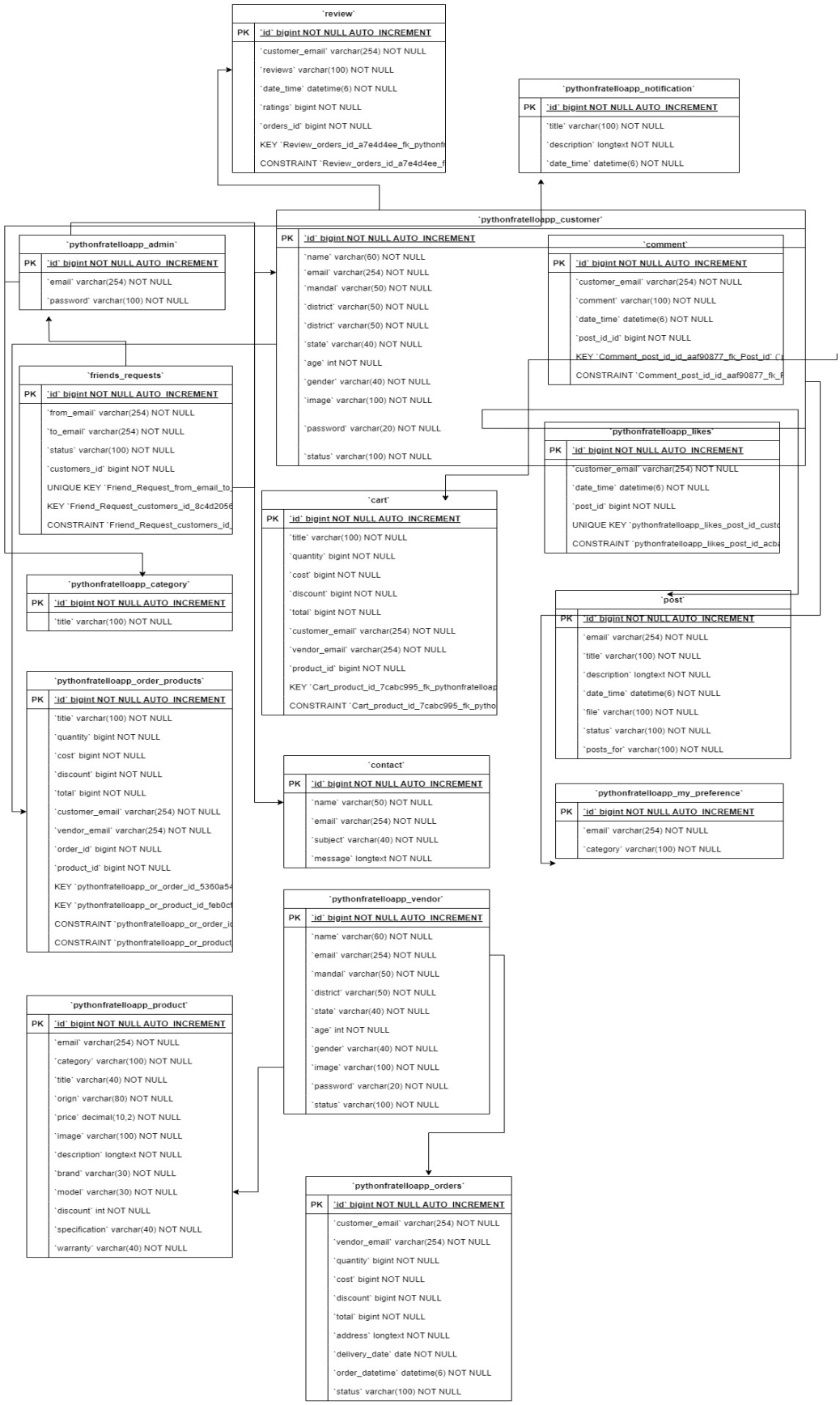
Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.


Customer
+id: int
+name: str
+email: email
+mandal: str
+district: str
+state: str
+age: int
+gender: str
+image: file
+password: password
+status: str
+custromer registration()
+login()
+home()
+add post()
+view my post()
+view all posts()
+view cart()
+view products()
+view notifications()
+my orders()
+add preference()
+my preference()
+all users()
+requests()
+my friends()
+my profile()
+change password()
+logout()
Admin
+id: int
+email: email
+password: password
+login()
+home()
+add notification()
+view notification()
+add category()
+view category()
+view contacts()
+view customers()
+view vendors()
+view all posts()
+change password()
+logout()
vendor
+id: int
+name: str
+email: email
+mandal
+district: str
+state: str
+age: int
+gender: str
+image: str
+password: password
+status: str
+vendor registration()
+login()
+home()
+view orders()
+add product()
+view product()
+view customers()
+my profile()
+change password()
+logout()
+Operation11()
Notification
+id: int
+title: str
+description: str
+date time: date time
Order Products
+id: id
+title: str
+quantity: int
+cost: int
+discount: int
+total: int
+customer email: email
+vendor email: email
+orders_id: int
+products_id: int
comment
+id: int
+customer email: email
+comment: str
+date time: datetime
+post_id_id: id
+Attribute6
My Preference
+id: int
+email: email
+category: str
Orders
+id: int
+customer email: iemail
+vendor_email: email
+quantity: int
+cost: int
+discount: int
+total: int
+address: str
+delivary_date: date
+order_date time: date
+status: str
Contact
+id: int
+name: str
+email: email
+subject: str
+message: str
Cart
+id: int
+title: str
+quantity: int
+cost: int
+discount: int
+total: int
+customer email: email
+vendor email: email
+product id: int
Category
+id: int
+title: str
likes
+id: int
+customer_email: email
+date_time: datetime
+post_id: int
friend request
+id: int
+from_email: email
+to_email: email
+status: str
+customer_id: int

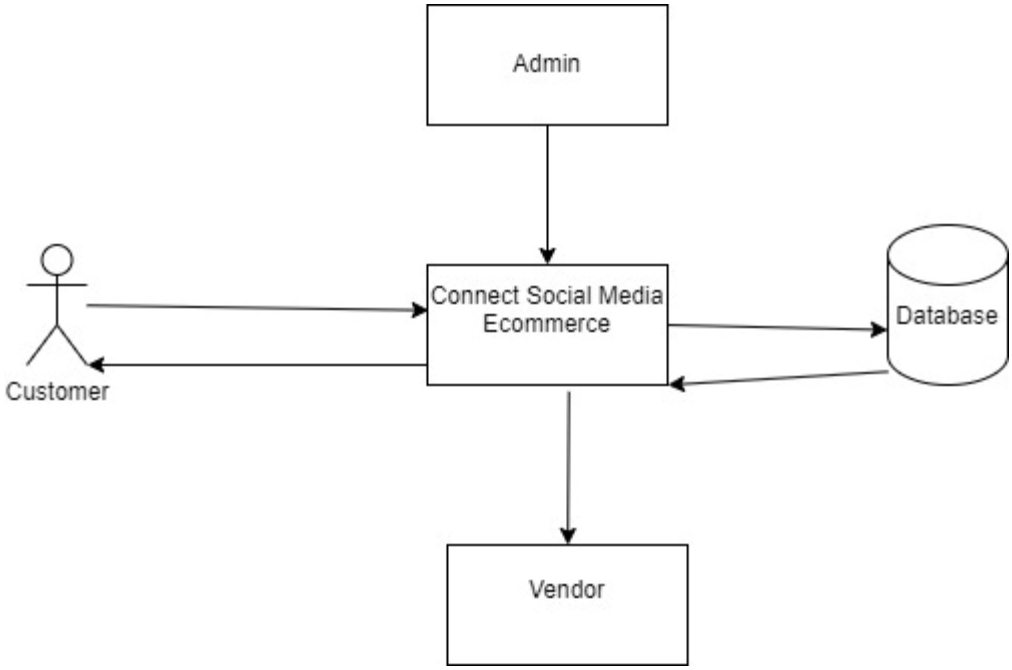
ER Diagram

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.


`review`
PK `id` bigint NOT NULL AUTO_INCREMENT
`customer_email` varchar(254) NOT NULL
`reviews` varchar(100) NOT NULL
`date_time` datetime(6) NOT NULL
`ratings` bigint NOT NULL
`orders_id` bigint NOT NULL
KEY `Review_orders_id_a7e4d4ee_fk_pythonfr
CONSTRAINT `Review_orders_id_a7e4d4ee_f
`pythonfratelloapp_notification`
PK `id` bigint NOT NULL AUTO_INCREMENT
`title` varchar(100) NOT NULL
`description` longtext NOT NULL
`date_time` datetime(6) NOT NULL
`pythonfratelloapp_customer`
PK `id` bigint NOT NULL AUTO_INCREMENT
`name` varchar(60) NOT NULL
`email` varchar(254) NOT NULL
`mandal` varchar(50) NOT NULL
`district` varchar(50) NOT NULL
`district` varchar(50) NOT NULL
`state` varchar(40) NOT NULL
`age` int NOT NULL
`gender` varchar(40) NOT NULL
`image` varchar(100) NOT NULL
`password` varchar(20) NOT NULL
`status` varchar(100) NOT NULL
`pythonfratelloapp_admin`
PK `id` bigint NOT NULL AUTO_INCREMENT
`email` varchar(254) NOT NULL
`password` varchar(100) NOT NULL
`comment`
PK `id` bigint NOT NULL AUTO_INCREMENT
`customer_email` varchar(254) NOT NULL
`comment` varchar(100) NOT NULL
`date_time` datetime(6) NOT NULL
`post_id_id` bigint NOT NULL
KEY `Comment_post_id_id_aaf90877_fk_Post_id`
CONSTRAINT `Comment_post_id_id_aaf90877_fk_P
`friends_requests`
PK `id` bigint NOT NULL AUTO_INCREMENT
`from_email` varchar(254) NOT NULL
`to_email` varchar(254) NOT NULL
`status` varchar(100) NOT NULL
`customers_id` bigint NOT NULL
UNIQUE KEY `Friend_Request_from_email_to_
KEY `Friend_Request_customers_id_8c4d2056
CONSTRAINT `Friend_Request_customers_id_
`pythonfratelloapp_likes`
PK `id` bigint NOT NULL AUTO_INCREMENT
`customer_email` varchar(254) NOT NULL
`date_time` datetime(6) NOT NULL
`post_id` bigint NOT NULL
UNIQUE KEY `pythonfratelloapp_likes_post_id_custo
CONSTRAINT `pythonfratelloapp_likes_post_id_acba
`cart`
PK `id` bigint NOT NULL AUTO_INCREMENT
`title` varchar(100) NOT NULL
`quantity` bigint NOT NULL
`cost` bigint NOT NULL
`discount` bigint NOT NULL
`total` bigint NOT NULL
`customer_email` varchar(254) NOT NULL
`vendor_email` varchar(254) NOT NULL
`product_id` bigint NOT NULL
KEY `Cart_product_id_7cabc995_fk_pythonfratelloap
CONSTRAINT `Cart_product_id_7cabc995_fk_pytho
`pythonfratelloapp_category`
PK `id` bigint NOT NULL AUTO_INCREMENT
`title` varchar(100) NOT NULL
`post`
PK `id` bigint NOT NULL AUTO_INCREMENT
`email` varchar(254) NOT NULL
`title` varchar(100) NOT NULL
`description` longtext NOT NULL
`date_time` datetime(6) NOT NULL
`file` varchar(100) NOT NULL
`status` varchar(100) NOT NULL
`posts_for` varchar(100) NOT NULL
`pythonfratelloapp_order_products`
PK `id` bigint NOT NULL AUTO_INCREMENT
`title` varchar(100) NOT NULL
`quantity` bigint NOT NULL
`cost` bigint NOT NULL
`discount` bigint NOT NULL
`total` bigint NOT NULL
`customer_email` varchar(254) NOT NULL
`vendor_email` varchar(254) NOT NULL
`order_id` bigint NOT NULL
`product_id` bigint NOT NULL
KEY `pythonfratelloapp_or_order_id_5360a54
KEY `pythonfratelloapp_or_product_id_feb0ct
CONSTRAINT `pythonfratelloapp_or_order_id
CONSTRAINT `pythonfratelloapp_or_product
`contact`
PK `id` bigint NOT NULL AUTO_INCREMENT
`name` varchar(50) NOT NULL
`email` varchar(254) NOT NULL
`subject` varchar(40) NOT NULL
`message` longtext NOT NULL
`pythonfratelloapp_my_preference`
PK `id` bigint NOT NULL AUTO_INCREMENT
`email` varchar(254) NOT NULL
`category` varchar(100) NOT NULL
`pythonfratelloapp_vendor`
PK `id` bigint NOT NULL AUTO_INCREMENT
`name` varchar(60) NOT NULL
`email` varchar(254) NOT NULL
`mandal` varchar(50) NOT NULL
`district` varchar(50) NOT NULL
`state` varchar(40) NOT NULL
`age` int NOT NULL
`gender` varchar(40) NOT NULL
`image` varchar(100) NOT NULL
`password` varchar(20) NOT NULL
`status` varchar(100) NOT NULL
`pythonfratelloapp_product`
PK `id` bigint NOT NULL AUTO_INCREMENT
`email` varchar(254) NOT NULL
`category` varchar(100) NOT NULL
`title` varchar(40) NOT NULL
`orign` varchar(80) NOT NULL
`price` decimal(10,2) NOT NULL
`image` varchar(100) NOT NULL
`description` longtext NOT NULL
`brand` varchar(30) NOT NULL
`model` varchar(30) NOT NULL
`discount` int NOT NULL
`specification` varchar(40) NOT NULL
`warranty` varchar(40) NOT NULL
`pythonfratelloapp_orders`
PK `id` bigint NOT NULL AUTO_INCREMENT
`customer_email` varchar(254) NOT NULL
`vendor_email` varchar(254) NOT NULL
`quantity` bigint NOT NULL
`cost` bigint NOT NULL
`discount` bigint NOT NULL
`total` bigint NOT NULL
`address` longtext NOT NULL
`delivery_date` date NOT NULL
`order_datetime` datetime(6) NOT NULL
`status` varchar(100) NOT NULL

Data Flow Diagram

A data-flow diagram is a way of representing a flow of data through a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow — there are no decision rules and no loops. A ***data flow diagram*** (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows,



IMPLEMENTATION

SAMPLE CODE:

VIEWS:

```
from itertools import chain

from django.contrib import messages
from django.db.models import Q, Count
from django.shortcuts import render, redirect, get_object_or_404
from django.utils import timezone

from .models import Admin, Customer, Vendor, Notification, Category, Product, Cart, Orders, Order_Products, \
    My_Preference, Contact, Likes
from .forms import ContactForm, CustomerForm, VendorForm, NotificationForm, CategoryForm, ProductForm, CartForm, \
    OrderForm, Order_ProductsForm, AdminForm, My_PreferenceForm, ReviewForm, Review, Friends_Requests, PostForm, Post, \
    Comment, CommentForm
from django.http import JsonResponse, HttpResponseForbidden


# Create your views here.

def index(request):
    return render(request, "index.html", {})


def doctor(request):
    return render(request, "doctor.html", {})


def departments(request):
    return render(request, "departments.html", {})


def department_single(request):
    return render(request, "department-single.html", {})


def blog_single(request):
    return render(request, "blog-single.html", {})
```

```
def blog(request):
    return render(request, "blog.html", {})


def about(request):
    return render(request, "about.html", {})


def contact(request):
    if request.method == "POST":
        form = ContactForm(request.POST)
        print(form.errors)
        if form.is_valid():
            form.save()
            return render(request, "contact.html", {"msg": "Thanks For Contacting Us"})

        return render(request, "contact.html", {"msg": "While Contacting Us Your Getting
Error"})
    return render(request, "contact.html", {})


def customer_register_form(request):
    if request.method == "POST":
        form = CustomerForm(request.POST, request.FILES)
        email = request.POST['email']
        if Customer.objects.filter(email=email).exists():
            return render(request, "customer_register_form.html",
                          {"msg": "This Email Already Exists Please Try With Another Email"})

        else:
            if form.is_valid():
                print(form.errors)
                form.save()
                return render(request, "customer_login.html", {"msg": "Registered
Successfully Done"})
            return render(request, "customer_register_form.html", {"msg": "Not Registered
Successfully Done"})
    return render(request, "customer_register_form.html", {})


def customer_login(request):
    if request.method == 'POST':
        email = request.POST['email']
```

```
        password = request.POST['password']
        log = Customer.objects.filter(email=email, password=password)
        if log.exists():
            if log[0].status == "Accepted":
                request.session['email'] = email
                return render(request, "customer_home.html", {"msg": "Accepted Successfully"})
            elif log[0].status == "Rejected":
                return render(request, "customer_login.html", {"msg": "Rejected login"})

            else:
                return render(request, "customer_login.html", {"msg": "Pending"})
        return render(request, "customer_login.html", {"msg": "Login Failed"})
    return render(request, "customer_login.html", {})


def customer_home(request):
    return render(request, "customer_home.html", {})


def customer_logout(request):
    return redirect('/customer_login')


def customer_profile(request):
    email = request.session['email']
    profile = Customer.objects.get(email=email)
    return render(request, "customer_profile.html", {"x": profile})




def admin_login(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        log = Admin.objects.filter(email=email, password=password)
        if log.exists():
            request.session['email'] = email
            return render(request, "admin_home.html", {})
        return render(request, "admin_login.html", {"msg": "Login Failed"})
    return render(request, "admin_login.html", {})
```

```
def admin_Logout(request):
    request.session.flush()
    return redirect('/admin_login')


def admin_home(request):
    return render(request, "admin_home.html", {})


def admin_viewcontact(request):
    contact = Contact.objects.all()
    return render(request, "admin_viewcontact.html", {"contact": contact})


def admin_viewcustomer(request):
    customer = Customer.objects.all()
    return render(request, "admin_viewcustomer.html", {"customer": customer})


def accept_customer(request, id):
    cus = Customer.objects.get(id=id)
    cus.status = 'Accepted'
    cus.save()
    return redirect('/admin_viewcustomer')


def reject_customer(request, id):
    cus = Customer.objects.get(id=id)
    cus.status = 'Rejected'
    cus.save()
    return redirect('/admin_viewcustomer')


def customer_edit(request):
    email = request.session['email']
    cus = Customer.objects.get(email=email)
    return render(request, 'customer_edit.html', {'x': cus})


def update(request):
```

```
    if request.method == 'POST':
        id = request.POST['id']
        cus = Customer.objects.get(id=id)
        form = CustomerForm(request.POST, request.FILES, instance=cus)
        print(form.errors)
        if form.is_valid():
            form.save()
            return redirect('/customer_profile')
        return render(request, "customer_edit.html", {})
    return render(request, "customer_edit.html", {})


def is_login(request):
    if request.session._contains_('email'):
        return True
    else:
        return False


def customer_change_password(request):
    email = request.session.get("email")
    if not email:
        return redirect('/customer_login')

    if is_login(request):
        if request.method == 'POST':
            password = request.POST['password']
            new_password = request.POST['new_password']
            if password == new_password:
                return render(request, "customer_change_password.html",
                              {"msg": "Your Old Password And New Password Are Same.", "email":
email})
            try:
                user = Customer.objects.get(email=email, password=password)
                user.password = new_password
                user.save()
                messages.success(request, 'Password Changed Successfully')
                return redirect('/customer_login')
            except Exception as e:
                print(e)
                return  render(request,  "customer_change_password.html",  {"msg":  "Invalid
Credentials", "email": email})
        return render(request, "customer_change_password.html", {"email": email})
```

```
    else:
        return redirect('/customer_login')


def deactivate(request, id):
    cus = Customer.objects.get(id=id)
    cus.status = 'pending'
    cus.save()
    return redirect('/customer_login')


def admin_change_password(request):
    email = request.session.get("email")
    if not email:
        return redirect('/admin_login')

    if is_login(request):
        if request.method == 'POST':
            password = request.POST['password']
            new_password = request.POST['new_password']
            if password == new_password:
                return render(request, "admin_change_password.html",
                              {"msg": "Your Old Password And New Password Are Same.", "email":
email})
            try:
                user = Admin.objects.get(email=email, password=password)
                user.password = new_password
                user.save()
                messages.success(request, 'Password Changed Successfully')
                return redirect('/admin_login')
            except Exception as e:
                print(e)
                return render(request, "admin_change_password.html", {"msg": "Invalid
Credentials", "email": email})
        return render(request, "admin_change_password.html", {"email": email})
    else:
        return redirect('/admin_login')


def vendor_register_form(request):
    if request.method == "POST":
        form = VendorForm(request.POST, request.FILES)
        email = request.POST['email']
```

```
        if Vendor.objects.filter(email=email).exists():
            return render(request, "vendor_register_form.html",
                          {"msg": "This Email Already Exists Please Try With Another Email"})

        else:
            if form.is_valid():
                print(form.errors)
                form.save()
                return render(request, "vendor_login.html", {"msg": "Registration Successfully
Done"})
            return render(request, "vendor_register_form.html", {"msg": "Not Registration
Successfully Done"})
    return render(request, "vendor_register_form.html", {})


def vendor_login(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        log = Vendor.objects.filter(email=email, password=password)
        if log.exists():
            request.session['email'] = email
            return render(request, "vendor_home.html", {"msg": "Accepted Successfully"})
        return render(request, "vendor_login.html", {"msg": "Login Failed"})
    return render(request, "vendor_login.html", {})


def vendor_home(request):
    return render(request, "vendor_home.html", {})


def vendor_logout(request):
    return redirect('/vendor_login')


def vendor_profile(request):
    email = request.session['email']
    profile = Vendor.objects.get(email=email)
    return render(request, "vendor_profile.html", {"x": profile})


def accept_vendor(request, id):
    vendor = Vendor.objects.get(id=id)
```

```
    vendor.status = 'accepted'
    vendor.save()
    return redirect('/view_vendor')


def reject_vendor(request, id):
    vendor = Vendor.objects.get(id=id)
    vendor.status = 'rejected'
    vendor.save()
    return redirect('/view_vendor')


def vendor_edit(request):
    email = request.session['email']
    vendor = Vendor.objects.get(email=email)
    return render(request, 'vendor_edit.html', {'x': vendor})


def vendor_update(request):
    if request.method == 'POST':
        id = request.POST['id']
        vendor = Vendor.objects.get(id=id)
        form = VendorForm(request.POST, request.FILES, instance=vendor)
        print(form.errors)
        if form.is_valid():
            form.save()
            return redirect('/vendor_profile')
        return render(request, "vendor_edit.html", {})
    return render(request, "vendor_edit.html", {})


def vendor_change_password(request):
    email = request.session.get("email")
    if not email:
        return redirect('/vendor_login')

    if is_login(request):
        if request.method == 'POST':
            password = request.POST['password']
            new_password = request.POST['new_password']
            if password == new_password:
                return render(request, "vendor_change_password.html",
```

```
                    {"msg": "Your Old Password And New Password Are Same.", "email":
email})
            try:
                user = Vendor.objects.get(email=email, password=password)
                user.password = new_password
                user.save()
                messages.success(request, 'Password Changed Successfully')
                return redirect('/vendor_login')
            except Exception as e:
                print(e)
                return render(request, "vendor_change_password.html", {"msg": "Invalid
Credentials", "email": email})
        return render(request, "vendor_change_password.html", {"email": email})
    else:
        return redirect('/vendor_login')


def deactivate_vendor(request, id):
    vendor = Vendor.objects.get(id=id)
    vendor.status = 'pending'
    vendor.save()
    return redirect('/vendor_login')


def vendor_view_customer(request):
    customer = Customer.objects.all()
    return render(request, "vendor_view_customer.html", {"customer": customer})


def add_notification(request):
    if request.method == 'POST':
        form = NotificationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('view_notification')
        return render(request, "add_notification.html", {})
    return render(request, "add_notification.html", {})


def view_notification(request):
    profile = Notification.objects.all()
    return render(request, "view_notification.html", {"profile": profile, })
```

```
def add_category(request):
    if request.method == 'POST':
        form = CategoryForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('view_category')
        return render(request, "add_category.html", {})
    return render(request, "add_category.html", {})


def view_category(request):
    profile = Category.objects.all()
    return render(request, "view_category.html", {"profile": profile, })


def view_vendor(request):
    customer = Vendor.objects.all()
    return render(request, "view_vendor.html", {"customer": customer})


def delete_view_notification(request, id):
    notify = Notification.objects.get(id=id)
    notify.delete()
    return redirect('/add_notification')


def delete_view_category(request, id):
    cat = Category.objects.get(id=id)
    cat.delete()
    return redirect('/add_category')


def add_product(request):
    email = request.session['email']
    category = Category.objects.all()
    if request.method == 'POST':
        form = ProductForm(request.POST, request.FILES)
        print('')
        print(form.errors)
        if form.is_valid():
            form.save()
            return redirect('view_product')
```

```
        return render(request, "add_product.html", {"email": email})
    return render(request, "add_product.html", {"email": email, "category": category})


def view_product(request):
    email = request.session['email']
    profile = Product.objects.filter(email=email)
    return render(request, "view_product.html", {"profile": profile})


def customer_view_product(request):
    profile = Product.objects.all()
    return render(request, "customer_view_product.html", {"profile": profile})


def delete_view_product(request, id):
    cat = Product.objects.get(id=id)
    cat.delete()
    return redirect('/add_product')


def product_edit(request, id):
    edit = Product.objects.get(id=id)
    return render(request, 'product_edit.html', {'x': edit})


def product_update(request):
    if request.method == 'POST':
        id = request.POST['id']
        product = Product.objects.get(id=id)
        form = ProductForm(request.POST, request.FILES, instance=product)
        print(form.errors)
        if form.is_valid():
            form.save()
            return redirect('/view_product')
        return render(request, "product_edit.html", {})
    return render(request, "product_edit.html", {})


def view_cart(request):
    email = request.session['email']
    cate = Cart.objects.filter(customer_email=email)
    tq = 0
```

```
    tc = 0
    td = 0
    fc = 0
    for x in cate:
        tq = tq + int(x.quantity)
        tc = tc + int(x.cost * x.quantity)
        td = td + int(x.discount * x.quantity)
        fc = tc - td
    return render(request, "view_cart.html", {"cate": cate, "td": td, "tc": tc, "tq": tq, "fc": fc})


def delete_view_cart(request, id):
    cart = Cart.objects.get(id=id)
    cart.delete()
    return redirect('/view_cart')


def customer_view_vendors(request):
    vendor = Vendor.objects.all()
    return render(request, "customer_view_vendors.html", {"vendor": vendor})


def customer_view_notification(request):
    profile = Notification.objects.all()
    return render(request, "customer_view_notification.html", {"profile": profile, })


def add_to_cart(request, id):
    email = request.session["email"]
    customer = Customer.objects.get(email=email)
    product = Product.objects.get(id=id)

    if request.method == "POST":
        # Check if the product is already in the cart
        existing_cart_item = Cart.objects.filter(product=product,
customer_email=customer.email).first()

        if existing_cart_item:
            return render(request, "add_to_cart.html", {
                "msg": "This product is already in your cart.",
                'email': email,
                'x': product
            })
```

```
        # Check if there are items in the cart from a different vendor
        cart_items = Cart.objects.filter(customer_email=customer.email)

        if cart_items.exists():
            existing_vendor_email = cart_items.first().vendor_email

            if existing_vendor_email != product.email:
                return render(request, "add_to_cart.html", {
                    "msg": "You can only select products from one vendor at a time. Please complete your current order before adding products from another vendor.",
                    "customer": customer,
                    "product": product,
                    "email": email
                })

        # If all checks pass, add the product to the cart
        form = CartForm(request.POST, request.FILES)
        print(form.errors)
        if form.is_valid():
            form.save()
            return render(request, "add_to_cart.html", {
                'msg': 'Product successfully added to your cart',
                'email': email,
                'x': product
            })
        else:
            print("Form is not valid:", form.errors)

    return render(request, "add_to_cart.html", {
        'email': email,
        'x': product
    })


def checkout(request):
    print("Checkout called")
    cart_items = Cart.objects.filter(customer_email=request.session["email"])

    tq = sum(item.quantity for item in cart_items)
    tc = sum(item.cost * item.quantity for item in cart_items)
    td = sum(item.discount * item.quantity for item in cart_items)
    fc = tc - td
```

```
if request.method == "POST":
    order_form = OrderForm(request.POST)
    print("POST data received:", request.POST)

    if order_form.is_valid():
        order = order_form.save()
        print("Order saved with ID:", order.id)

        for item in cart_items:
            Order_Products.objects.create(
                order=order,
                product=item.product,
                title=item.product.title,
                quantity=item.quantity,
                cost=item.cost,
                discount=item.discount,
                total=item.total,
                customer_email=item.customer_email,
                vendor_email=item.vendor_email
            )

        cart_items.delete()
        return render(request, "checkout.html", {
            "msg": "Your Order Is Success",
            "customer_email": order.customer_email,
            "vendor_email": order.vendor_email,
            "td": td,
            "tc": tc,
            "tq": tq,
            "fc": fc
        })
    else:
        print("Form is not valid:", order_form.errors)
else:
    print("Request method not POST")

return render(request, "checkout.html", {
    "customer_email": cart_items[0].customer_email if cart_items else "",
    "vendor_email": cart_items[0].vendor_email if cart_items else "",
    "td": td,
    "tc": tc,
    "tq": tq,
```

```
        "fc": fc
    })


def my_orders(request):
    email = request.session['email']
    orders = Orders.objects.filter(customer_email=email)
    return render(request, "my_orders.html", {"orders": orders})


def my_orders_products(request, id):
    orders = Orders.objects.get(id=id)
    order_products = Order_Products.objects.filter(order_id=orders.id)
    return render(request, "my_orders_products.html", {"order_products":
order_products})


def customer_view_products(request, id):
    order_products = Order_Products.objects.filter(order_id=id)
    return render(request, "customer_view_products.html", {"order_products":
order_products})


def edit_item(request, id):
    cart = Cart.objects.get(id=id)
    return render(request, "edit_item.html", {"cart": cart})


def edit_item_update(request):
    if request.method == "POST":
        print("error:")
        id = request.POST["id"]
        print("hello")
        user = Cart.objects.get(id=id)
        users = CartForm(request.POST, instance=user)
        print("error:", users.errors)
        if users.is_valid():
            print("error:", users.errors)
            users.save()
        return redirect("/customer_view_cart")
    return redirect("/customer_view_cart")
```

```
def remove_item(request, id):
    cart = Cart.objects.get(id=id)
    cart.delete()
    return redirect("/customer_view_cart")


def customer_checkout(request):
    order = Orders.objects.all()
    return render(request, "customer_checkout.html", {"order": order})

def cancel_order(request, id):
    order = Orders.objects.get(id=id)
    if request.method == "POST":
        print("hii")
        form = OrderForm(request.POST, instance=order)
        print("hii22")
        print(form.errors)
        print("hii3")
        if form.is_valid():
            print(form.errors)
            order.status = 'Cancelled'
            form.save()
            return redirect("/my_orders")
    return render(request, "cancel_order.html", {"order": order})


def customer_cart_products(request, id):
    order_products = Order_Products.objects.filter(order_id=id)
    return render(request, "customer_cart_products.html", {"order_products":
order_products})


def accept_checkout(request, id):
    accept = Orders.objects.get(id=id)
    accept.status = 'Accepted'
    accept.save()
    return redirect('/vendor_checkout')


def reject_checkout(request, id):
    reject = Orders.objects.get(id=id)
    reject.status = 'Rejected'
    reject.save()
```

```
    return redirect('/vendor_checkout')


def delivery_checkout(request, id):
    delivery = Orders.objects.get(id=id)
    delivery.status = 'Delivered'
    delivery.save()
    return redirect('/vendor_checkout')


def vendor_cart_products(request, id):
    order_products = Order_Products.objects.filter(order_id=id)
    return render(request, "vendor_cart_products.html", {"order_products":
order_products})


def vendor_checkout(request):
    order = Orders.objects.all()
    return render(request, "vendor_checkout.html", {"order": order})


def customer_orders(request):
    if request.method == 'POST':
        form = OrderForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('customer_view_cart')
        return render(request, "customer_orders.html", {})
    return render(request, "customer_orders.html", {})


def order_products(request):
    if request.method == 'POST':
        form = Order_ProductsForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('customer_orders')
        return render(request, "order_products.html", {})
    return render(request, "order_products.html", {})


def admin(request):
    if request.method == 'POST':
```

```
        form = AdminForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('admin_login')
        return render(request, "admin.html", {})
    return render(request, "admin.html", {})


def add_preference(request):
    email = request.session["email"]
    category = Category.objects.all()

    if request.method == "POST":
        form = My_PreferenceForm(request.POST)
        print(form.errors)
        if form.is_valid():
            form.save()
            return render(request, "add_preference.html", {"msg": "Submitted Preference
Form"})

        return render(request, "add_preference.html", {"msg": "Not Submitted Preference
Form"})
    return render(request, "add_preference.html", {"email": email, "category": category})


def my_preference(request):
    email = request.session['email']
    preferences = My_Preference.objects.filter(email=email)
    profile = []
    for preference in preferences:
        products = Product.objects.filter(category=preference.category)
        profile.extend(products)

    return render(request, "my_preference.html", {"profile": profile})


def customer_view_cart_edit(request, id):
    cate = Cart.objects.get(id=id)
    return render(request, "customer_view_cart_edit.html", {"cate": cate})


def customer_view_cart_update(request):
    if request.method == "POST":
```

```
        print("error:")
        id = request.POST["id"]
        print("hello")
        user = Cart.objects.get(id=id)
        users = CartForm(request.POST, instance=user)
        print("error:", users.errors)
        if users.is_valid():
            print("error:", users.errors)
            users.save()
            return redirect('/view_cart')
        return render(request, "customer_view_cart.html", {"msg": "not update"})
    return redirect("/customer_view_cart")


def vendor_view_orders(request):
    email = request.session['email']
    orders = Orders.objects.filter(vendor_email=email)
    return render(request, "vendor_view_orders.html", {"orders": orders})


def vendor_view_orders_products(request, id):
    orders = Orders.objects.get(id=id)
    order_products = Order_Products.objects.filter(order_id=orders.id)
    return render(request, "vendor_view_orders_products.html", {"order_products":
order_products})


def accept_order(request, id):
    orders = Orders.objects.get(id=id)
    orders.status = 'Accepted'
    orders.save()
    return redirect('/vendor_view_orders')


def reject_order(request, id):
    orders = Orders.objects.get(id=id)
    orders.status = 'Rejected'
    orders.save()
    return redirect('/vendor_view_orders')


def cancel_order(request, id):
    orders = Orders.objects.get(id=id)
```

```
    orders.status = 'Cancelled'
    orders.save()
    return redirect('//my_orders')


def search(request):
    queryval = request.POST["profile"]
    profile = Product.objects.filter(category__contains=queryval)
    return render(request, "customer_view_product.html", {"profile": profile})


def all_users(request):
    email = request.session['email']
    customers = Customer.objects.exclude(email=email)

    # Sent and received requests
    sent_requests = Friends_Requests.objects.filter(from_email=email)
    received_requests = Friends_Requests.objects.filter(to_email=email)

    # Requested and accepted requests
    requested_ids = list(sent_requests.values_list('to_email', flat=True)) + list(
        received_requests.values_list('from_email', flat=True))

    # Friends list (accepted requests)
    accepted_requests = Friends_Requests.objects.filter(
        (Q(from_email=email) | Q(to_email=email)), status='Accepted'
    ).values_list('from_email', 'to_email')

    friends_ids = [req[0] if req[1] == email else req[1] for req in accepted_requests]

    return render(request, 'all_users.html', {
        "customers": customers,
        "requested_ids": requested_ids,
        "friends_ids": friends_ids
    })


def customer_add_review(request, id):
    email = request.session["email"]
    profile = Product.objects.get(id=id)
    if request.method == "POST":
        form = ReviewForm(request.POST)
        print(form.errors)
```

```
        if form.is_valid():
            form.save()
            return render(request, "customer_add_review.html", {'msg': 'Review Added Successfully '})
        return render(request, "customer_add_review.html", {})
    return render(request, 'customer_add_review.html', {"email": email, "profile": profile})


def customer_view_review(request, id):
    profile = Product.objects.get(id=id)
    reviews = Review.objects.filter(orders_id=profile.id)
    return render(request, 'customer_view_review.html', {"reviews": reviews})



def add_friends_requests(request, id):
    if request.method == "POST":
        email = request.session['email']
        customer = get_object_or_404(Customer, id=id)

        # Check if request already exists
        if Friends_Requests.objects.filter(from_email=email, to_email=customer.email).exists():
            return JsonResponse({"msg": "Request Already Sent"}, status=400)

        # Create friend request
        Friends_Requests.objects.create(
            customers=customer,
            from_email=email,
            to_email=customer.email
        )
        return JsonResponse({"msg": "Friend Request Sent"}, status=200)

    return JsonResponse({"msg": "Invalid Request"}, status=400)


def view_friends_requests(request):
    email = request.session['email']
    requests = Friends_Requests.objects.filter(to_email=email)
    return render(request,'view_friends_requests.html',{"requests":requests})
```

```
def accept_request(request,id):
    requests = Friends_Requests.objects.get(id=id)
    requests.status='Accepted'
    requests.save()
    return redirect("/view_friends_requests")

def reject_request(request, id):
    requests = Friends_Requests.objects.get(id=id)
    requests.status='Rejected'
    requests.save()
    return redirect("/view_friends_requests")

def my_friends(request):
    email = request.session['email']
    sent_friends = Friends_Requests.objects.filter(from_email=email, status='Accepted')
    received_friends                                                                    =
Friends_Requests.objects.filter(to_email=email,status='Accepted')

    return render(request, "my_friends.html",{
        "sent_friends": sent_friends,
        "received_friends": received_friends
    })



def post(request):
    email = request.session['email']
    if request.method == "POST":
        form = PostForm(request.POST,request.FILES)
        print(form.errors)
        if form.is_valid():
            form.save()
            return render(request, "post.html", {"msg": "Thanks For Posting Us"})
        return render(request, "post.html", {"msg": "While Posting Us Your Getting Error"})
    return render(request, "post.html", {"email":email})




def view_post(request):
    email = request.session['email']
    friends_pairs = Friends_Requests.objects.filter(
```

```
        (Q(from_email=email) | Q(to_email=email)),
        status='Accepted'
    ).values_list('from_email','to_email')

    friends = set(chain(*friends_pairs)) - {email}

    post = Post.objects.filter(
        Q(posts_for='all') |
        (Q(posts_for = 'friends') & Q(email__in=friends)),
        status='Accepted'
    ).exclude(email=email).annotate(like_count=Count('likes'))

    liked_posts                                                                    =
Likes.objects.filter(customer_email=email).values_list('post_id',flat=True)
    return render(request, "view_post.html", {"post":post,"liked_posts":liked_posts})


def view_all_posts(request):
    email = request.session['email']
    post = Post.objects.filter(email=email)
    return render(request, "view_all_posts.html", {"post":post})

def admin_view_all_posts(request):
    post = Post.objects.all()
    return render(request, "admin_view_all_posts.html", {"post":post})


def accept_all_posts(request,id):
    requests = Post.objects.get(id=id)
    requests.status='Accepted'
    requests.save()
    return redirect("/admin_view_all_posts")



def reject_all_posts(request, id):
    requests = Post.objects.get(id=id)
    requests.status='Rejected'
    requests.save()
    return redirect("/admin_view_all_posts")

def like_post(request, post_id):
    email = request.session['email']
```

```
    post = get_object_or_404(Post, id=post_id)

    if not Likes.objects.filter(post=post, customer_email=email).exists():
        Likes.objects.create(post=post, customer_email=email,
date_time=timezone.now())
        msg = "Liked Successfully"
    else:
        msg = "You've Already Likes This Post"
    return redirect('/view_post')



def comment(request, id):
    email = request.session["email"]
    profile = Post.objects.get(id=id)
    comments = Comment.objects.filter(post_id_id=profile.id)
    if request.method == "POST":
        form = CommentForm(request.POST)
        print(form.errors)
        if form.is_valid():
            form.save()
            return render(request, "comment.html", {'msg': 'Comment Added Successfully
','comments":comments})
        return render(request, "comment.html", {"comments":comments})
    return render(request, 'comment.html', {"email": email, "profile":
profile,"comments":comments})



URLS:
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path
from pythonfratelloapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.index,name="index"),
    path('doctor',views.doctor,name="doctor"),
    path('contact', views.contact, name="contact"),
    path('departments', views.departments, name="departments"),
    path('department', views.department_single, name="department"),
    path('blog', views.blog, name="blog"),
```

```
    path('blogs', views.blog_single, name="blogs"),
    path('about', views.about, name="about"),
    path('customer_login',views.customer_login, name="customer_login"),

path('customer_register_form',views.customer_register_form,name="customer_registe
r_form"),
    path('customer_home',views.customer_home,name="customer_home"),
    path('customer_profile',views.customer_profile,name="customer_profile"),
    path('customer_logout',views.customer_logout,name="customer_logout"),
    path('admin_login',views.admin_login,name="admin_login"),
    path('admin_home',views.admin_home,name="admin_home"),
    path('admin_Logout',views.admin_Logout,name="admin_Logout"),
    path('admin_viewcontact',views.admin_viewcontact,name="admin_viewcontact"),

path('admin_viewcustomer',views.admin_viewcustomer,name="admin_viewcustomer")
,
    path('accept_customer/<int:id>',views.accept_customer,name="accept_customer"),
    path('reject_customer/<int:id>', views.reject_customer, name="reject_customer"),
    path('customer_edit',views.customer_edit,name="customer_edit"),
    path('update',views.update,name="update"),

path('customer_change_password',views.customer_change_password,name="custom
er_change_password"),
    path('deactivate/<int:id>', views.deactivate, name="deactivate"),
    path('admin_change_password',                views.admin_change_password,
name="admin_change_password"),
    path('vendor_login', views.vendor_login, name="vendor_login"),
    path('vendor_register_form',                      views.vendor_register_form,
name="vendor_register_form"),
    path('vendor_home', views.vendor_home, name="vendor_home"),
    path('vendor_profile', views.vendor_profile, name="vendor_profile"),
    path('vendor_logout', views.vendor_logout, name="vendor_logout"),

path('vendor_view_customer',views.vendor_view_customer,name="vendor_view_cust
omer"),
    path('accept_vendor/<int:id>',views.accept_vendor,name="accept_vendor"),
    path('reject_vendor/<int:id>', views.reject_vendor, name="reject_vendor"),
    path('vendor_edit',views.vendor_edit,name="vendor_edit"),
    path('vendor_update',views.vendor_update,name="vendor_update"),

path('vendor_change_password',views.vendor_change_password,name="vendor_cha
nge_password"),
    path('deactivate_vendor/<int:id>',                views.deactivate_vendor,
name="deactivate_vendor"),
```

```
    path('add_notification',views.add_notification,name="add_notification"),
    path('view_notification', views.view_notification, name="view_notification"),
    path('add_category/', views.add_category, name="add_category"),
    path('view_category', views.view_category, name="view_category"),
    path('view_vendor', views.view_vendor, name="view_vendor"),
    path('delete_view_notification/<int:id>',views.delete_view_notification,
name="delete_view_notification"),
    path('delete_view_category/<int:id>',                views.delete_view_category,
name="delete_view_category"),
    path('add_product', views.add_product, name="add_product"),
    path('view_product', views.view_product, name="view_product"),
    path('customer_view_product',                views.customer_view_product,
name="customer_view_product"),
    path('delete_view_product/<int:id>',                 views.delete_view_product,
name="delete_view_product"),
    path('product_edit/<int:id>', views.product_edit, name="product_edit"),
    path('product_update', views.product_update, name="product_update"),
    path('add_to_cart/<int:id>', views.add_to_cart, name="add_to_cart"),
    path('view_cart', views.view_cart, name="view_cart"),
    path('delete_view_cart/<int:id>', views.delete_view_cart, name="delete_view_cart"),
    path('customer_view_vendors',                views.customer_view_vendors,
name="customer_view_vendors"),
    path('customer_view_notification',           views.customer_view_notification,
name="customer_view_notification"),
    #                path('customer_view_cart',                views.customer_view_cart,
name="customer_view_cart"),
    path('checkout', views.checkout, name="checkout"),
    path('edit_item', views.edit_item, name="edit_item"),
    path('edit_item_update', views.edit_item_update, name="edit_item_update"),
    path('remove_item', views.remove_item, name="remove_item"),
    path('cancel_order', views.cancel_order, name="cancel_order"),
    path('customer_checkout', views.customer_checkout, name="customer_checkout"),
    path('customer_cart_products',               views.customer_cart_products,
name="customer_cart_products"),
    path('accept_checkout', views.accept_checkout, name="accept_checkout"),
    path('reject_checkout', views.reject_checkout, name="reject_checkout"),
    path('delivery_checkout', views.delivery_checkout, name="delivery_checkout"),
    path('vendor_cart_products',                 views.vendor_cart_products,
name="vendor_cart_products"),
    path('vendor_checkout', views.vendor_checkout, name="vendor_checkout"),
    path('customer_orders', views.customer_orders, name="customer_orders"),
    path('order_products', views.order_products, name="order_products"),
    path('admin', views.admin, name="admin"),
    path('add_preference', views.add_preference, name="add_preference"),
```

```
    path('my_preference', views.my_preference, name="my_preference"),
    path('customer_view_cart_edit/<int:id>',                views.customer_view_cart_edit,
name="customer_view_cart_edit"),
    path('customer_view_cart_update',                    views.customer_view_cart_update,
name="customer_view_cart_update"),
    path('my_orders', views.my_orders, name="my_orders"),
    path('my_orders_products/<int:id>',                          views.my_orders_products,
name="my_orders_products"),
    path('vendor_view_orders',                                   views.vendor_view_orders,
name="vendor_view_orders"),
    path('vendor_view_orders_products/<int:id>',    views.vendor_view_orders_products,
name="vendor_view_orders_products"),
    path('accept_order/<int:id>', views.accept_order, name="accept_order"),
    path('reject_order/<int:id>', views.reject_order, name="reject_order"),
    path('cancel_order/<int:id>', views.cancel_order, name="cancel_order"),
    path('search', views.search, name="search"),
    path('all_users', views.all_users, name="all_users"),
    path('customer_add_review/<int:id>',                         views.customer_add_review,
name="customer_add_review"),
    path('customer_view_review/<int:id>',                       views.customer_view_review,
name="customer_view_review"),
    path('add_friends_requests/<int:id>',                        views.add_friends_requests,
name='add_friends_requests'),
    path('view_friends_requests',                                views.view_friends_requests,
name="view_friends_requests"),
    path('accept_request/<int:id>', views.accept_request, name="accept_request"),
    path('reject_request/<int:id>', views. reject_request, name="reject_request"),
    path('my_friends', views.my_friends, name="my_friends"),
    path('post', views.post, name="post"),
    path('view_post', views.view_post, name="view_post"),
    path('view_all_posts', views.view_all_posts, name="view_all_posts"),
    path('admin_view_all_posts',                                 views.admin_view_all_posts,
name="admin_view_all_posts"),
    path('accept_all_posts/<int:id>', views.accept_all_posts, name="accept_all_posts"),
    path('reject_all_posts/<int:id>', views.reject_all_posts, name="reject_all_posts"),

    path('like_post/<int:post_id>', views.like_post, name='like_post'),
    path('comment/<int:id>', views.comment, name="comment")

]


if settings.DEBUG:  # new
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

FORMS:

```
from django import forms
from .models import Contact,Customer,Vendor,Notification,Category,Product,Cart,Orders,Order_Products, Admin, My_Preference,Review,Friends_Requests,Post,Likes,Comment
class ContactForm(forms.ModelForm):
    class Meta:
        model = Contact
        fields = "_all_"
class CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = "_all_"
        exclude=['status']
class VendorForm(forms.ModelForm):
    class Meta:
        model = Vendor
        fields = "_all_"
        exclude=['status']
class NotificationForm(forms.ModelForm):
    class Meta:
        model = Notification
        fields = "_all_"
class CategoryForm(forms.ModelForm):
    class Meta:
        model = Category
        fields= "_all_"
class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = "_all_"
class CartForm(forms.ModelForm):
    class Meta:
        model = Cart
        fields = "_all_"
class OrderForm(forms.ModelForm):
    class Meta:
        model = Orders
        exclude = ["status"]
```

```
class Order_ProductsForm(forms.ModelForm):
    class Meta:
        model = Order_Products
        fields = "_all_"
class AdminForm(forms.ModelForm):
    class Meta:
        model = Admin
        fields = "_all_"


class My_PreferenceForm(forms.ModelForm):
    class Meta:
        model = My_Preference
        fields = "_all_"


class ReviewForm(forms.ModelForm):
    class Meta:
        model =Review
        fields = "_all_"


class Friends_RequestsForm(forms.ModelForm):
    class Meta:
        model =Friends_Requests
        exclude = ["status"]



class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        exclude = ["status"]


class LikesForm(forms.ModelForm):
    class Meta:
        model = Likes
        fields = "_all_"

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
```

```
        fields = "_all_"

MODELS:
from django.contrib.auth.models import User
from django.db import models
from django.utils import timezone


# Create your models here.
class Contact(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()
    subject = models.CharField(max_length=40)
    message = models.TextField()

    class Meta:
        db_table = "Contact"


class Admin(models.Model):
    email = models.EmailField()
    password = models.CharField(max_length=100)


class Customer(models.Model):
    name = models.CharField(max_length=60)
    email = models.EmailField()
    mandal = models.CharField(max_length=50)
    district = models.CharField(max_length=50)
    state = models.CharField(max_length=40)
    age = models.IntegerField()
    gender = models.CharField(max_length=40)
    image = models.FileField()
    password = models.CharField(max_length=20)
    status = models.CharField(max_length=100, default='pending')


class Vendor(models.Model):
    name = models.CharField(max_length=60)
    email = models.EmailField()
    mandal = models.CharField(max_length=50)
    district = models.CharField(max_length=50)
    state = models.CharField(max_length=40)
```

```
    age = models.IntegerField()
    gender = models.CharField(max_length=40)
    image = models.FileField()
    password = models.CharField(max_length=20)
    status = models.CharField(max_length=100, default='pending')

    def _str_(self):
        return self.name


class Notification(models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField()
    date_time = models.DateTimeField(auto_now=True)


class Category(models.Model):
    title = models.CharField(max_length=100)


class Product(models.Model):
    email = models.EmailField()
    category = models.CharField(max_length=100)
    title = models.CharField(max_length=40)
    orign = models.CharField(max_length=80)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.FileField()
    description = models.TextField()
    brand = models.CharField(max_length=30)
    model = models.CharField(max_length=30)
    discount = models.IntegerField()
    specification = models.CharField(max_length=40)
    warranty = models.CharField(max_length=40)

    def _str_(self):
        return self.name


class Cart(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    quantity = models.BigIntegerField()
    cost = models.BigIntegerField()
```

```
    discount = models.BigIntegerField()
    total = models.BigIntegerField()
    customer_email = models.EmailField()
    vendor_email = models.EmailField()

    class Meta:
        db_table = 'Cart'


class Orders(models.Model):
    customer_email = models.EmailField()
    vendor_email = models.EmailField()
    quantity = models.BigIntegerField()
    cost = models.BigIntegerField()
    discount = models.BigIntegerField()
    total = models.BigIntegerField()
    address = models.TextField()
    delivery_date = models.DateField()
    order_datetime = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=100, default="Pending")


class Order_Products(models.Model):
    order = models.ForeignKey(Orders, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    quantity = models.BigIntegerField()
    cost = models.BigIntegerField()
    discount = models.BigIntegerField()
    total = models.BigIntegerField()
    customer_email = models.EmailField()
    vendor_email = models.EmailField()


class My_Preference(models.Model):
    email = models.EmailField()
    category = models.CharField(max_length=100)


class Review(models.Model):
    customer_email = models.EmailField()
    orders = models.ForeignKey(Product, on_delete=models.CASCADE)
    reviews = models.CharField(max_length=100)
```

```
    date_time = models.DateTimeField(auto_now=True)
    ratings = models.BigIntegerField()

    class Meta:
        db_table = "Review"


class Friends_Requests(models.Model):
    customers = models.ForeignKey(Customer, on_delete=models.CASCADE)
    from_email = models.EmailField()
    to_email = models.EmailField()
    status = models.CharField(max_length=100, default="Pending")

    class Meta:
        db_table = "Friends_Requests"
        unique_together = ('from_email', 'to_email')


class Post(models.Model):
    posts_for = models.CharField(max_length=100)
    email = models.EmailField()
    title = models.CharField(max_length=100)
    description = models.TextField()
    date_time = models.DateTimeField(auto_now=True)
    file = models.FileField()
    status = models.CharField(max_length=100, default="Pending")

    class Meta:
        db_table = "Post"


class Likes(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    customer_email = models.EmailField()
    date_time = models.DateTimeField(default=timezone.now)

    class Meta:
        unique_together = ('post', 'customer_email')


class Review(models.Model):
    customer_email = models.EmailField()
    orders = models.ForeignKey(Product, on_delete=models.CASCADE)
```

```
    reviews = models.CharField(max_length=100)
    date_time = models.DateTimeField(auto_now=True)
    ratings = models.BigIntegerField()

    class Meta:
        db_table = "Review"


class Comment(models.Model):
    customer_email = models.EmailField()
    post_id = models.ForeignKey(Post,on_delete=models.CASCADE)
    comment = models.CharField(max_length=100)
    date_time = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = "Comment"
```

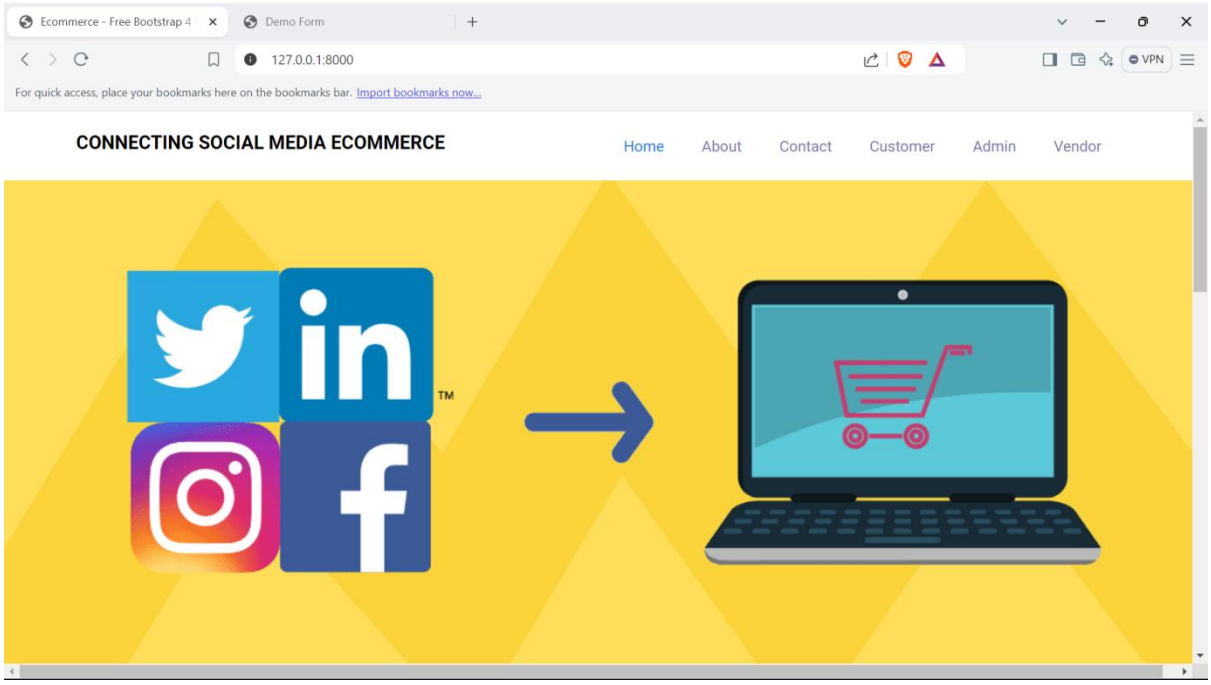
CUSTOMER LOGIN:

```
{% load static %}
{% include "header.html" %}

    <div class="hero-wrap" style="background-image: url({% static 'images/S8.jpg' %});
background-attachment:fixed;">
      <div class="overlay"></div>
      <div class="container">
        <div class="row no-gutters slider-text align-items-center justify-content-center"
data-scrollax-parent="true">
          <div class="col-md-8 ftco-animate text-center">
            <h1 class="mb-3 bread"> Login Here</h1>
          </div>
        </div>
      </div>
    </div>

   <section class="ftco-section contact-section ftco-degree-bg">
      <div class="container">
        <div class="row d-flex mb-5 contact-info">
          <div class="col-md-12 mb-4">


        <div class="row block-9">
          <div class="col-md-6 pr-md-5">
```

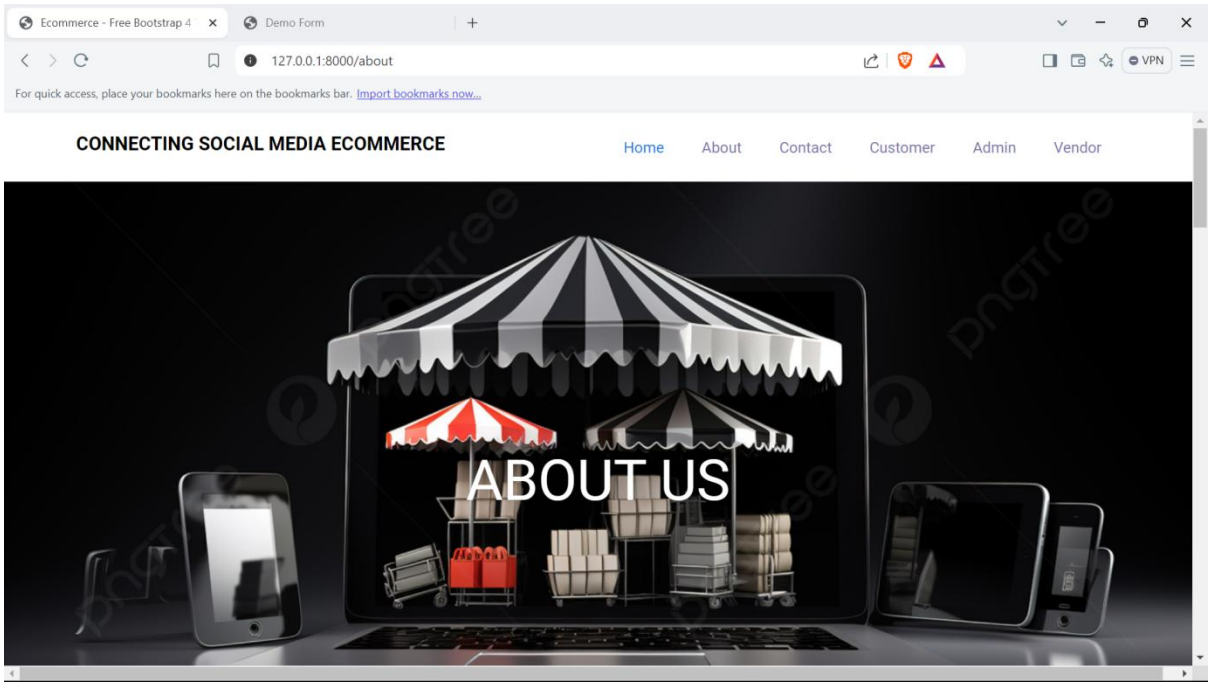
```
          <form method="post">
            {% csrf_token %}

            <div class="form-group">
              <input type="email" class="form-control" placeholder="Your Email"
name="email" required>
            </div>
            <div class="form-group">
              <input type="password" class="form-control" placeholder="Your Password"
name="password" required>
            </div>

            <div class="form-group">
              <input type="submit" value="Login" class="btn btn-primary py-3 px-5"
placeholder="Login" required>
            </div>
            <p><b>NOTE:If You Are Not Registered?</b></p>
            <a href="{% url 'customer_register_form' %}">Register Here</a>
<b style="color:red">
      {{msg}}
     {% if messages %}
      {% for message in messages %}
        {{ message }}
      {% endfor %}
     {% endif%}
          </b>
           </form>

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



    {% include "footer.html" %}
```

SCREEN SHORTS:

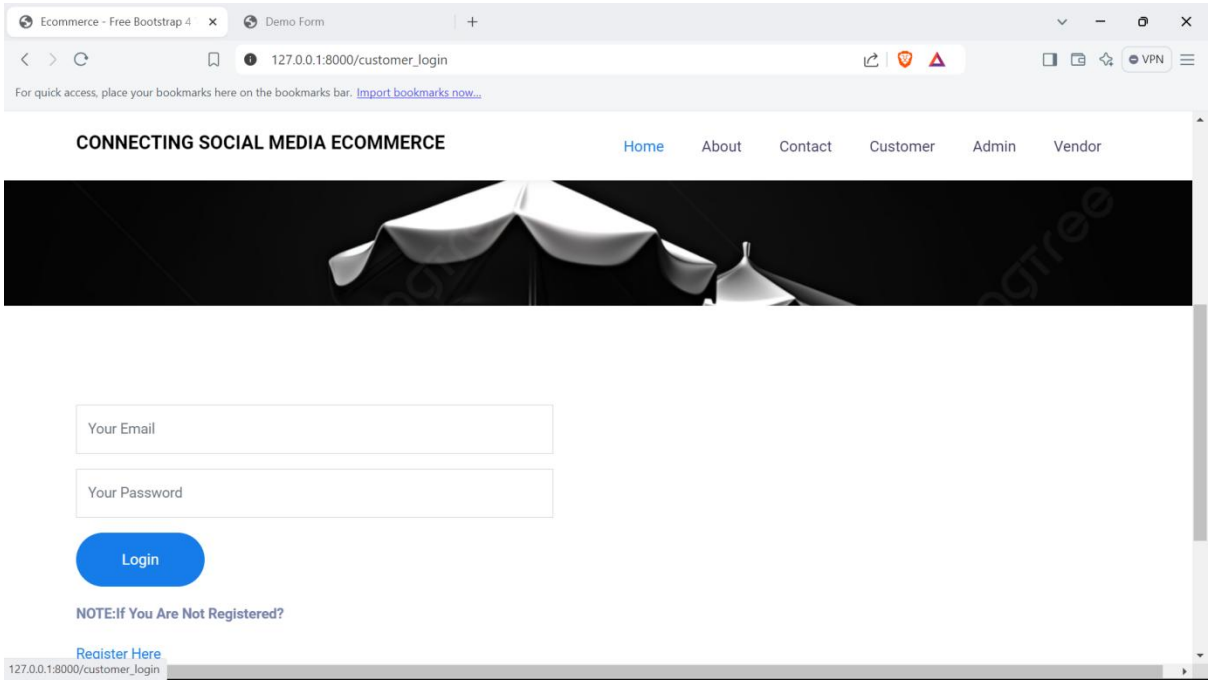
Home:



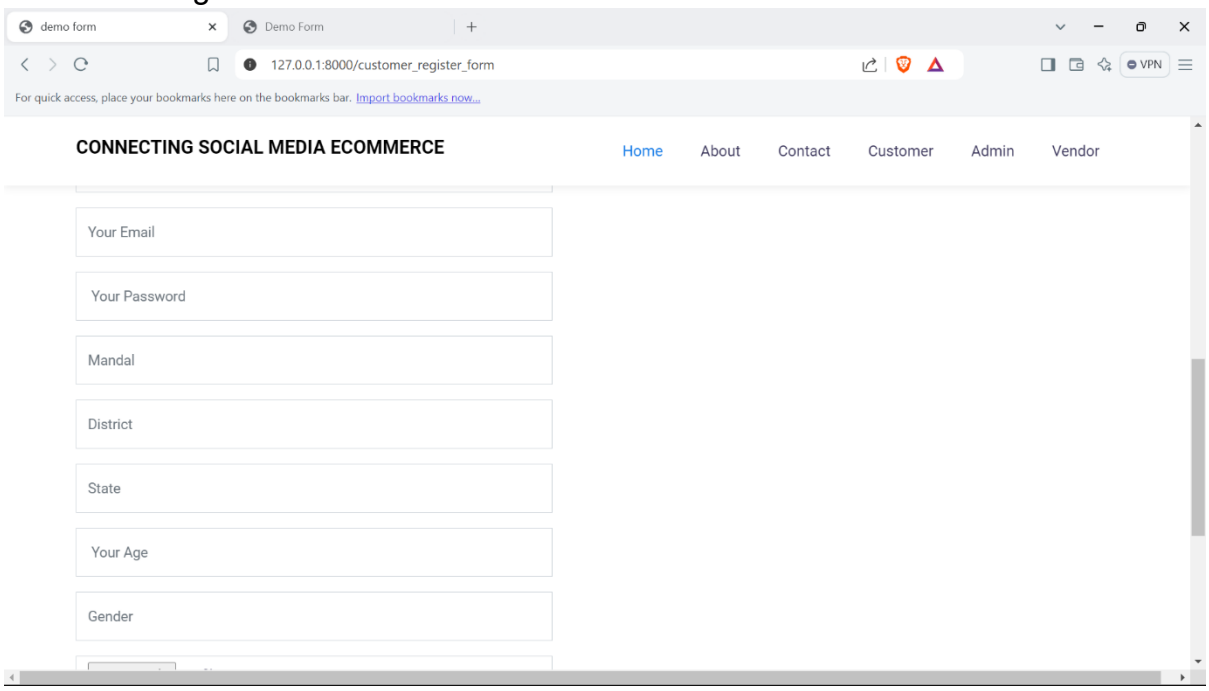
About:



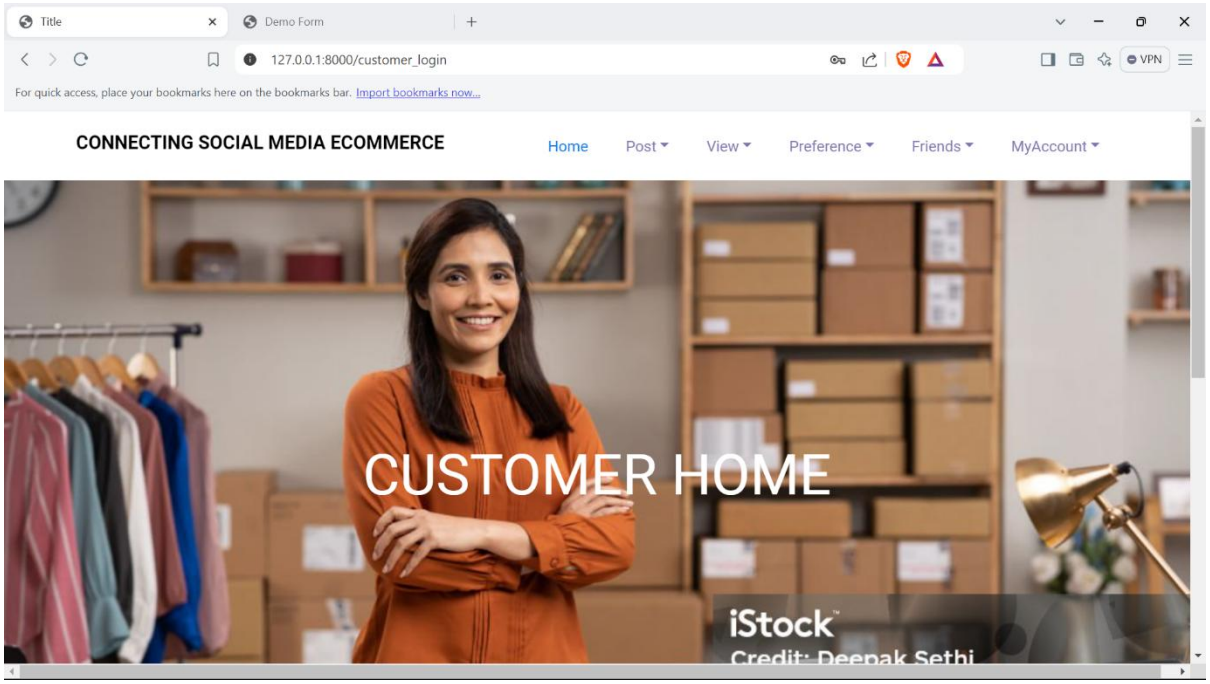
Customer Login:



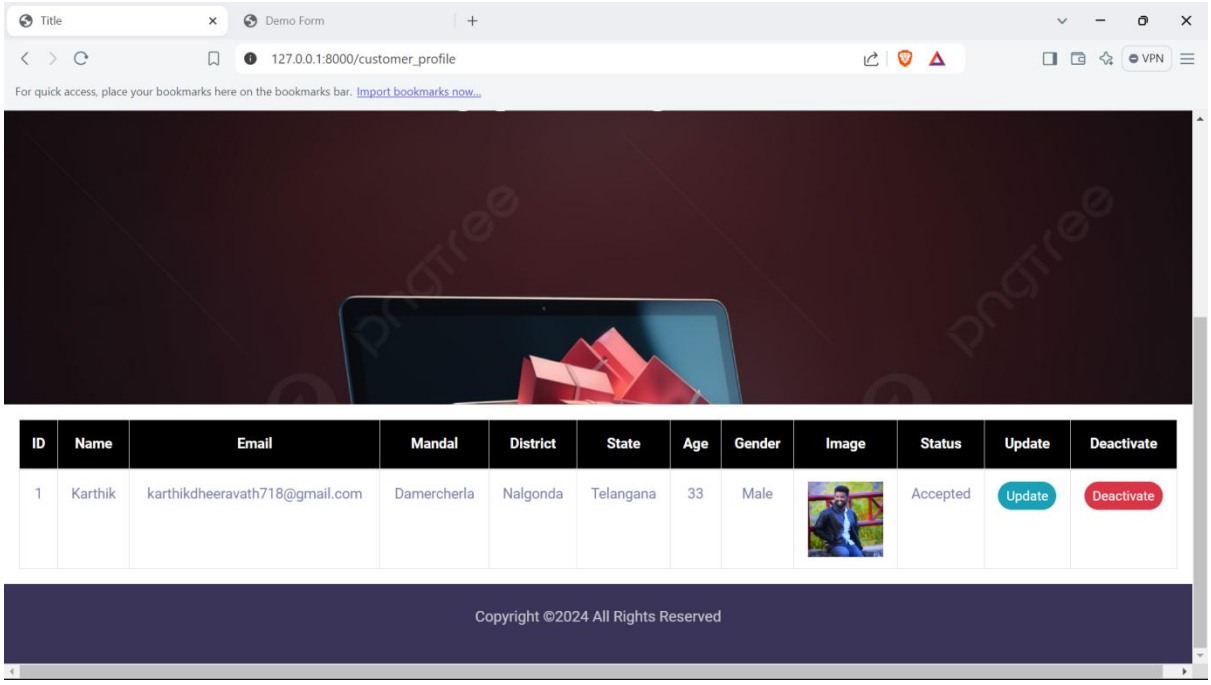
Customer Registration:



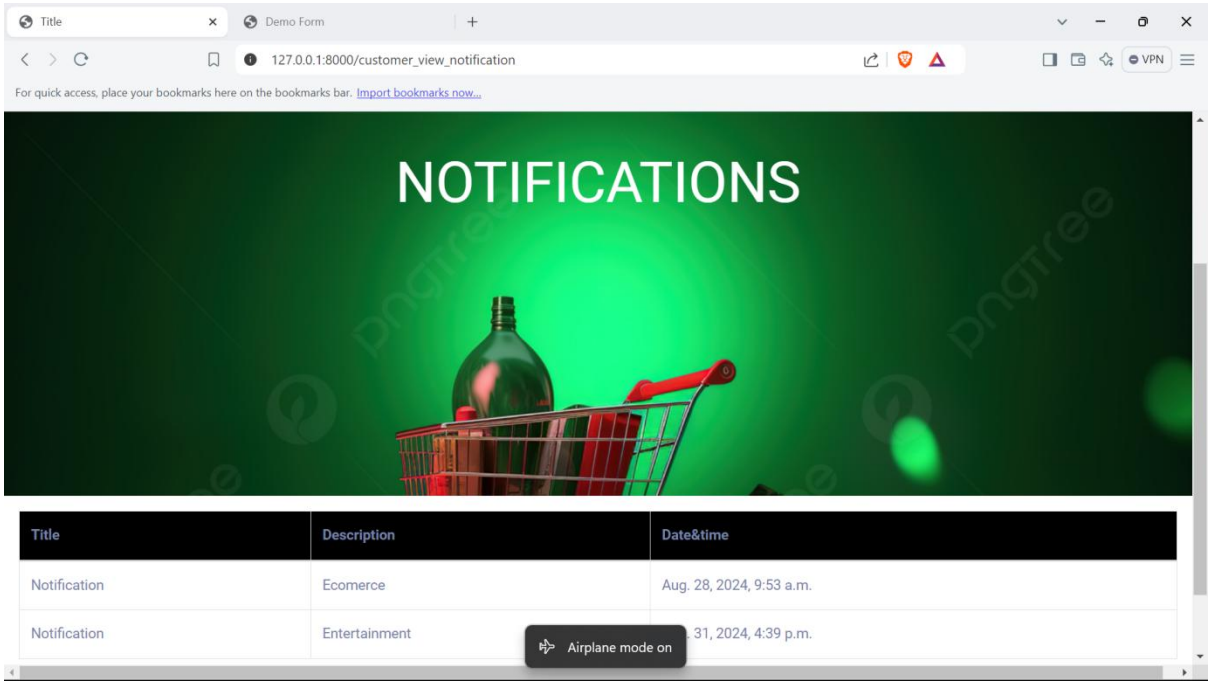
Customer Home:



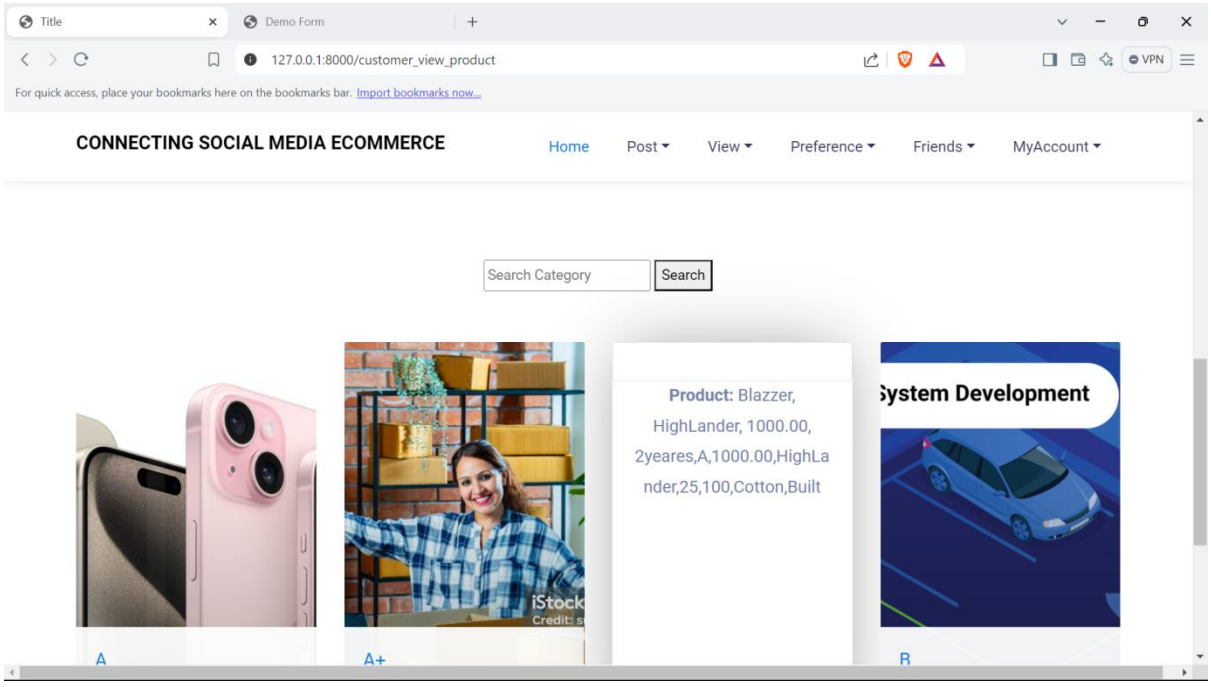
Your Profile:



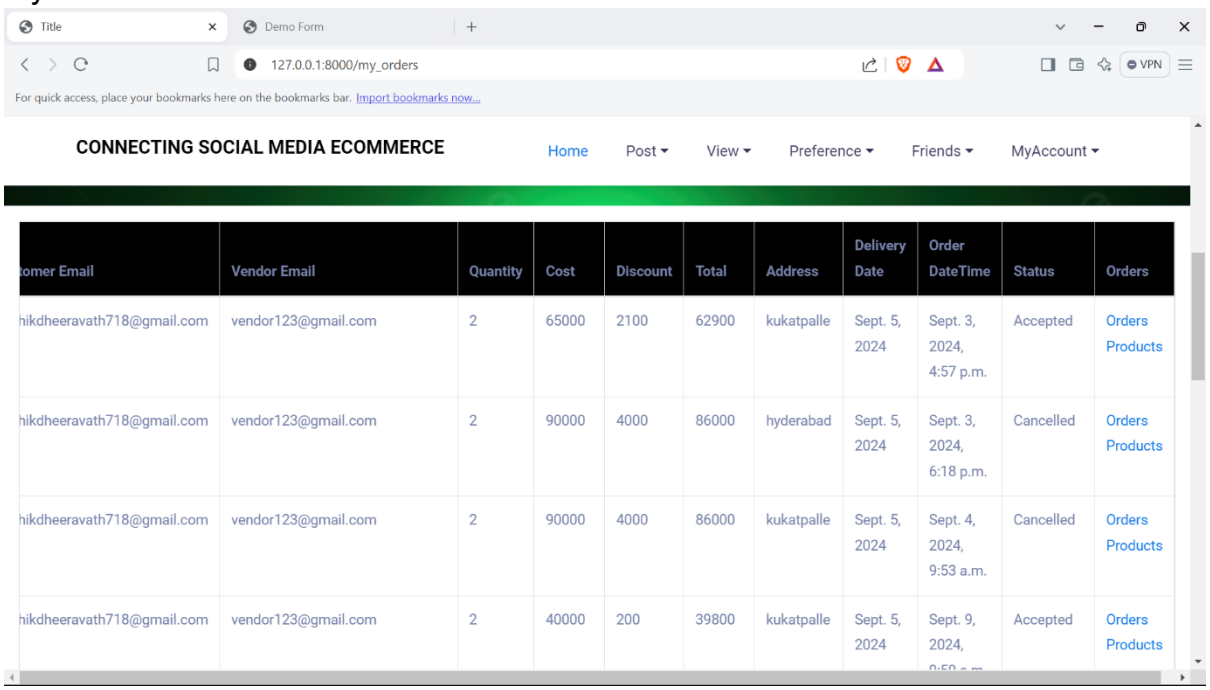
View Notifications:



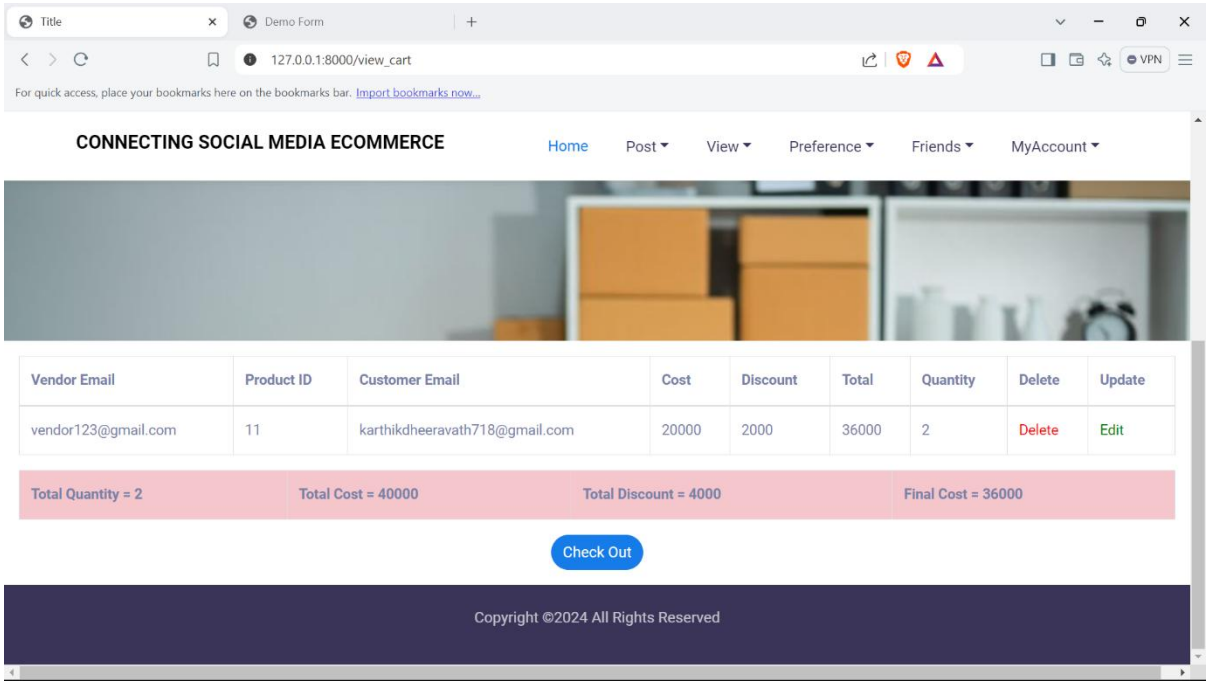
Customer View Products:



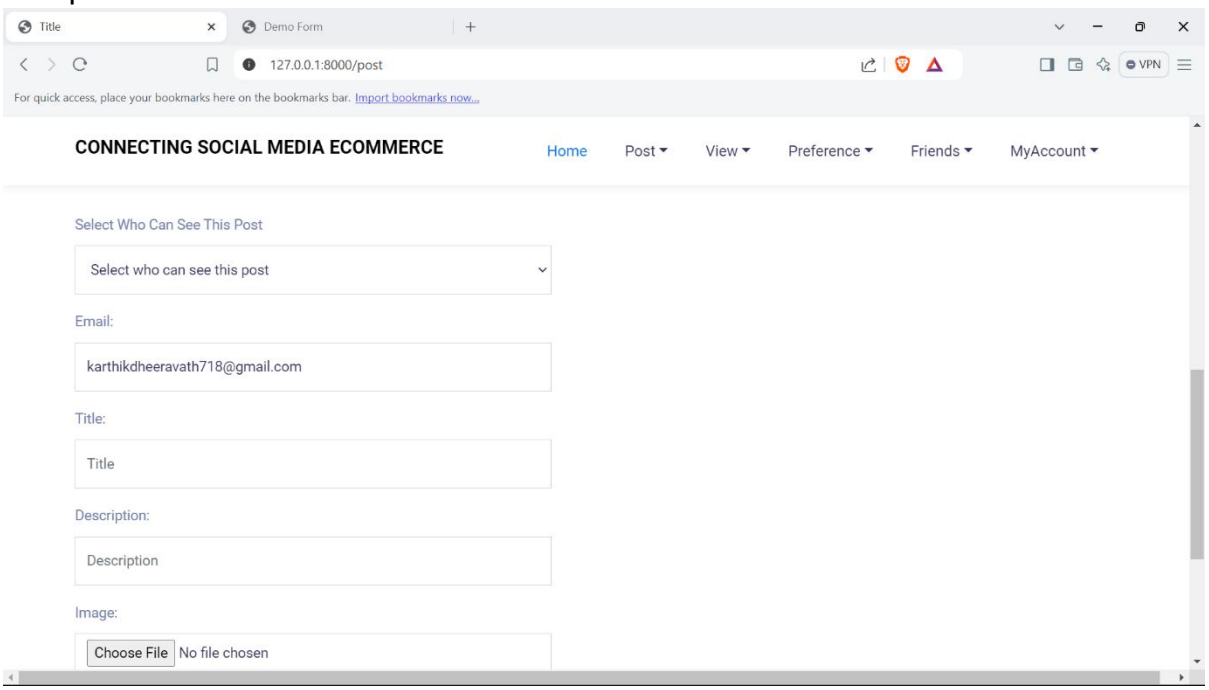
My Orders:



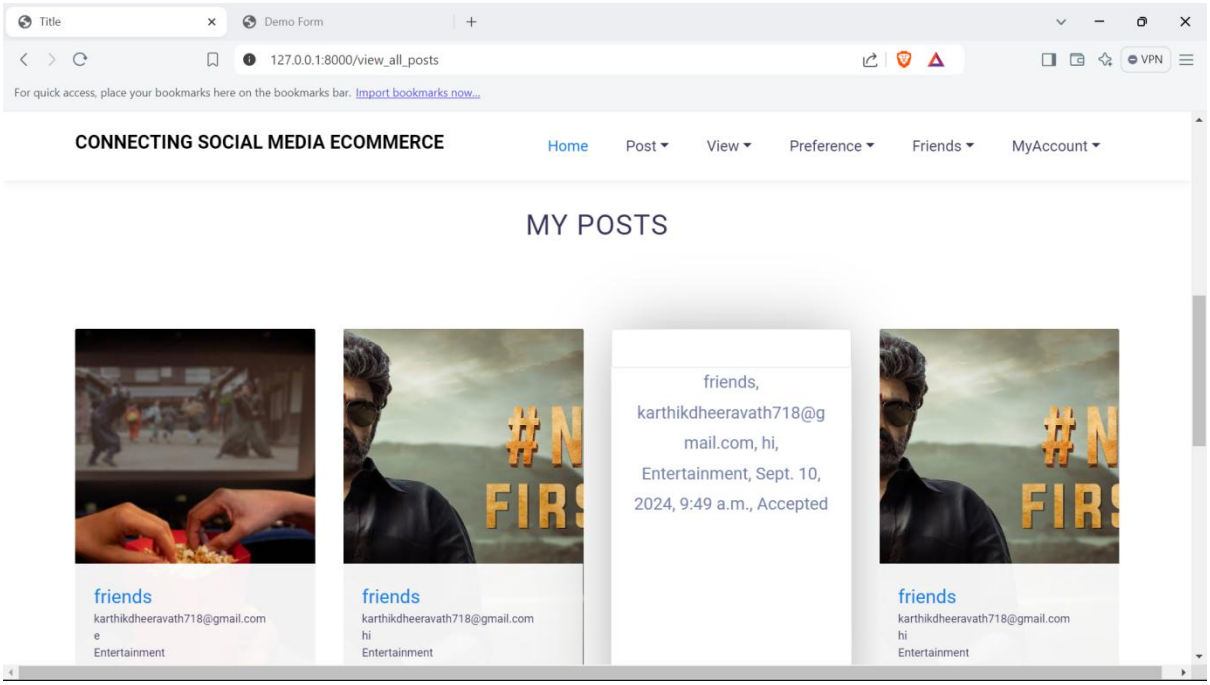
My Cart:



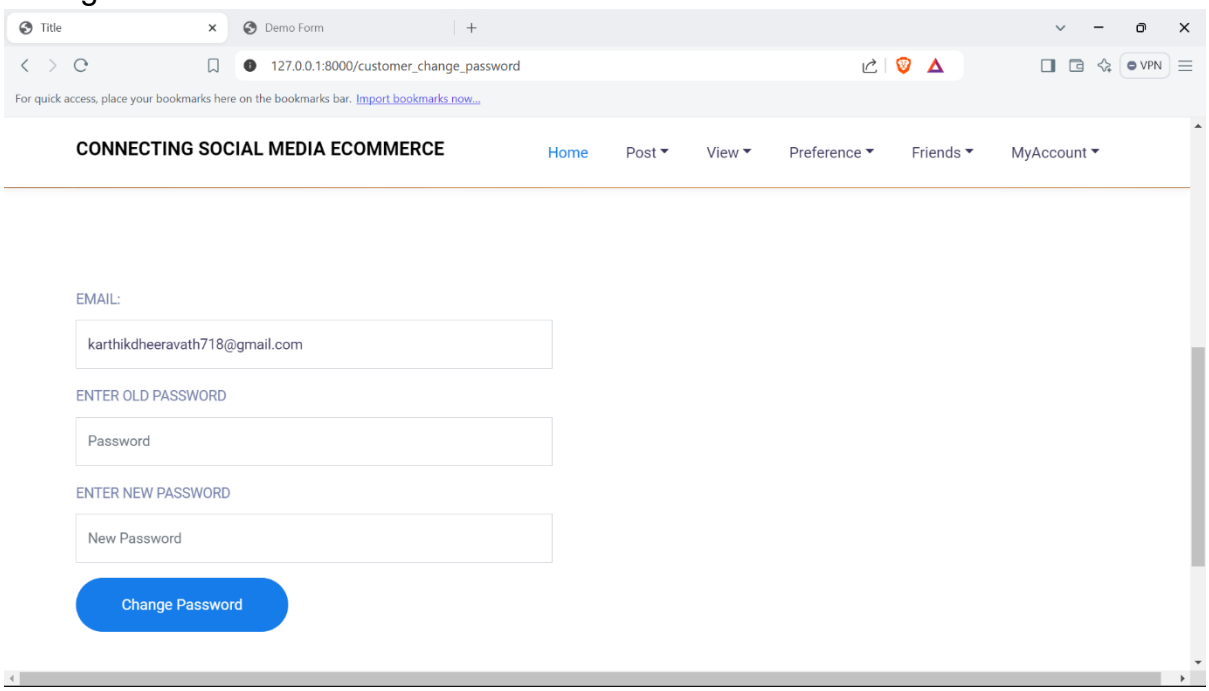
Add post:



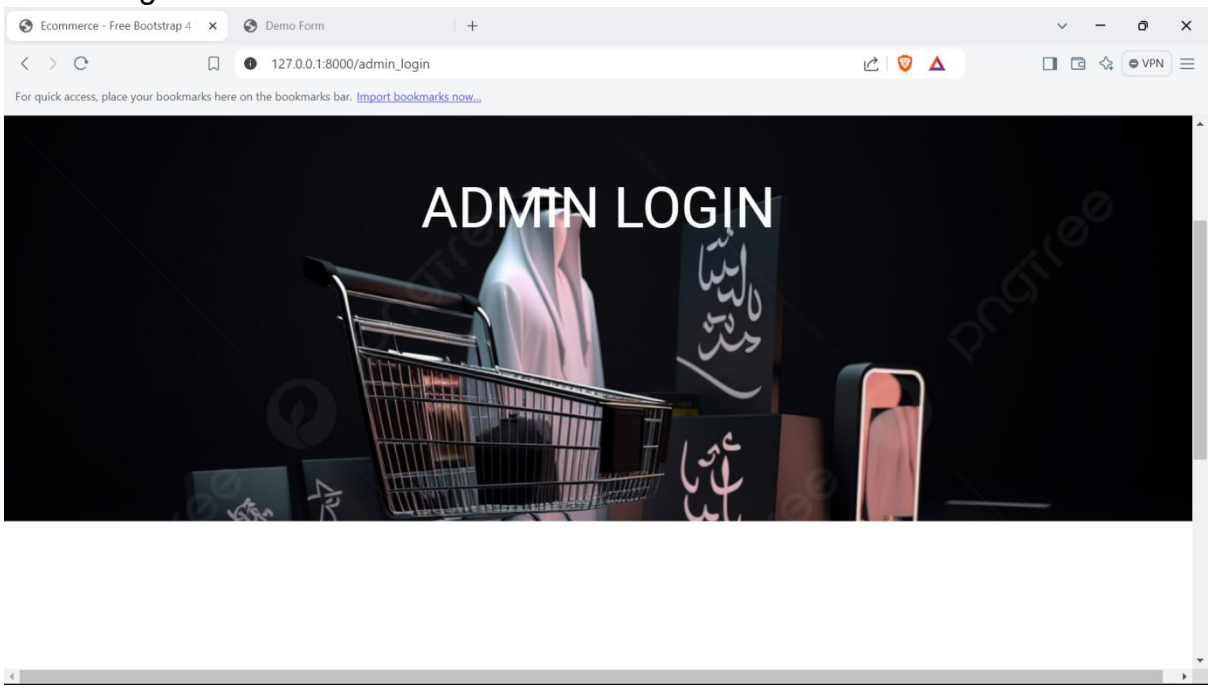
View My Post:



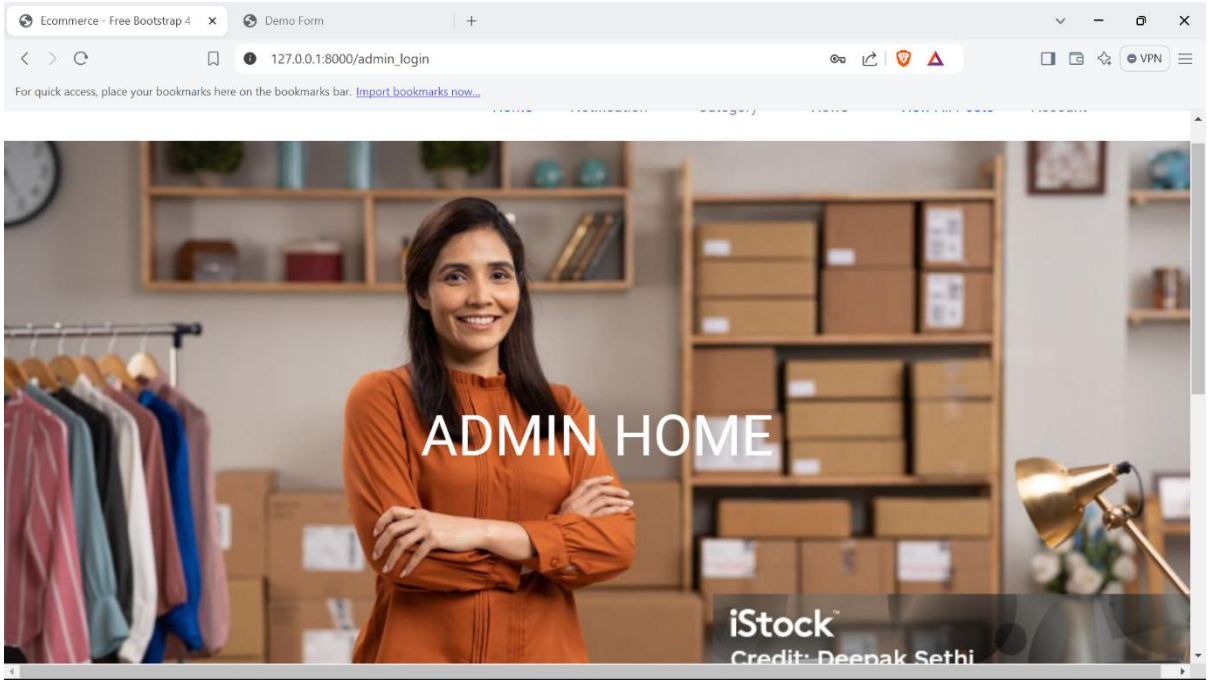
Change Password:



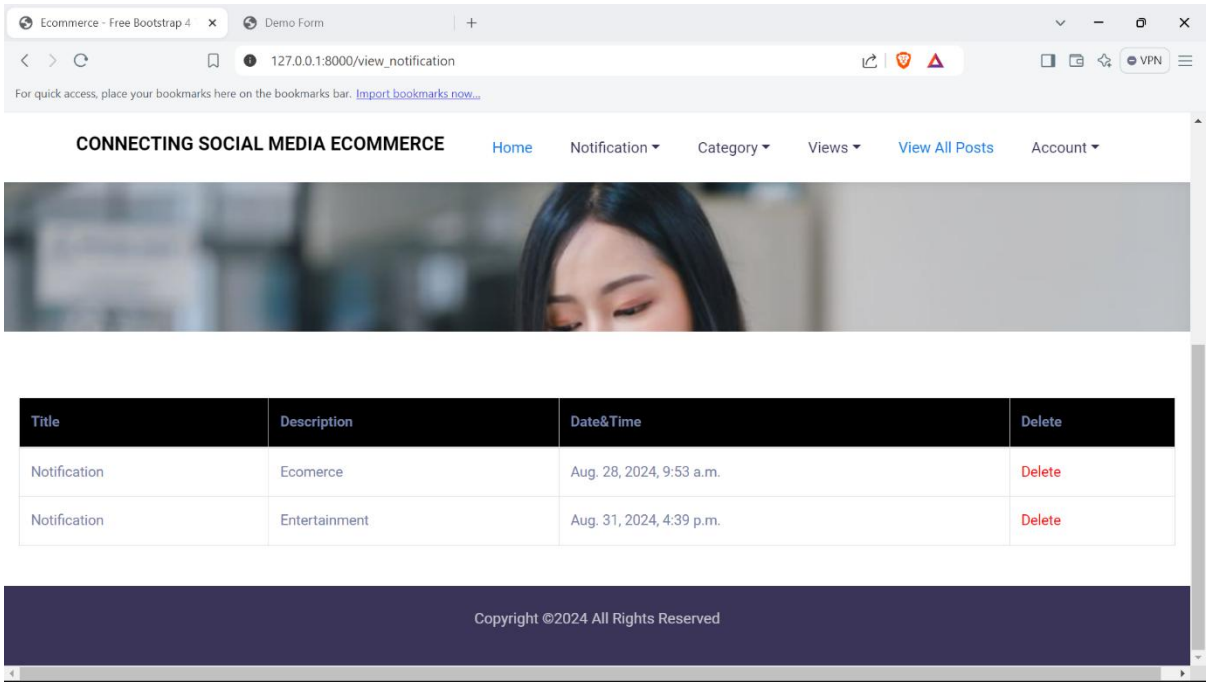
Admin Login:



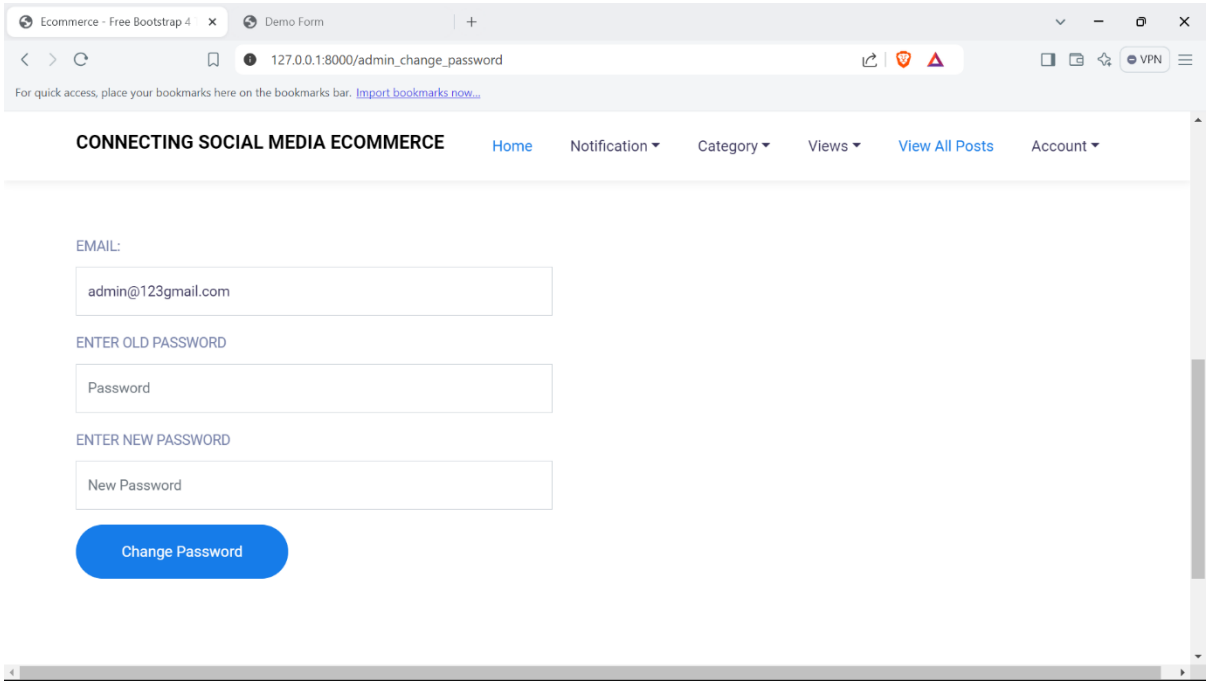
Admin Home:



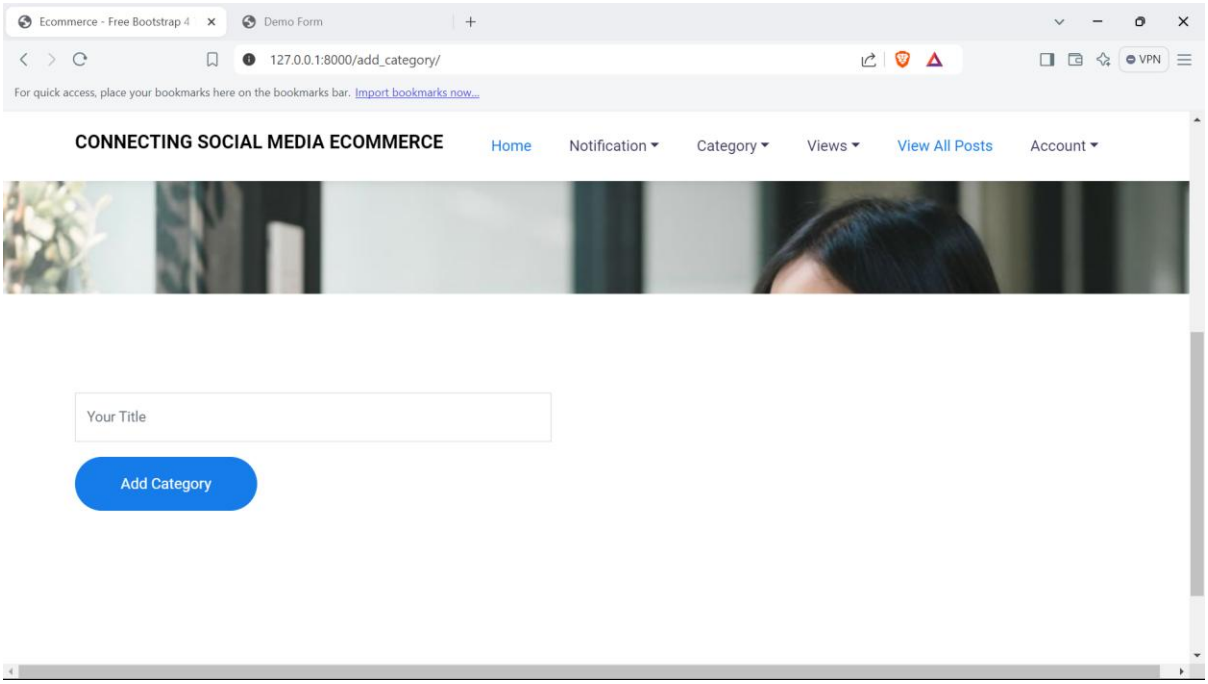
Admin View Notifications:



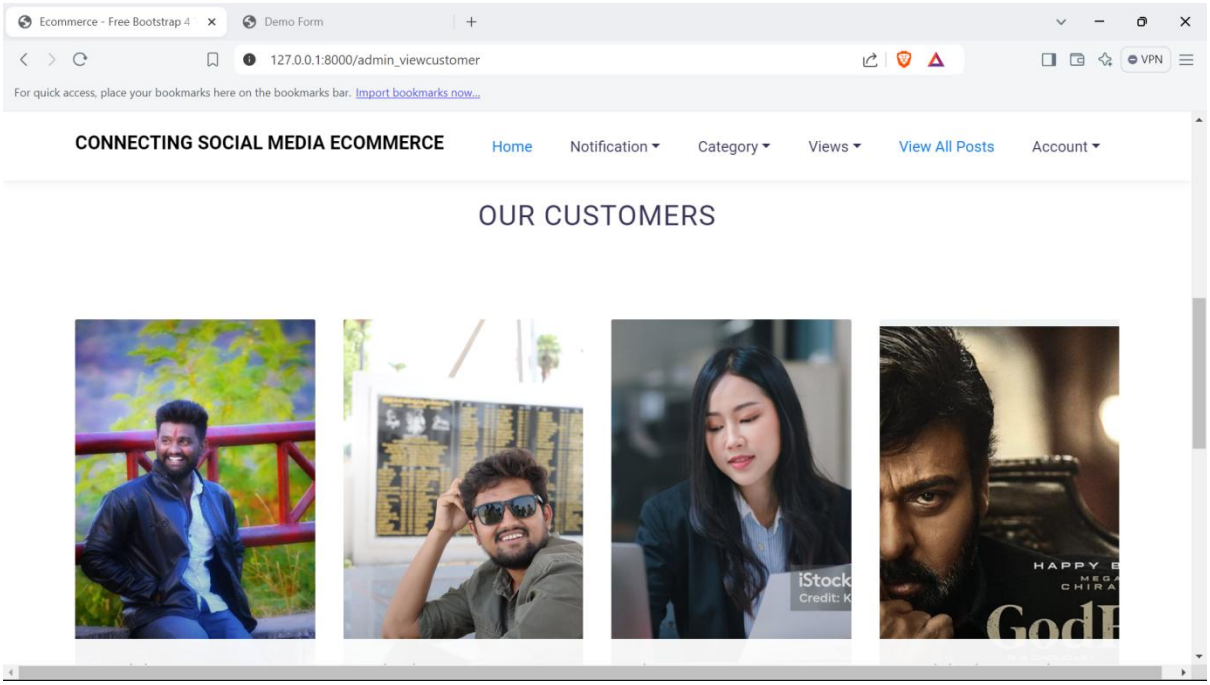
Admin Change Password:



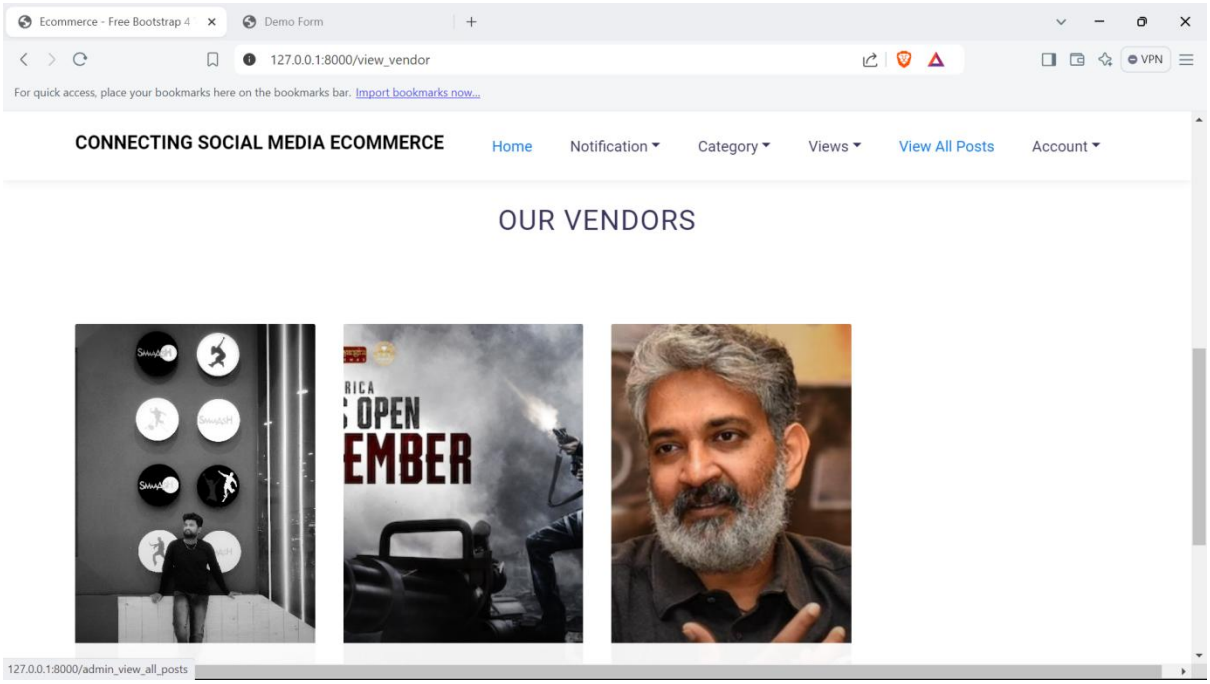
Admin Add Category:



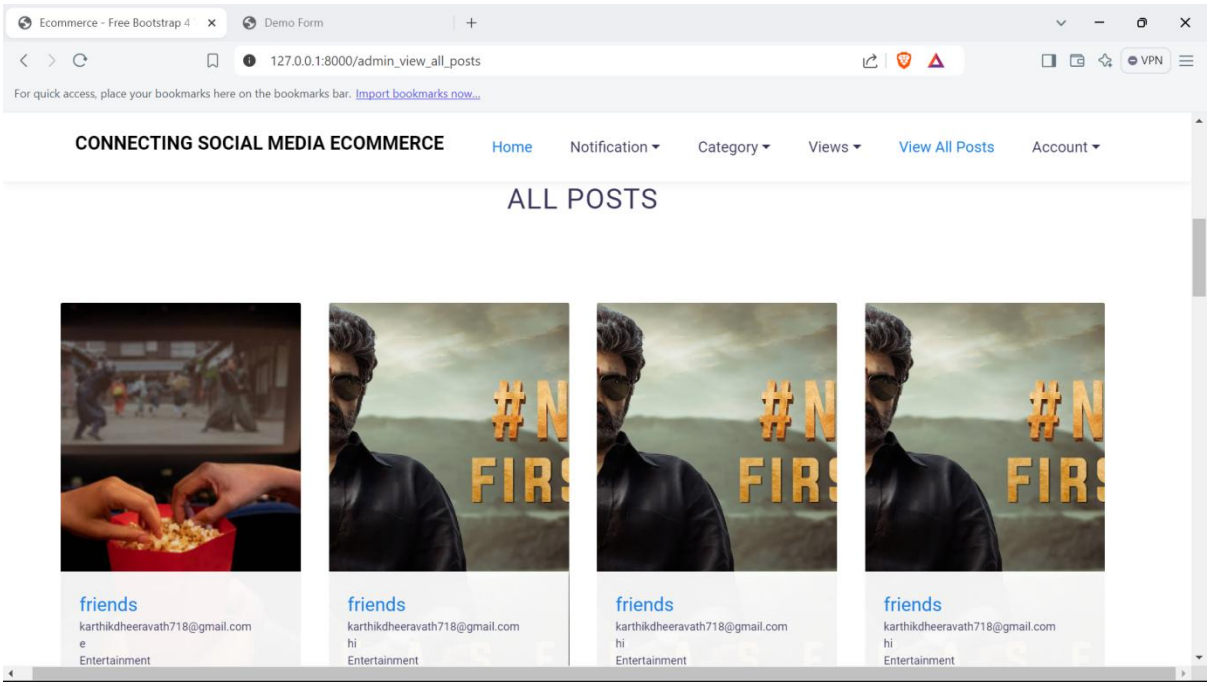
Admin View Customers :



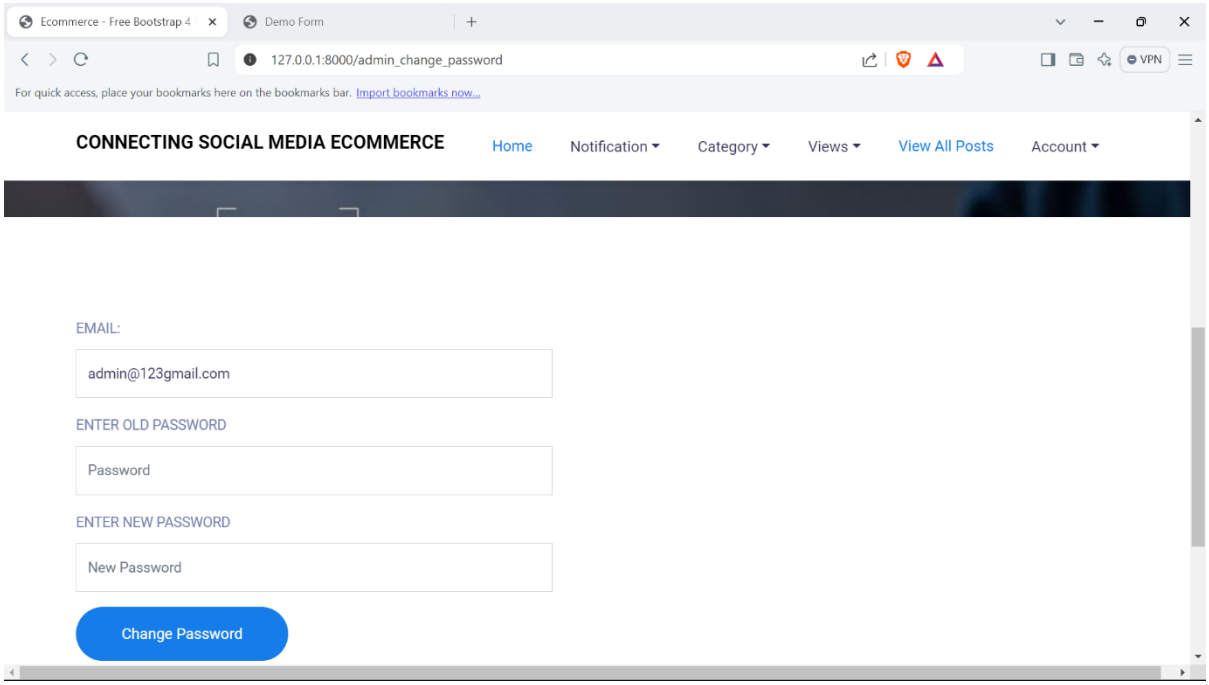
Admin View Vendors:



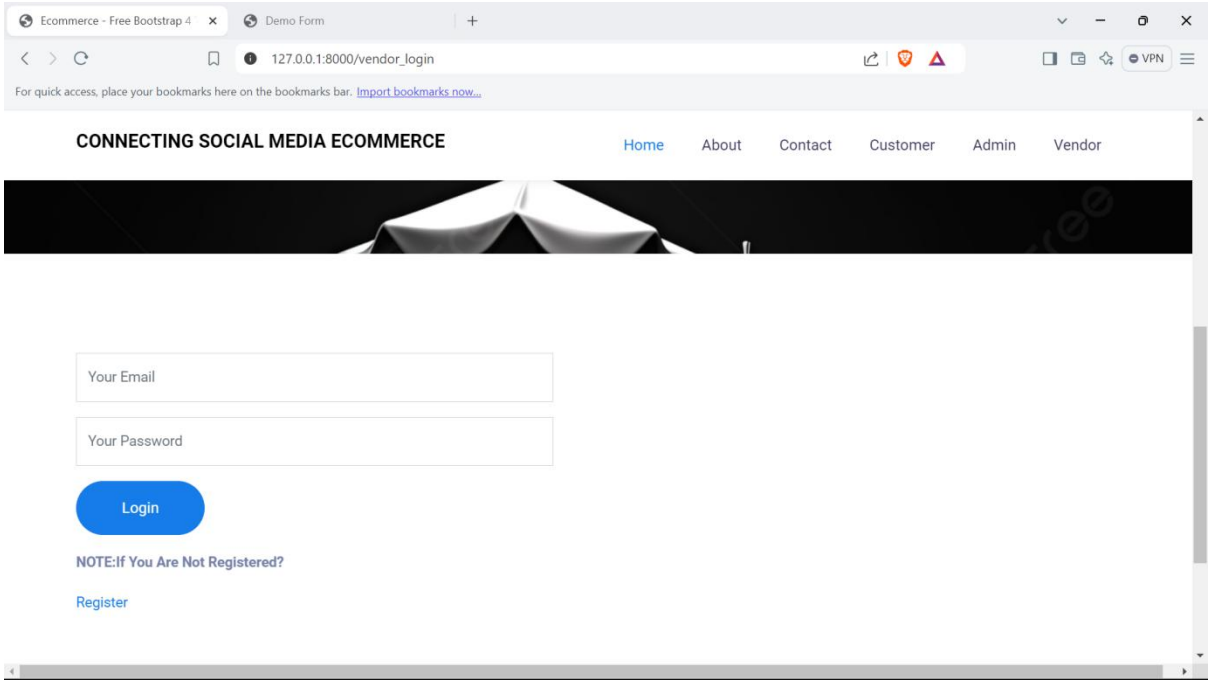
Admin View All Posts:



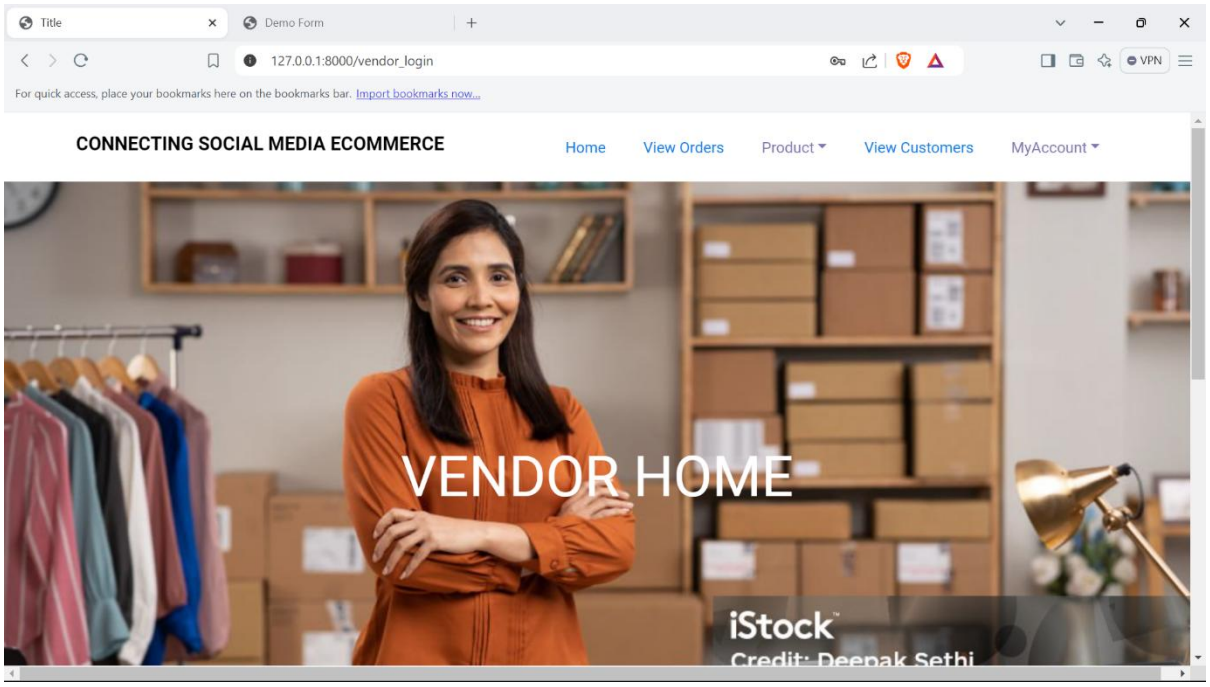
Admin Change Password:



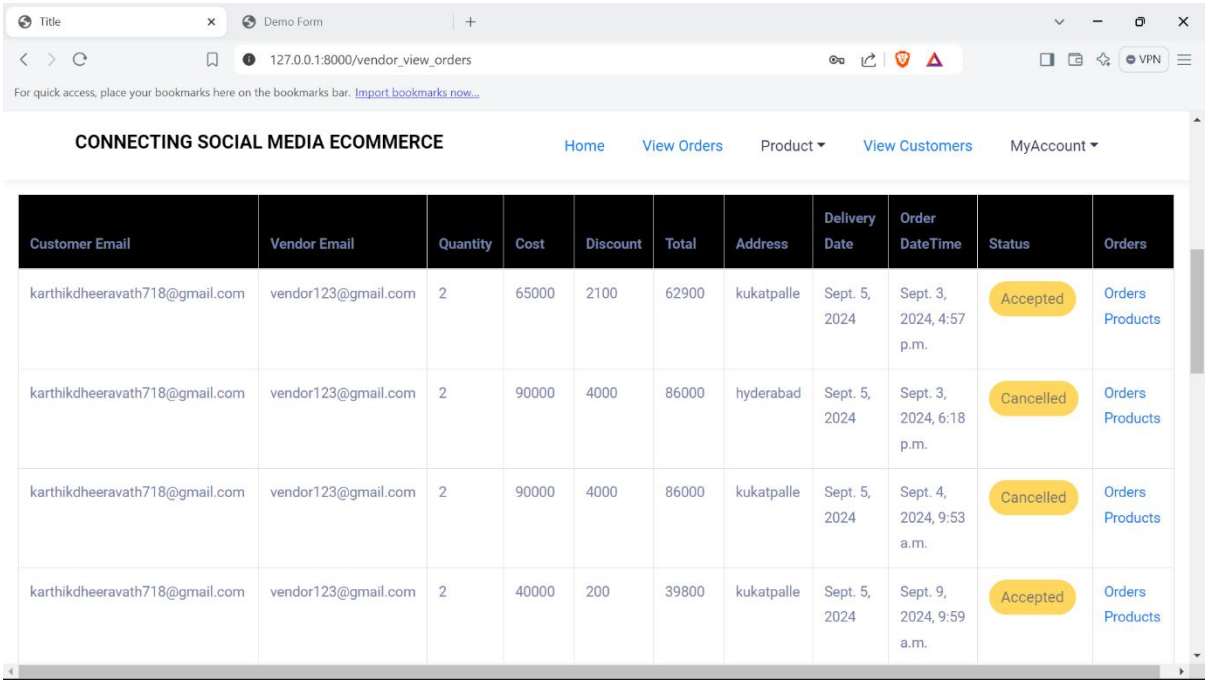
Vendor Login:



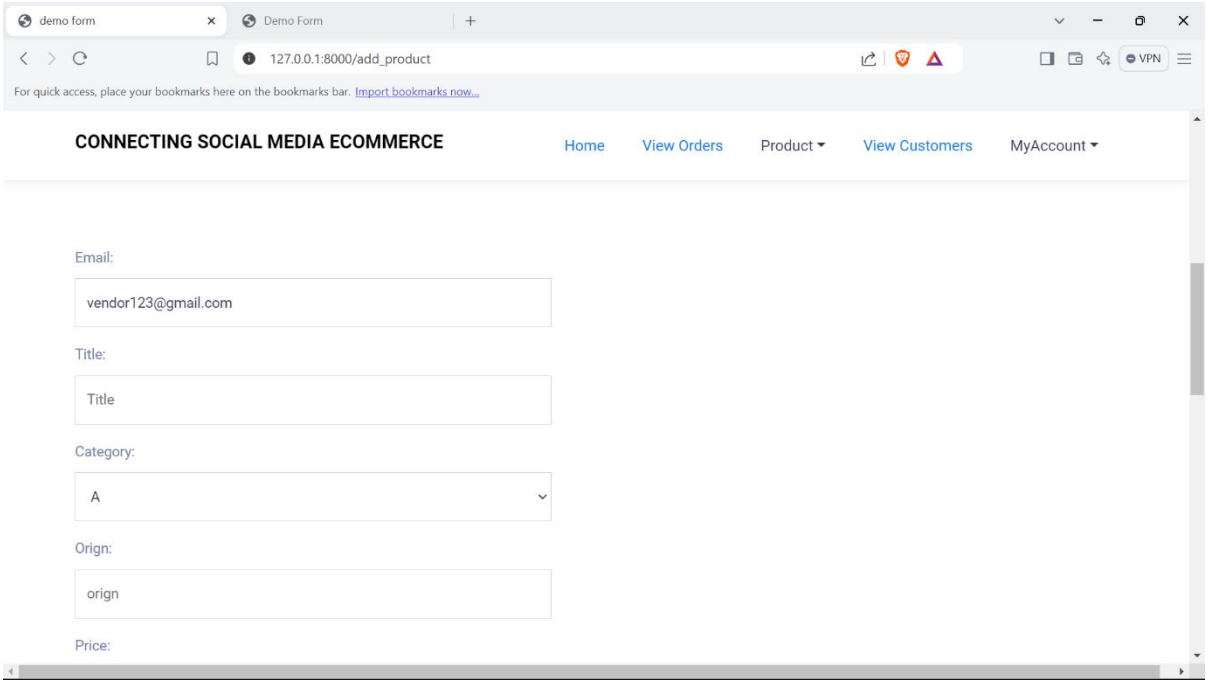
Vendor Home:



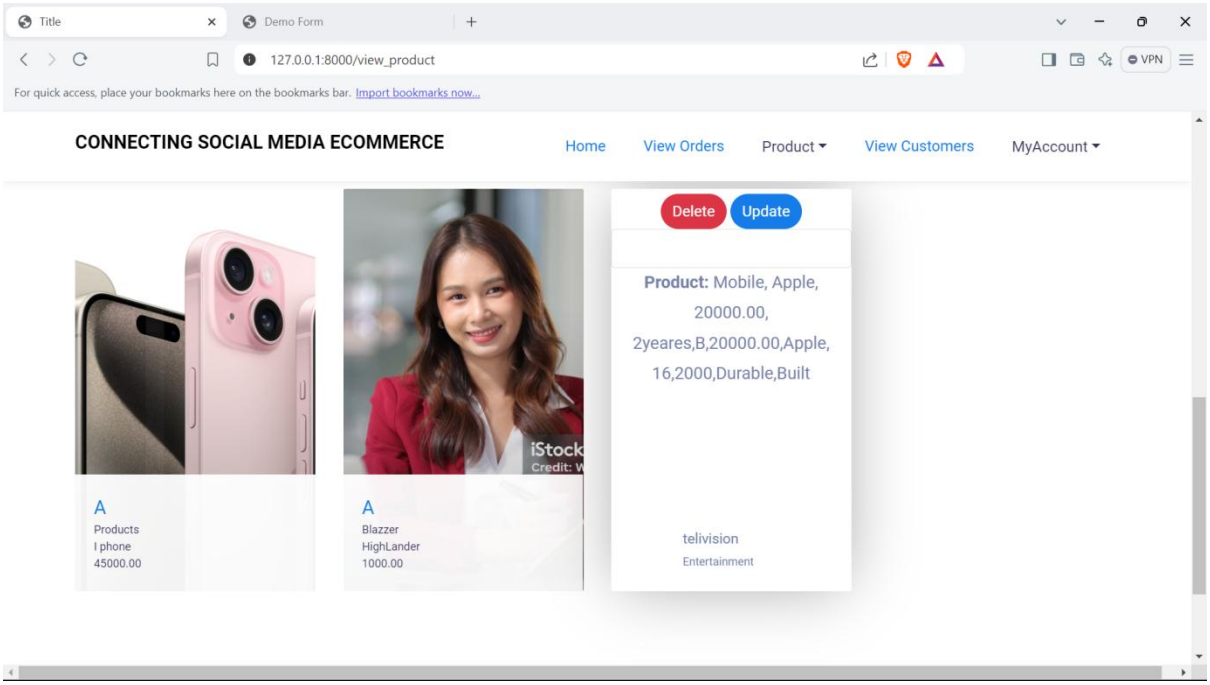
Vendor View Orders:



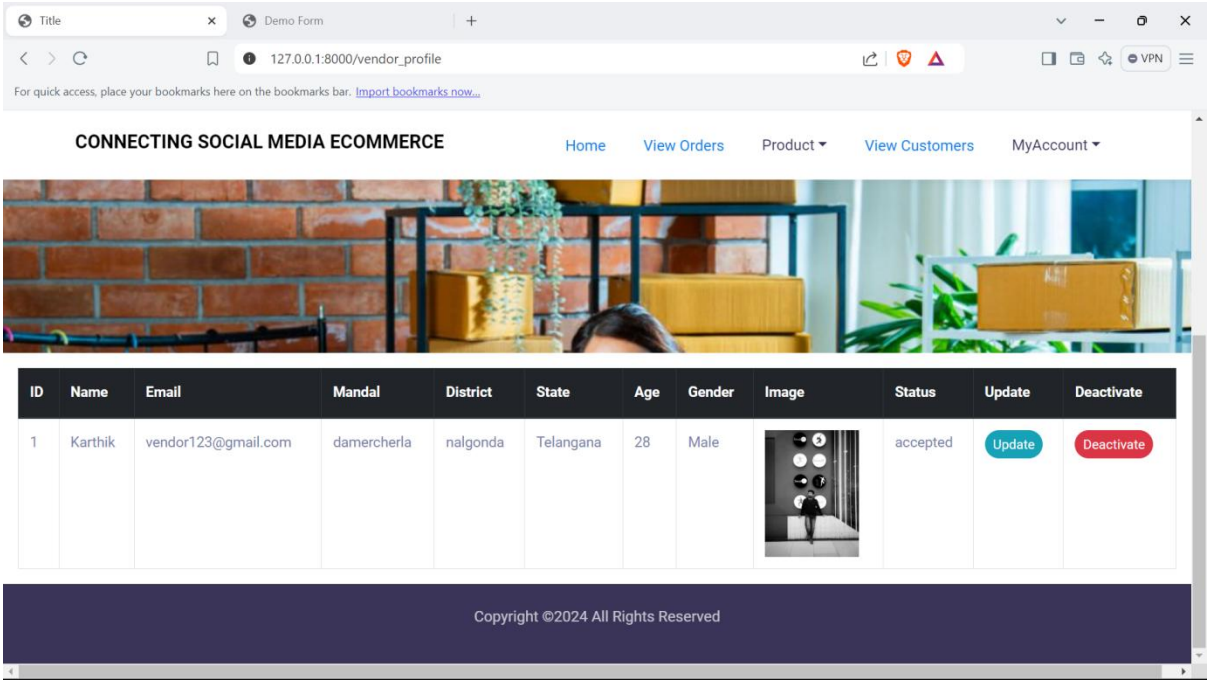
Vendor Add Product:



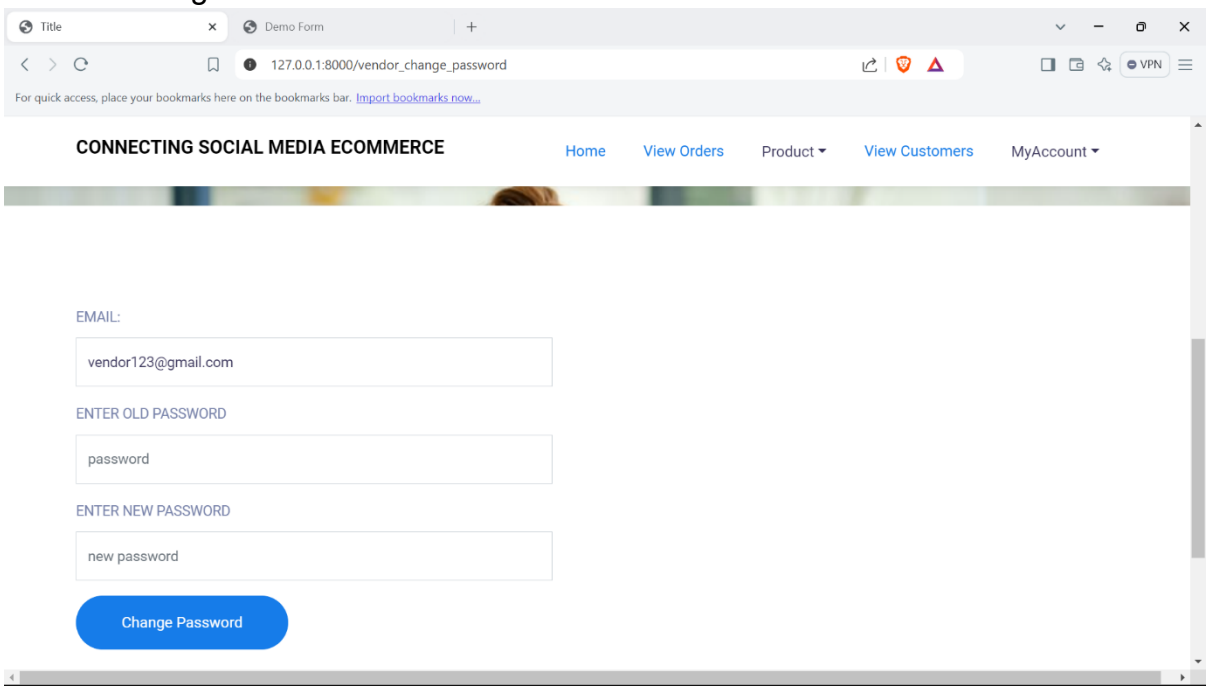
Vendor View Products:



Vendor My Profile:



Vendor Change Password:



- Logout

TESTING

Testing Process:

i. Introduction:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Types of Testing

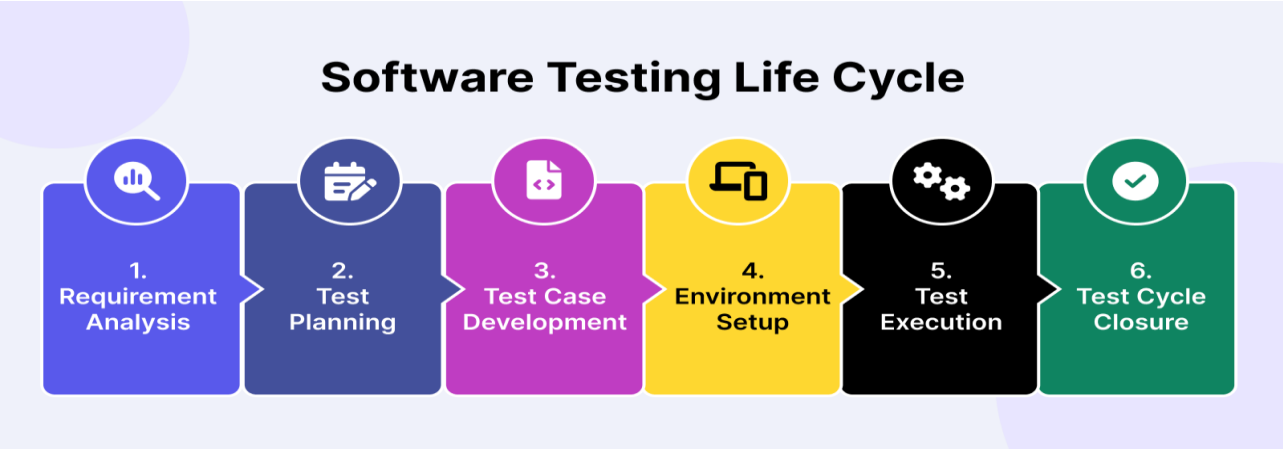
Unit Testing:
Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration Testing:
Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

System Testing:
System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

User Acceptance Testing (UAT):
User Acceptance Testing (UAT), or application testing, is the final stage of any software development or change request lifecycle before go-live. UAT meaning the final stage of any development process to determine that the software does what it was designed to

do in real-world situations. Actual users test the software to determine if it does what it was designed to do in real-world situations, validating changes made and assessing adherence to their organization's business requirements. The main purpose of acceptance testing is to validate end-to-end business flow.
User Acceptance Testing is a testing methodology where clients/end users participate in product testing to validate the product against their requirements. It is done at the client's site on the developer's site. For industries such as medicine or aerospace, contractual and regulatory compliance testing, and operational acceptance tests are also performed as part of user acceptance tests. It can be done in two levels i.e alpha testing and beta testing.



Functional Testing:
Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

Non Functional Testing:
Non functional testing is a type of software testing that verifies non functional aspects of the product, such as performance, stability, and usability. Whereas functional testing verifies whether or not the product does what it is supposed to, non functional testing verifies how well the product performs.

White Box Testing:
White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Test Strategy And Approach:

Field testing will be performed manually and functional tests will be written in detail.

Test Objectives:

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features To Be Tested:

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

iii. Test Cases:

Testcase Id	Objective	Description	Expected Result
log001	To test the URL	1)Enter URL 2)Open the webpage 3)Check for GUI	Login validation page should open
log002	To test the GUI	1)open the webpage 2) check the GUI components	It should display webpage with label,email,password,submit …etc
log003	To validate the Login page	1)Open the login page 2)keep empty the input elements 3)click on login button	It should display error message as "Please enter email and password".

log004	To validate the login page	1)Open the login page 2)Enter valid credentials 3)click on login button	It should display message as "Login successfully ".
mv005	To validate the mobile number	1)Open the login page 2)Enter mobile with less than 10 digits 3)click on login button	It should display error message as "Please enter mobile number with 10 digits".
mv006	To validate the mobile number	1)Open the login page 2)Enter mobile with greater than 10 digits 3)click on login button	It should display error message as "Please enter mobile number with 10 digits".
mv007	To validate the mobile number	1)Open the login page 2)Enter mobile number starts with 6,7,8,9 digits 3)click on login button	It should sent OTP to the respective mobile number
mv008	To validate the mobile number	1)Open the login page 2)Enter mobile number doesnot starts with 6,7,8,9 digits 3)click on login button	It should display error message as "Please enter mobile number strats with 6,7,8,9 digits".

mv009	To validate the mobile number	1)Open the login page 2)Enter mobile number with special characters 3)click on login button	It should display error message as "Please enter mobile number with valid digits

Test Results : All the test cases mentioned above passed successfully. No defects encountered.

Future Scope:

As the technology evolves, the web development industry promises noteworthy progress. We will see the industry growing and expanding with the introduction of new tools and technologies. Here is what the future of our website development are Responsive and Adaptive Designs, Payment Gateway Integration, Email / SMS Gateway integration, Mobile OTP Integrations, and also we can enhance this web application to mobile users by developing same application into mobile technology like android, IOS. So that user can use same application features even in mobile devices also. And also we can add GPS or location based services. You can even enhance the application by providing ML features like recommendations, predictions and classifications etc. We can also upgrade the best search and filter mechanisms too for this website.

CONCLUSION

Conclusion:

The **Connecting Social Media to E-Commerce** system is a forward-thinking solution that merges the dynamic nature of social interaction with the practical utility of online shopping. It offers personalized, engaging, and efficient user experiences for customers, vendors, and administrators alike. By integrating social proof and interactive features into the e-commerce journey, this platform aims to revolutionize how users shop and interact online.

BIBLIOGRAPHY

&

WEBILOGRAPHY

PYTHON BIBLIOGRAPHY

- Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming - Eric Matthes - No Starch Press
- Python Pocket Reference: Python in Your Pocket - Mark Lutz- O′Reilly
- Python Programming: An Introduction to Computer Science - John M Zelle - Ingram short title
- Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython - Wes Mckinney - O′Reilly
- Python Cookbook: Recipes for Mastering Python 3 - David Beazley - O'Reilly
- Python Data Science Handbook: Essential Tools for Working with Data - Jake Vanderplas - O′Reilly
- Python Programming for Beginners - Philip Robbins
- Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow - Sebastian Raschka Vahid Mirjalili - Ingram short title
- Django for Beginners: Build websites with Python and Django - William S. Vincent - WelcomeToCode
- Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit - Steven Bird - O′Reilly
- HTML Black Book by Holzner
- Software Engineering by Roger Pressman

- Database Management Systems - Raghu Ramakrishnan and Johannes Gehrke

References:

References for the Project Development Were Taken From the following Books and Web Sites

Web References

1. http://www.javatpoint.com
2. http://www.w3schools.com
3. http://www.stackoverflow.com
4. https://docs.oracle.com/javase/tutorial/java/index.html
5. https://www.programiz.com
6. https://docs.python.org/3/tutorial/index.html
7. https://www.djangoproject.com/start/
8. https://www.geeksforgeeks.org/django-tutorial/
9. https://www.umsl.edu/~siegelj/CS4010/murach/Chapter1slides.pdf
10. https://github.com/