

1.Problem Statement

Problem Scenario:

Electronic Commerce is process of doing business through computer networks. A person sitting on his chair in front of a computer can access all the facilities of the Internet to buy or sell the products. Unlike traditional commerce that is carried out physically with effort of a person to go & get products, ecommerce has made it easier for human to reduce physical work and to save time.

Proposed Solution:

This project provides an insight into the working of ecommerce applications. It manages all the information about Products, Sales, Category, Products.

There are two entities who will have the access to the system. One is the admin and another one will be the registered user. Admin can add product details, view all the order details and can also view the sales of the products. User need to register with basic registration details to generate a valid username and password. After the user logs in, it can view all the products that are recommended on the homepage compiled by the system based on user's information. From the recommended products, the user can even further view its details and then if interested to buy, the system gives add to cart option for purchasing the product.

The services offered by Fashion Store ecommerce application for multiple roles are as follows.

- **User**
 1. Login to the site
 2. New user have to register
 3. Search for items/products
 4. View the product details
 5. Add products in cart
 6. Order the products
 7. Order confirmation
 8. Logout from the site

- **Admin**
 1. Login to the site
 2. View admin dashboard
 3. Add product Details
 4. Add categories
 5. View all order details
 6. Logout from the site

ANALYSIS

1. Requirements Elicitation

The following table represents the categorization of requirements captured for ecommerce application store:

SNO	REQUIREMENTS	TYPE	PRIORITY
1	Home	Functional	Must Have
2	<u>Login</u> : User logs in by providing email and password.	Functional	Must have
3	<u>Register</u> : New user can register by providing first name, last name, email, and password.	Functional	Must have
4	<u>Shop</u> : User's can see the shop details and then can view the products	Functional	Must have
5	<u>View Products</u> : User's can view all the products description and can add them in cart	Functional	Must have
6	<u>Cart</u> : Display the products added by user and can now place the order for them	Functional	Must have
7	<u>Checkout</u> : User can checkout by providing first name, last name, country, address, postal code, city, phone number, email, and some additional information	Functional	Must have
8	<u>Order Placed</u> : User now get the receipt of the placed order and order confirmation	Functional	Must have

SNO	REQUIREMENTS	TYPE	PRIORITY
9	<u>Logout:</u> User can log out from this site.	Functional	Must have
10	<u>Admin Login:</u> Admin can login using his email id and password	Functional	Must have
11	<u>Admin Dashboard:</u> Admin have the option to manage the categories and product list of the application	Functional	Must have
12	<u>CRUD Categories:</u> Admin can add, delete, update all the categories list	Functional	Must have
12	<u>CRUD Products:</u> Admin can add, delete, update all the list of products that needs to be displayed to users	Functional	Must have
13	<u>Admin Logout:</u> Admin can logout of his account	Functional	Must have

Use-case View

2. Identification of Actors

Actors represent system users. They are NOT part of the system. They represent anyone or anything that interacts with the system. An actor is someone or something that:

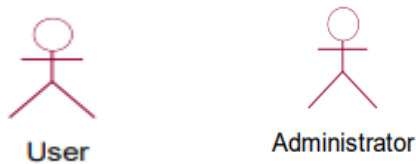
- Interacts with or uses the system
- Provides input to and receives information from the system
- Is external to the system and has no control over the use cases

Actors are discovered by examining:

- Who directly uses the system
- Who is responsible for maintaining the system
- External hardware used by the system
- Other systems that need to interact with the system

The needs of the actor are used to develop use cases. This insures that the system will be what the user expected.

Graphical depiction: An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram.



For example, User Actors identified are:

- 1) User
- 2) Administrator

- 1) User: User has to login into his/her account to view list of products, add them in cart, confirm and place the order and he/she has to logout the account after his/her is placed.
- 2) Administrator: He has the power to perform CRUD operations on products in the shop and CRUD operations on categories. Also he has the access to all the transaction details

IDENTIFICATION OF USE-CASES OR SUB USE-CASES

Use-case diagrams graphically represent system behaviour. These diagrams present a high level view of how the system is used as viewed from an outsider's perspective. A use-case diagram may contain all or some of the use cases of a system.

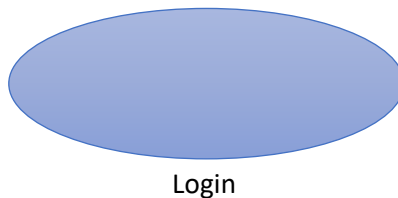
A use-case diagram can contain:

- Actors
- Use cases
- Relationships between actors and use cases in the system

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behaviour of the system as implemented. In its simplest form, a use case can be described as a specific way of using the system from a user's perspective.

A more detailed description might characterize a use case as:

- A pattern of behaviour the system exhibits
- A sequence of related transactions performed by an actor and the system The UML notation for use case is:



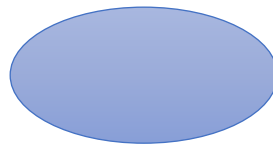
PURPOSE OF USE CASES:

- Well structured use cases denote essential system or subsystem behaviours only, and are neither overly general nor too specific.
- A use case represents a functional requirement of the system as a whole
- Use cases represent an external view of the system
- A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself.

Use-cases identified for ecommerce application are:

1. Use-case name: Login

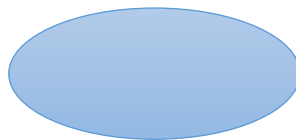
This is a use case which is used by actor to log on to the system and view the available set of operations that he/she can perform



Login

2. Use-case name: view Products

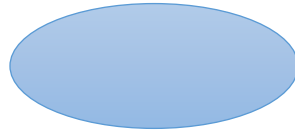
System allows the customer to view the product images and details of the product that they want



ViewProducts

3. Use-case name: Register

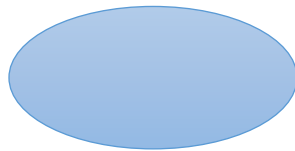
This use case allows the an anonymous user to register on to the site



Register

4. Use-case name: cart

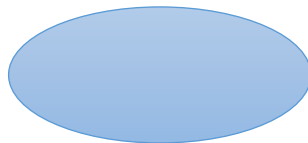
System allows the customer to add products in cart which they want to buy



Cart

5. Use-case name: Product Add

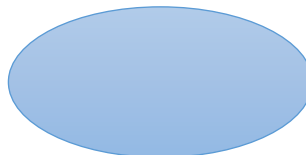
System allows the administrator to add, update or delete products based on the need



ProductAdd

6. Use-case name: Categories Add

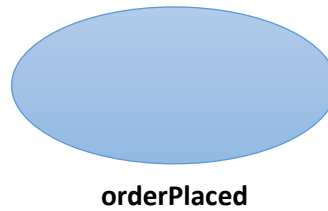
System allows the administrator to add, update or delete products categories based on the need



CategoriesAdd

7. Use-case name: order Placed

System allows the customer to place and confirm the order of the product that they want



BUILDING REQUIREMENTS MODEL THROUGH USE-CASE DIAGRAM

Definition:

Use-case diagrams graphically represent system behaviour. These diagrams present a high level view of how the system is used as viewed from an outsider's perspective. Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behaviour of the system as implemented.

RELATIONS:

Association Relationship:

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.



Dependency Relationship:

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

We can provide here

1. Include relationship: It is a stereotyped relationship that connects a base use case to an inclusion use case. An include relationship specifies how the behaviour in the inclusion use case is used by the base use case.

<<includes>>

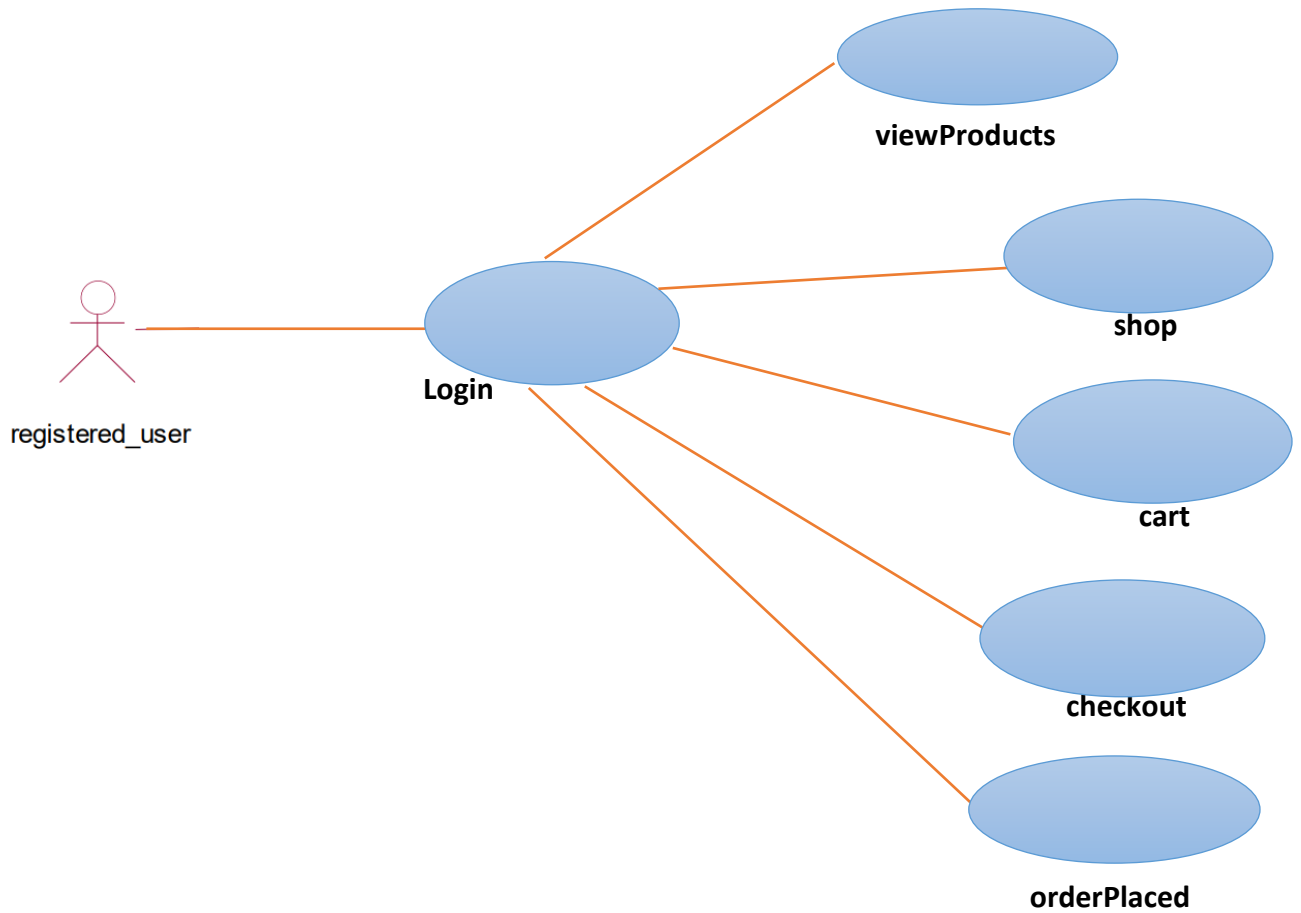


2. Extend relationship: It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case. <> is used when you wish to show that a use case provides additional functionality that may be required in another use case.

<<excludes>>



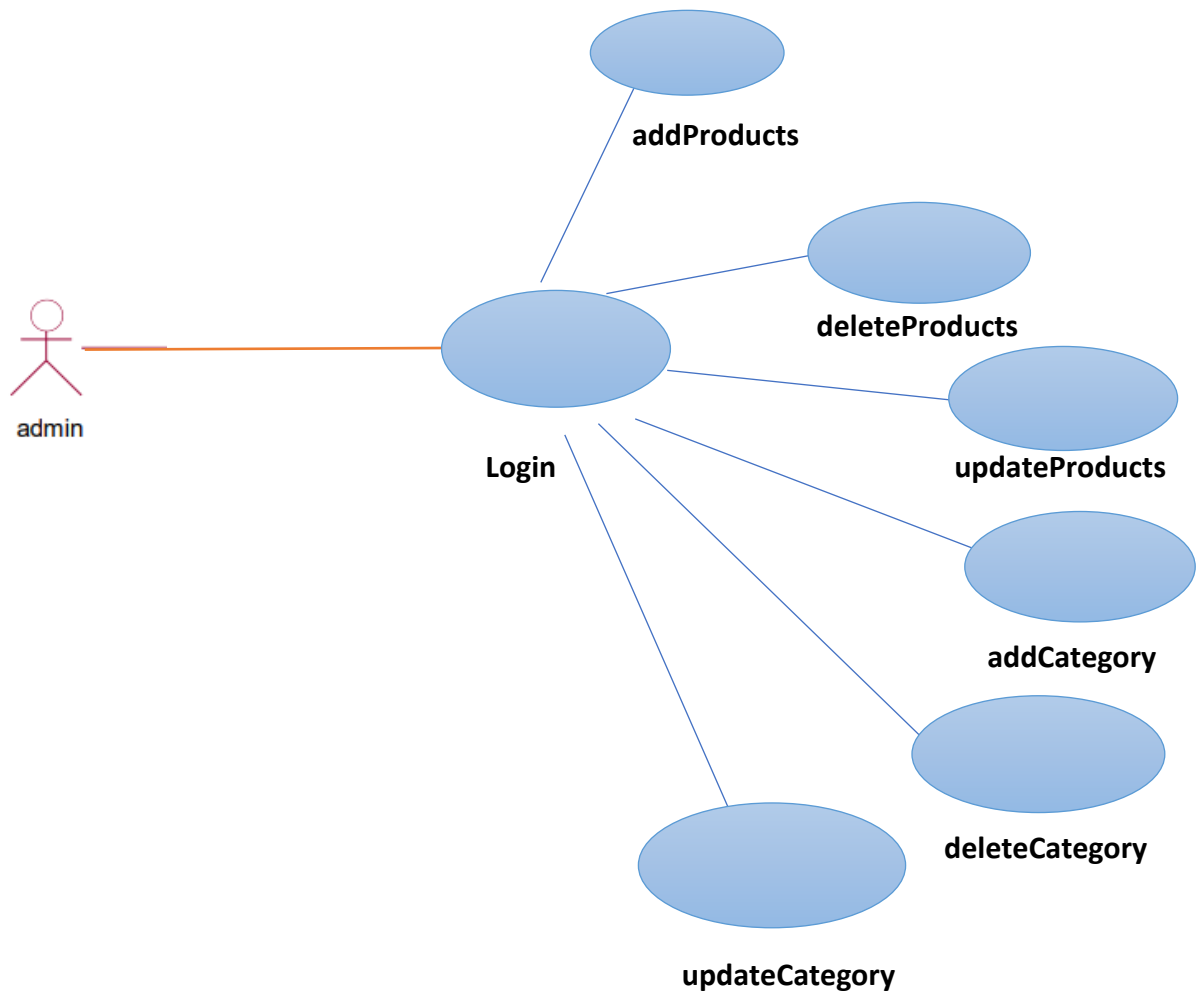
USE CASE DIAGRAM FOR LOGIN TO THE SITE:



USE CASE FOR REGISTERING TO THE SITE:



USECASE FOR ADMIN OF THE SITE:



FLOW OF EVENTS

A flow of events is a sequence of transactions performed by the system. They typically contain very detailed information. Flow of events document is typically created in the elaboration phase.

Each use case is documented with flow of events

- A description of events needed to accomplish required behaviour
- Written in terms of what the system should do, NOT how it should do it
- Written in the domain language, not in terms of the implementation

A flow of events should include

- When and how the use case starts and ends
- What interaction the use case has with the actors
- What data is needed by the use case
- The description of any alternate or exceptional flows

The flow of events for a use case is contained in a document called the use case specification. Each project should use a standard template for the creation of the use case specification. Includes the following

1. Use case name – Brief Description

2. Flow of events –

- Basic flow
- Alternate flow
- Special requirements
- Pre conditions
- Post conditions
- Extension points

Use case specification for Buying the Product:

1. Use case name: Online Shopping brief description: This Use case allows the customer/user to buy the product they want.

2. Flow of events:

2.1. Basic flow

1) This use case begins when customer logs onto online shopping site and enters his/her password. The system verifies that the password is valid(if the password is invalid , alternate flow 3.1 is executed)

2) System displays the list of products available in e-shops

2.1) If customer need to buy the product then he/she should select that product image on the home screen

2.1.1) Customer should add that product in the cart.

2.1.2) Customer should enter his/her address details before placing the order.

3. Alternate flow:

3.1) Invalid password: An invalid password is entered. The customer can re-enter password or terminate use case.

3.2) Forgot Password: If the customer forget its password then can get it using email or terminate use case.

3.3) If there is no quantity left of that product(out of stock) then customer should select the other one or terminate the use case

4 .Special requirements: There are no special requirements

5. Pre conditions: Login use case must execute before this use case begins and customer must be a valid customer.

6. Post conditions: After completing the use case the status or confirmation mail of this use case must be sent to user email and mobile.

7. Extension points: There are no extension points.

Use case specification for View Products:

1. Use case name: Request for Viewing Product: This use case allows customer to view the images and description about the product.

2. Flow of events:

2.1. Basic flow:

1) On selecting the product just click on the product image and then you can able to read the details and price of the product.

2) It displays the description of the specified product and now you can add them in cart

2.2. Alternate flows:

2.2.1) Product images or description not available.

3. Special requirements: There are no special requirements.

4. Pre conditions: The user must log in to the site and then select and purchase the product.

5. Post conditions: There are no post conditions

6. Extension points: There are no Extension points.

IDENTIFICATION OF ANALYSIS CLASSES

The class diagram is fundamental to object-oriented analysis. Through successive iterations, it provides both a high level basis for systems architecture, and a low-level basis for the allocation of data and behavior to individual classes and object instances, and ultimately for the design of the program code that implements the system. So, it is important to identify classes correctly. However, given the iterative nature of the object-oriented approach, it is not essential to get this right on the first attempt itself.

Approaches for identifying classes: We have four alternative approaches for identifying classes:

- 1) The noun phrase approach;
- 2) The common class patterns approach;
- 3) The use- case driven TO sequence/collaboration modeling approach;
- 4) Class Responsibility collaboration cards (CRC) approach.

1. NOUN PHRASE APPROACH:

In this method, analyst read through the requirements or use cases looking for noun phrases. Nouns in the textual description are considered to be classes and verbs to be methods of the classes. All plurals are changed to singular, the nouns are listed, and the list divided into three categories: relevant classes, fuzzy classes (the "fuzzy area," classes we are not sure about), and irrelevant classes. It is safe to scrap the irrelevant classes, which either have no purpose or will be unnecessary. Candidate classes then are selected from the other two categories. Here identifying classes and developing a UML class diagram just like other activities is an iterative process.

1.) Identifying Tentative Classes: The following are guidelines for selecting classes in an application:

- Look for nouns and noun phrases in the use cases.
- Some classes are implicit or taken from general knowledge.
 - All classes must make sense in the application domain; avoid computer implementation classes-defer them to the design stage.
- Carefully choose and define class names.

2.) Selecting Classes from the Relevant and Fuzzy Categories: The following guidelines help in selecting candidate classes from the relevant and fuzzy categories of classes in the problem domain.

a) Redundant classes. Do not keep two classes that express the same information. If more than one word is being used to describe the same idea, select the one that is the most meaningful in the context of the system. This is part of building a common vocabulary for the system as a whole. Choose your vocabulary carefully; use the word that is being used by the user of the system.

b) Adjectives classes: "Be wary of the use of adjectives. Adjectives can be used in many ways. An adjective can suggest a different kind of object, different use of the same object, or it could be utterly irrelevant. Does the object represented by the noun behave differently when the adjective is applied to it? If the use of the adjective signals that the behavior of the object is different, then make a new class".

c) Attribute classes: Tentative objects that are used only as values should be defined or restated as attributes and not as a class.

2) COMMON CLASS PATTERNS APPROACH

The second method for identifying classes is using common class patterns, which is based on a knowledge base of the common classes. The following patterns are used for finding the candidate class and object:

a) Concept class: A concept is a particular idea or understanding that we have of our world. The concept class encompasses principles that are not tangible but used to organize or keep track of business activities or communications.

b) Events class: Events classes are points in time that must be recorded. Things happen, usually to something else at a given date and time or as a step in an ordered sequence. Associated with things remembered are attributes (after all, the things to remember are objects) such as who, what, when, where, how, or why.

c) Organization class: An organization class is a collection of people, resources, facilities, or groups to which the users belong; their capabilities have a defined mission, whose existence is largely independent of the individuals

d) People class (also known as person, roles, and roles played class): The people class represents the different roles users play in interacting with the application.

3) USE-CASE DRIVEN APPROACH: IDENTIFYING CLASSES AND THEIR BEHAVIORS THROUGH SEQUENCE/COLLABORATION MODELING

One of the first steps in creating a class diagram is to derive from a use case, via a collaboration (or collaboration diagram), those classes that participate in realizing the use case. Through further analysis, a class diagram is developed for each use case and the various use case class diagrams are then usually assembled into a larger analysis class diagram. This can be drawn first for a single subsystem or increment, but class diagrams can be drawn at any scale that is appropriate, from a single use case instance to a large, complex system. Identifying the objects involved in a collaboration can be difficult at first, and takes some practice before the analyst can feel really comfortable with the process. Here a collaboration (i.e. the set of classes that it comprises) can be identified directly for a use case, and that, once the classes are known, the next step is to consider the interaction among the classes and so build a collaboration diagram. From collaboration diagram to class diagram.

The next step in the development of a requirements model is usually to produce a class diagram that corresponds to each of the collaboration diagrams. Collaboration diagrams are obtained by result of reasonably careful analysis, the transition is not usually too difficult. The similarities & differences B/W Collaboration and class diagrams are :

First, consider the similarities: Both show class or object symbols joined by connecting lines. In general, a class diagram has more or less the same structure as the corresponding collaboration diagram. In particular, both should show classes or objects of the same types. Any of the three analysis stereotype notations for a class can be used on either diagram, and stereotype labels can also be omitted from individual classes, or from an entire diagram. Next, the differences are:

1. The difference is that an actor is almost always shown on a collaboration diagram, but not usually shown on a class diagram. This is because the collaboration diagram represents a particular interaction and the actor is an important part of this interaction. However, a class diagram shows the more enduring structure of associations among the classes, and frequently supports a number of different interactions that may represent several different use cases.
2. A collaboration diagram usually contains only object instances, while a class diagram usually contains only classes.
3. The connections between the object symbols on a collaboration diagram symbolize links between objects, while on a class diagram the corresponding connections stand for associations between classes.

4. A collaboration diagram shows the dynamic interaction of a group of objects and thus every link needed for message passing is shown. The labelled arrows alongside the links represent messages between objects. On a class diagram, the associations themselves are usually labelled, but messages are not shown.

5. Finally, any of the three stereotype symbols can be used on either diagram , there are also differences in this notation.

4) Class Responsibility collaboration Cards (CRC Cards)

At the starting , for the identification of classes we need to concentrate completely on uses cases. A further examination of the use cases also helps in identifying operations and the messages that classes need to exchange. However, it is easy to think first in terms of the overall responsibilities of a class rather than its individual operations. A responsibility is a high level description of something a class can do.

IDENTIFICATION OF RESPONSIBILITIES OF CLASSES Class Responsibility Collaboration (CRC)

CRC cards provide an effective technique for exploring the possible ways of allocating responsibilities to classes and the collaborations that are necessary to fulfill the responsibilities. CRC cards can be used at several different stages of a project for different purposes.

1. They can be used early in a project to help the production of an initial class diagram .
2. To develop a shared understanding of user requirements among the members of the team.
3. CRCs are helpful in modeling object interaction.

IMPLEMENTATION SCREENSHOTS

CUSTOMER

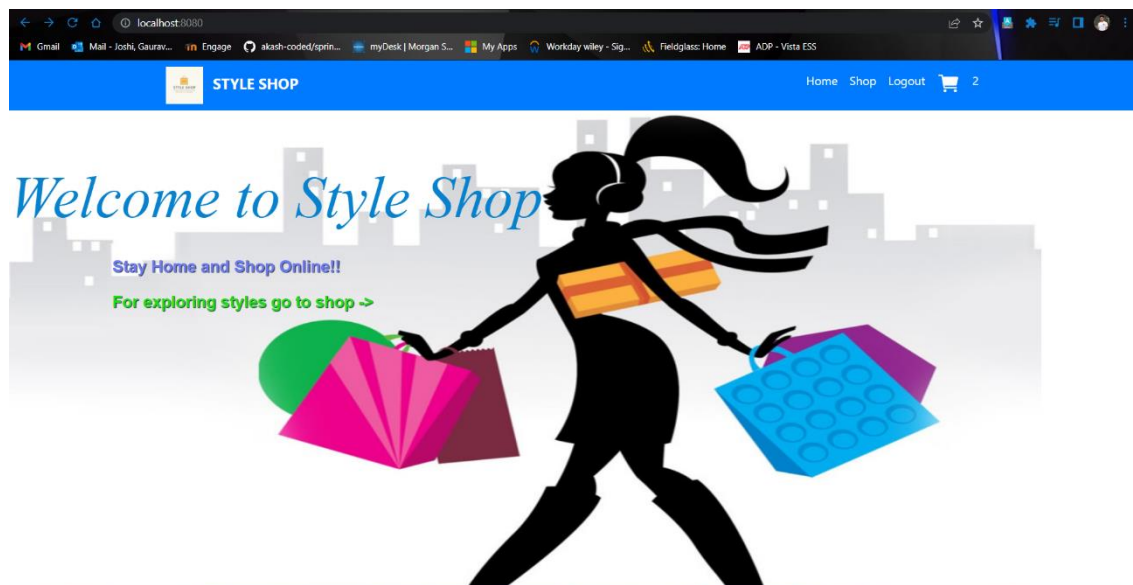
LOGIN PAGE:

The screenshot shows a web browser at localhost:8080/login. The page has a blue header with the 'STYLE SHOP' logo and navigation links for Home, Shop, and Sign Up. The main content area is a white box titled 'Login'. It contains two input fields: 'Email' with placeholder text 'Your Email' and 'Password' with placeholder text 'Password'. Below these is a blue 'Login' button. Underneath the button are two links: 'Don't have an account Register here' and 'Forgot password?'. Below these links is the text 'OR' and a blue button labeled 'Sign-In with google'.

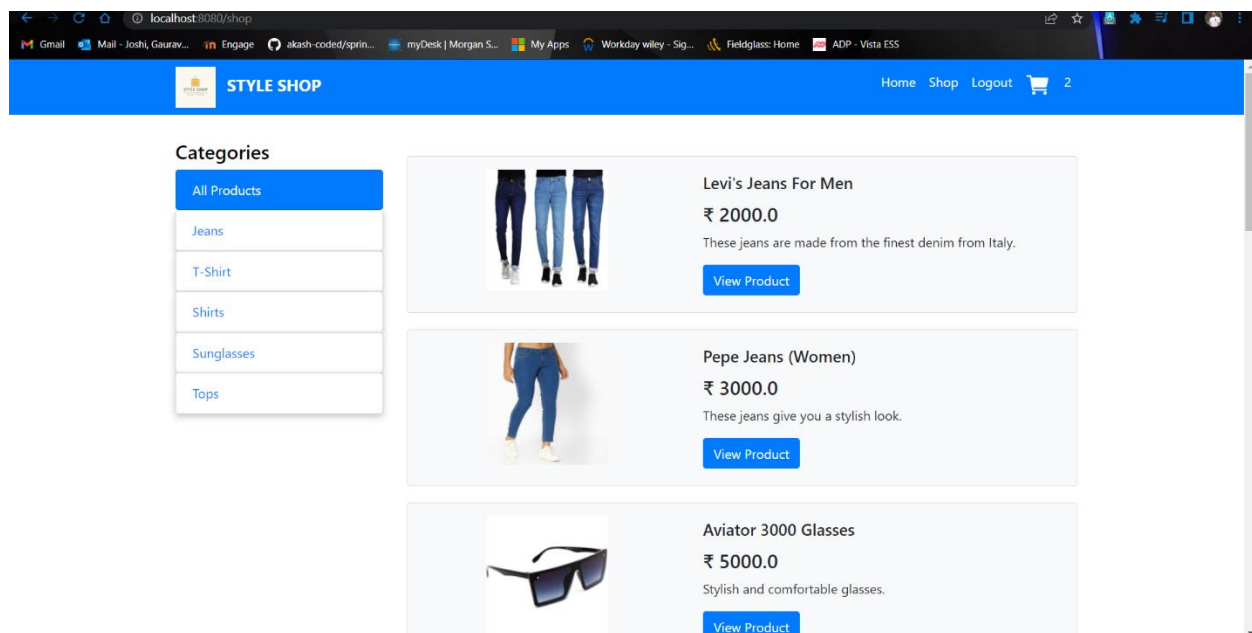
REGISTER:

The screenshot shows a web browser at localhost:8080/register. The page has a blue header with the 'STYLE SHOP' logo and navigation links for Home, Shop, and Sign Up. The main content area is a white box titled 'Sign Up Now!'. It contains four input fields: 'First Name' with placeholder text 'Your Firstname', 'Last Name' with placeholder text 'Your Lastname', 'Email address' with placeholder text 'Email', and 'Password' with placeholder text 'Password'. Below the 'Email address' field is a small text line: 'We'll never share your email with anyone else.' Below the 'Password' field is a blue 'Register' button.

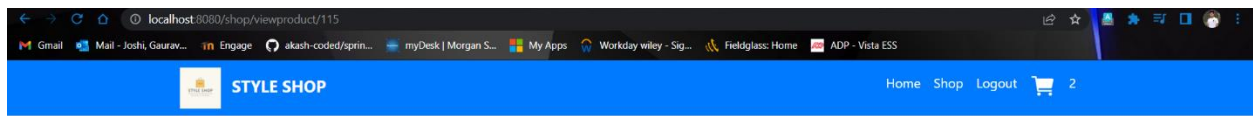
HOME PAGE:



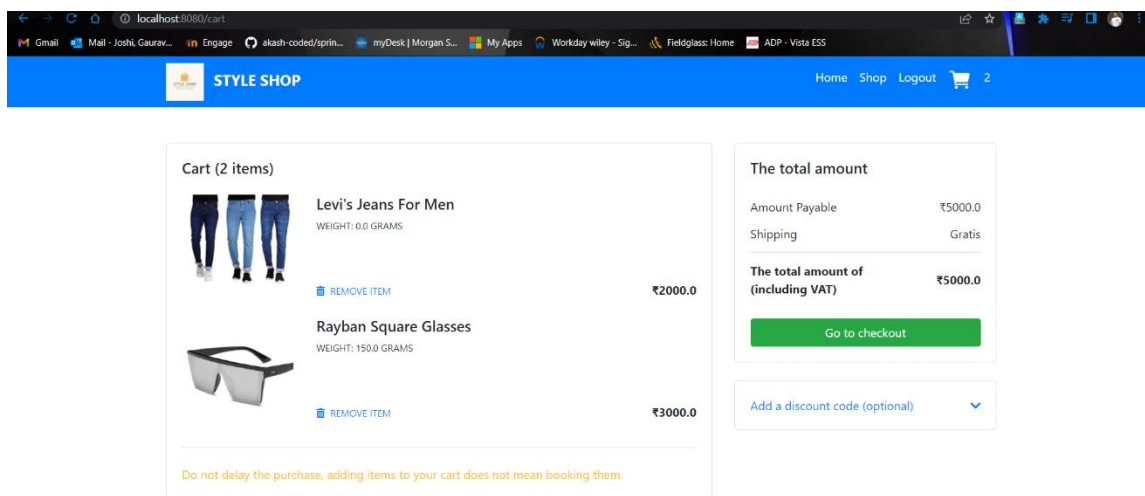
SHOP:



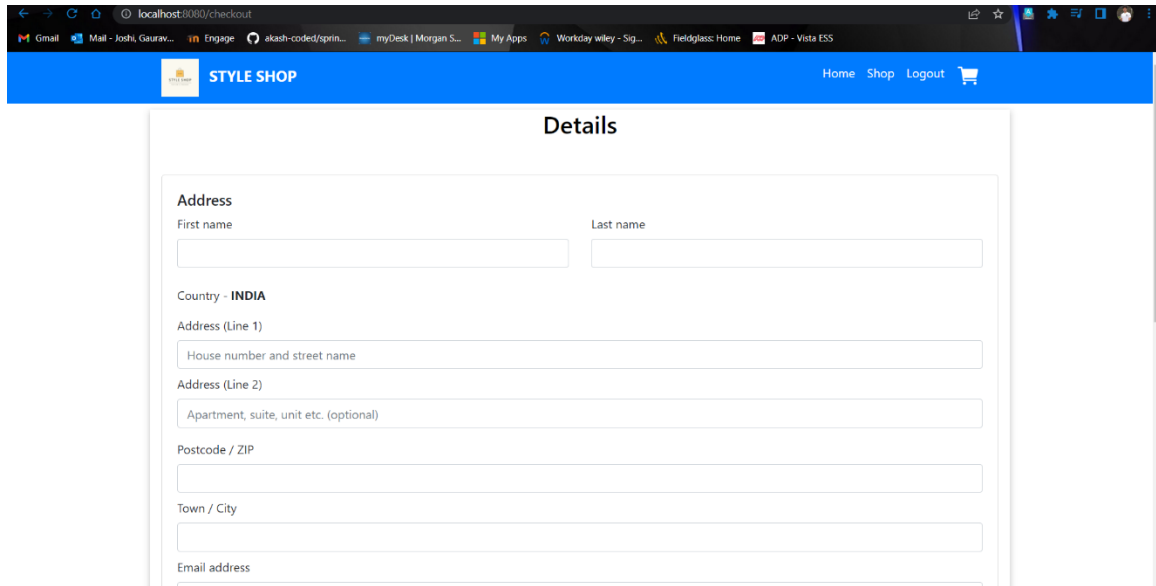
VIEW PRODUCT:



CART:



CHECKOUT:



The screenshot shows a web browser window with the URL `localhost:8080/checkout`. The browser's address bar and tabs are visible at the top. The page has a blue header with the text "STYLE SHOP" and navigation links "Home", "Shop", and "Logout". The main content area is titled "Details" and contains a form for address information. The form includes fields for "First name", "Last name", "Country" (set to "INDIA"), "Address (Line 1)", "Address (Line 2)", "Postcode / ZIP", "Town / City", and "Email address".

Details

Address

First name

Last name

Country - **INDIA**

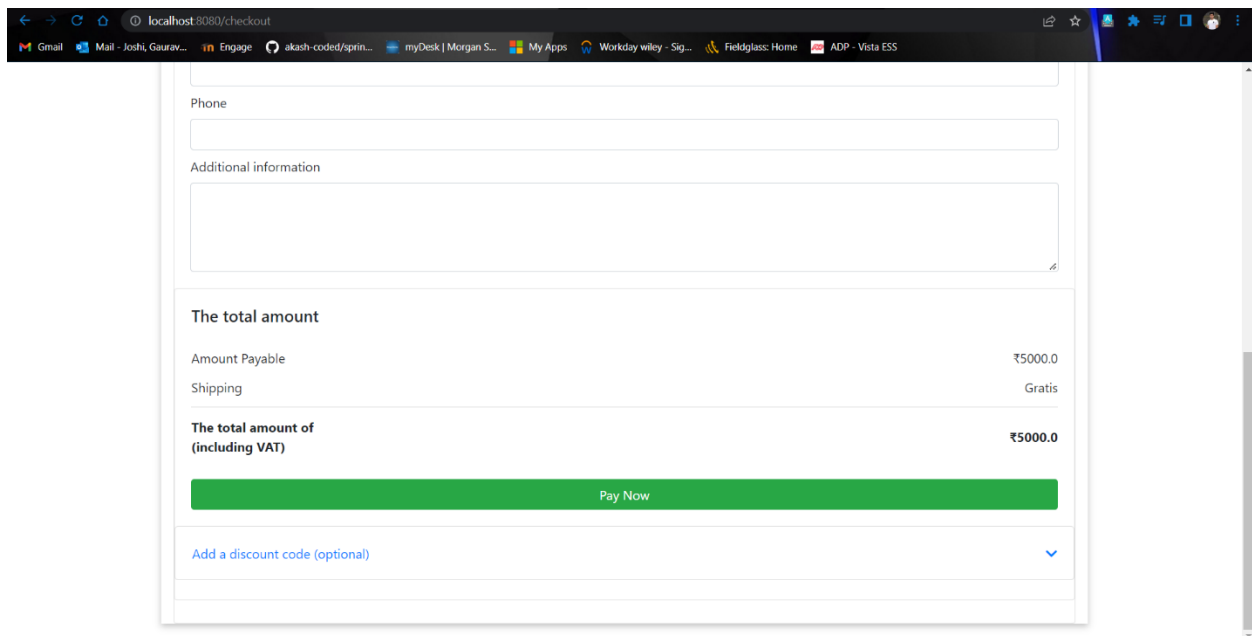
Address (Line 1)

Address (Line 2)

Postcode / ZIP

Town / City

Email address



The screenshot shows the same web browser window, but the form is scrolled down to show the payment summary. The "Phone" field is visible at the top. Below it is the "Additional information" section. The "The total amount" section shows the "Amount Payable" as ₹5000.0 and "Shipping" as "Gratis". The "The total amount of (including VAT)" is ₹5000.0. A green "Pay Now" button is present. At the bottom, there is a link to "Add a discount code (optional)".

Phone

Additional information

The total amount

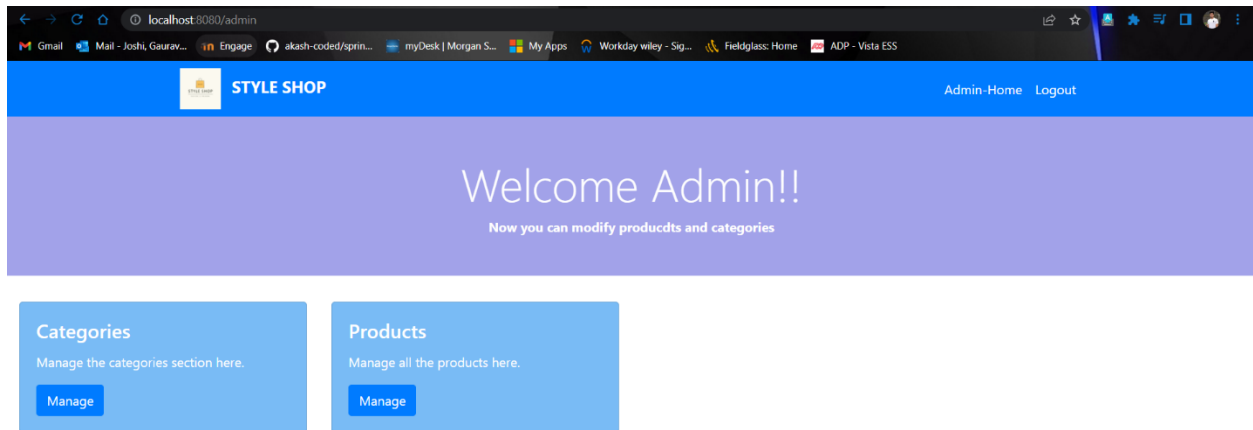
Amount Payable	₹5000.0
Shipping	Gratis
The total amount of (including VAT)	₹5000.0

[Add a discount code \(optional\)](#)

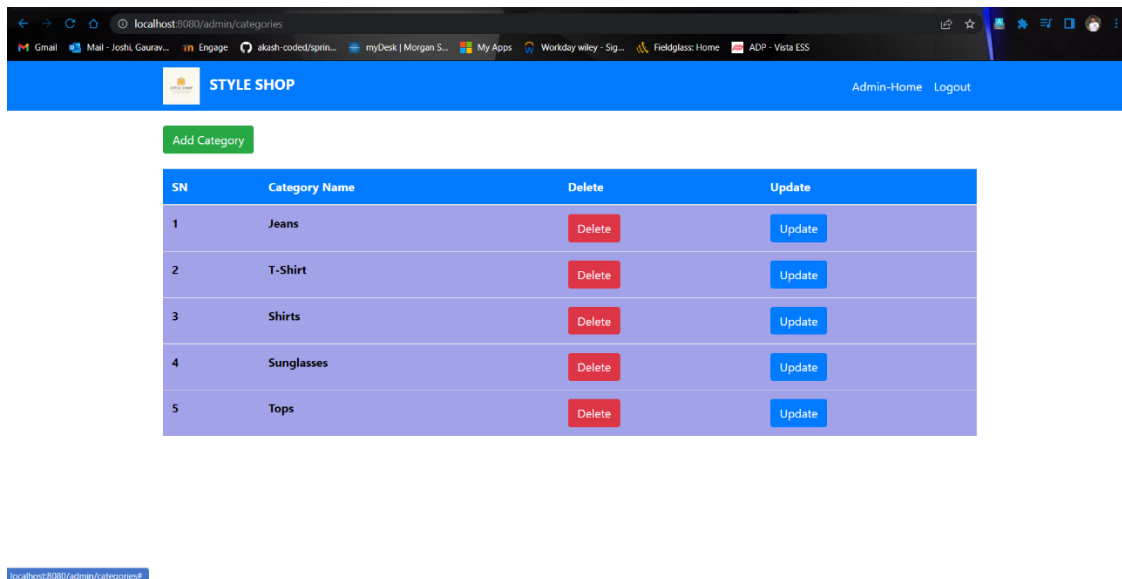
[Pay Now](#)

ADMIN

ADMIN-HOME PAGE:



ADMIN-ADD CATEGORIES



ADMIN-ADD PRODUCTS

localhost:8080/admin/products/add

STYLE SHOP Admin-Home Logout

Add a new Product

Enter name

Choose file Browse

Jeans

0.0





0.0

Submit

localhost:8080/admin/products

STYLE SHOP Admin-Home Logout

Add Product

SN	Product Name	Category	Preview	Delete	Update
1	Levi's Jeans For Men	Jeans		Delete	Update
2	Pepe Jeans (Women)	Jeans		Delete	Update
3	Aviator 3000 Glasses	Sunglasses		Delete	Update
4	Rayban Square Glasses	Sunglasses		Delete	Update