A Project Report on

"La-Firangi(Web-Application)"

Submitted in the partial fulfilment for the award of degree for

Master of Computer Application

Submitted By

Yash Paliwal 2170172

Under the Guidance of:

Dr. Sumit Kumar Tetarave
Asst. Professor, SCA
KIIT, Bhubaneswar

School of Computer Applications

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Bhubaneswar, Odisha

May 2023

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled **La-firangi(Web-Application)** submitted to **School of Computer Application, KIIT University** in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS (MCA)** , is an authentic and original work carried out by Mr/Ms. **Yash Paliwal** with Roll no. **2170172** and Regd. No. **21456781717** under my guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of therequirements of any course of study.

……………………….

Signature of the Student:

Date: ………………..
…………………

...…………………….

Signature of the Guide

Date:

Name,
Designation

School of Computer Application
KIIT University, Bhubaneswar

CERTIFICATE

This is to certify that the project work entitled **La-Firangi** Submitted by (NAME) **YASH PALIWAL** bearing roll no. **2 1 7 0 1 7 2 ,** is an authentic and originalwork.

Signature	Signature
(Internal Examiner)	(External Examiner)
Date..................	Date..................

DECLARATION

I , Student name, roll no do hereby declare that the project report entitled
Yash Paliwal submitted to **School of Computer Application, KIIT University, Bhubaneswar** for the award of the degree of **MASTER OF COMPUTER APPLICATION (MCA) ,** is an authentic and original work carried out by me from 1st Jan 2023 to 1st May 2023 at Coditas name under the guidance of **Dr. Sumit Kumar Tetarave.**

Signature of the student

Date..................

ACKNOWLEDGEMENT

This project is an acknowledgement to the intensity drive and technical competence of many persons who have contributed to it. I express my heartiest gratitude and deepest thanks to project guide **Dr. Sumit Kumar Tetarave** for his proper guidance, suggestions and helping me in completing the project. I am very grateful to the staff and faculty members of the college. I am highly grateful to my parents who have been the source of all kind of financial and emotional encouragement during the course of my work.

Thanking You

Name: **Yash Paliwal**

TABLE OF CONTENTS

1.Introduction

La-Firangi is an E Commerce Web-Application developed for the company of "La-Firangi pvt.ltd" and is indulged in selling a variety of clothes , shoes and many other types of men products . The company operated under the guidance of "Mr. Piyush Malhotra". His wish to expand his business through online market led us to Our project.

Our Website is for selling men products like shirts, shoes and various other type of men products like tie and belts.

Types of Fabric Made from Cotton

While there are definitely other types of materials offered on the market, many people still swear by cotton. That's because cotton is a sturdy fabric and feels soft.
Cotton is also affordable and works perfectly for different types of shirts.
Our Website also have many Clothes of different fabrics made from cotton like:

But did you know that there is more than one type of cotton we can choose from? Let's take a look at the various types of cotton we can make the best T-shirts with:

Combed Cotton: Combed cotton is a bit more expensive than basic cotton but it's also really soft. That's because the cotton fibers are specially treated before they're spun into yarn.

Organic Cotton: Organic cotton production maintains soil fertility.

Pima/Supima Cotton: Supima cotton is 100% American grown. This type of cotton is very soft and durable.
Considered among the highest quality of cotton available, you'll love how it resists pilling, fading, and stretching. After you wash, you'll notice the only change is how much softer it feels.

Slub Cotton: You'll love slub cotton because it's a unique fabric with texture to it. It's also light, airy, and won't cling to your body.

Our Website sells large variety of casual shirts and it also have its own manufacturing. We will sell our products with their own tag.

2.Problem Selection

Scaling any business is not easy, whether it is a brick-and-mortar location or an online business, but an online business has advantages. La-firangi pvt.ltd want to sell their Products online. The main aim of website is to expand business by selling products online.
In present system all business work done manually and it takes very hard to maintain the information of stock of engine oils and Lubricants and take order.
There were many problems in manual system, some of them were:-

- No validation of Data – The clerical if by mistake made a wrong entry; he was never prompted to correct it.
- The major problem was the Relation i.e. whenever an item was added to the item register, the issue register had to be consulted and also a corresponding entry had to be made in the department register.
- The Report generation was the most difficult task and time consuming. Consulting all the registers was a tedious task and required a lot of expertise.
- Wastage of stationary was there.
- No Security of data was there as anyone could access any portion of the registers.
- Highly paid clerical staff.

E-commerce website has many benefits and La-firangi pvt.ltd also wants these benefits like:

- The automation of checkout, billing, payments, inventory management, and other operational processes lowers the number of employees required to run an e-commerce setup.
- This one is a no-brainer. An e-commerce merchant does not need a prominent physical location.
- Our new website will help company to get order and maintain all the details on a digital database. They can add product revoke it and update the status.
- Automated recommendation system using SVM technology

3.Literature Survey

Existing System: The existing system is a manual one. Thereby the three registers – Item register, Issue register, Department register is maintained manually.

La-Firangi pvt.ltd sell their products offline as a wholesaler, retailer and trader. The company is having many products for men. The company want to expand the business through online due to the digital era.

Problem in Manual System:

When La-Firangi pvt.ltd became an established business, they were selling kurta only. Over the years, as their products became in more demand, record keeping using a manual system became tedious. There are many other problem in manual system and these are:

- Data was constantly being duplicated. This was time-consuming and resources were being wasted (For example, the use of stationary).
- Some of the data entered were redundant as there was no repetition check as with a computer.
- The office space had to be increased as more space was needed to store filing cabinets as the amount of paper increased.
- Lack of security as data was stored in filing cabinets. Without the knowledge of products of engine oils and Lubricant, unauthorized users may have had access to confidential information.
- Data was only being accessed by one employee at a time. This was time-consuming. Hence, this led to the business being unproductive.
- Data was constantly being misplaced.
- Risk of data being destroyed due to natural disasters, age, etc.

Proposed System:

The proposed system is fully computerized and has very less problems compared to manual system. The advantages of computerized system are:–

- Security of data is achieved as only member functions are allowed to access private data members.
- Validation of data is now achieved and the operator is not allowed to make a wrong entry.
- Report Generation is made very easy; on a simple key press many reports can be seen.
- No highly paid staff is required only data entry operator is required.
- Automized recommendation for better sales.

4.System Study

4.1 FEASIBILITY STUDY:

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study.

The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below:

• To analyze whether the software will meet organizational requirements.

• To determine whether the software can be implemented using the current technology and within the specified budget and schedule.

• To determine whether the software can be integrated with other existing software.

Types of Feasibility

Various types of feasibility that are commonly considered include technical feasibility, operational feasibility, and economic feasibility.

Technical Feasibility

There are number of technical issues which are generally raised during the feasibility stage of the investigation. They are as follows:-

- Does the necessary technology exist to do at this suggested (and can it be acquired)?
- Does the proposed equipment have the technical capacity to hold the data required to use the new software?
- Can the software be upgraded if developed?
- Are there technical guarantees of accuracy, reliability, ease access of and data security?

Software that can be developed technically and that will be used if installed must still be profitable for organization? Financial benefits must equal or exceed the costs. The analysts raise various financial and economic questions during the preliminary investigation to estimate the following:

- The cost to conduct a full software investigation.
- The cost of hardware and software for the class of application being considered.
- The benefits in the form of reduced costs or fewer costly errors.
- The cost if nothing changes (i.e. the proposed software is not developed).

Economic Feasibility

For any system if the expected benefits equal or exceed the expected costs, the system can be judged to be economically feasible. In economic feasibility, cost benefit analysis is done in which expected costs and benefits are evaluated. Economic analysis is used for evaluating the effectiveness of the proposed system.

Operational Feasibility

Operational feasibility is mainly concerned with issues like whether the system will be used if it is developed and implemented. Whether there will be resistance from users that will effect the possible application benefits? The essential questions that help in testing the operational feasibility of a system are following.

- Does management support the project?
- Are the users not happy with current business practices? Will it reduce the time (operation) considerably? If yes, then they will welcome the change and the new system.
- Have the users been involved in the planning and development of the project? Early involvement reduces the probability of resistance towards the new system.
- Will the proposed system really benefit the organization? Does the overall response increase? Will accessibility of information be lost? Will the system effect the customers in considerable way?

The main objectives of feasibility analysis are :

- To identify the deficiencies in the current software.
- To determine objectives of the proposed software.
- To acquire a sense of scope of the software.
- To identify the responsible user.

5.System Analysis

The decision to acquire computer hardware or software must be handled in the same way as any other business decision. The variety of sizes and types of computing resources available puts a burden on the analyst who must select suitable hardware, software or services and advise the top management-accordingly.

Today, selecting a system is a serious and time-consuming business. The time spent on the selection process is a function of the applications and whether the system is a basic micro- computer or a mainframe. In either case, planning system selection and acquiring experienced help where necessary pay off in the long run.

The selection process consists of several steps, which are discussed below:

1.Requirements analysis: The first step in selection understands the user's requirements within the framework of the organization's objectives and the environment in which the system is being installed.

2. System specifications: System specifications must be clearly defined. These specifications must reflect the actual applications to be handled by the system and include system objectives, flowcharts, input-output requirements, file structure and cost.

3. Request for proposal: After the requirement analysis and system specifications have been defined, a request for proposal is prepared and sent to selected vendors for bidding.

4. Evaluation and validation: The evaluation phase ranks various vendor proposals and determines the one best suited to the user's requirements. It looks into items such as price, availability and technical support. System validation ensures that the vendor can, in fact, match his/her claims, especially system performance.

5. Vendor selection: This step determines the vendor with the best combination of reputation, reliability, services, training, delivery time, lease/finance terms. The selected vendors are invited to give a presentation of their system. The system chosen goes through contract negotiations before implementation.

Thus, System Analysis Means

- Knowing about the current system
- Finding the problem of the system
- Finding the requirement of the system
- System Analysis is a process of gathering and interpreting facts

5.1 System Requirement

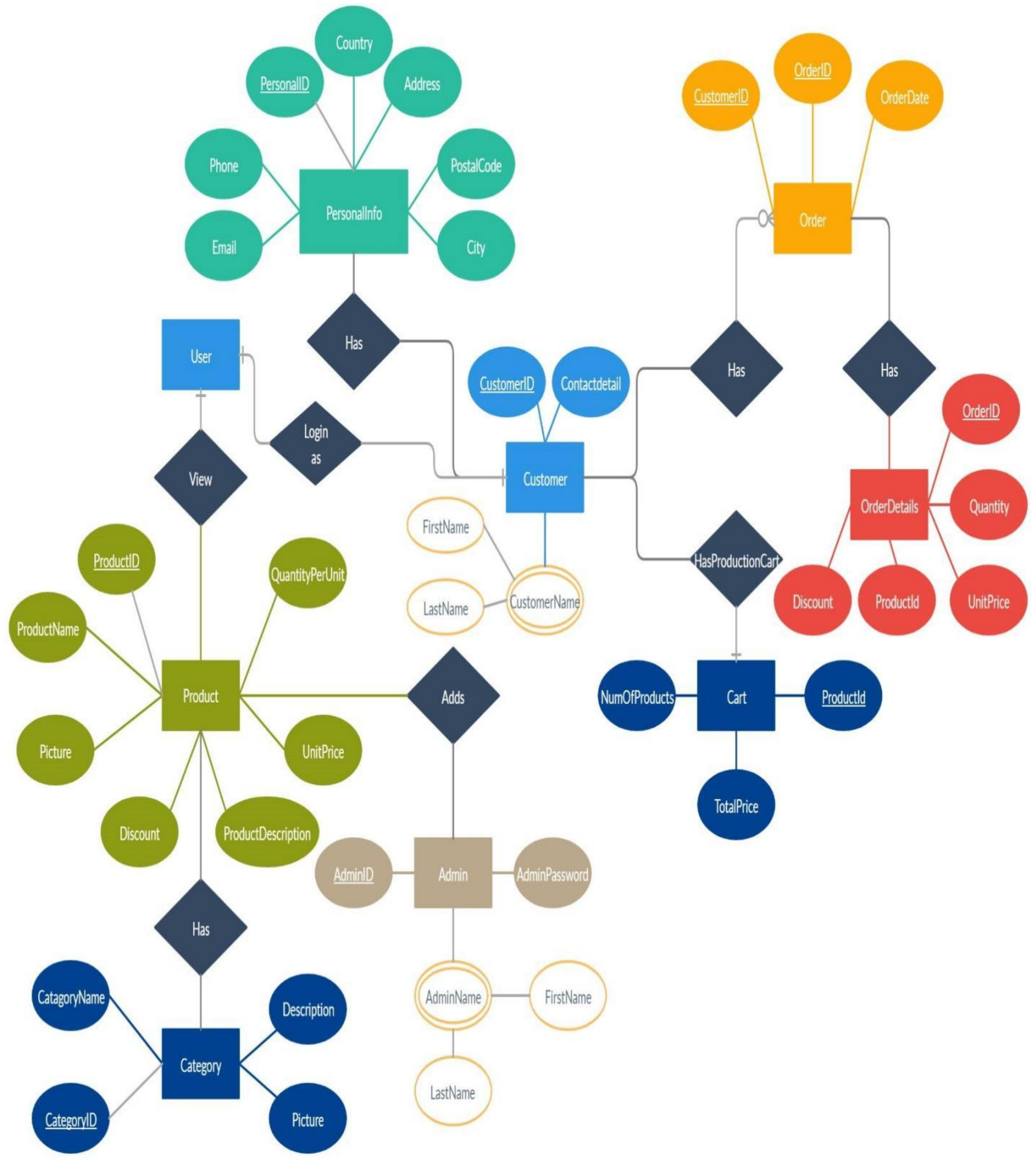
Hardware Requirements

1. Ram	- minimum 1GB
2. Hard disk	-100 MB

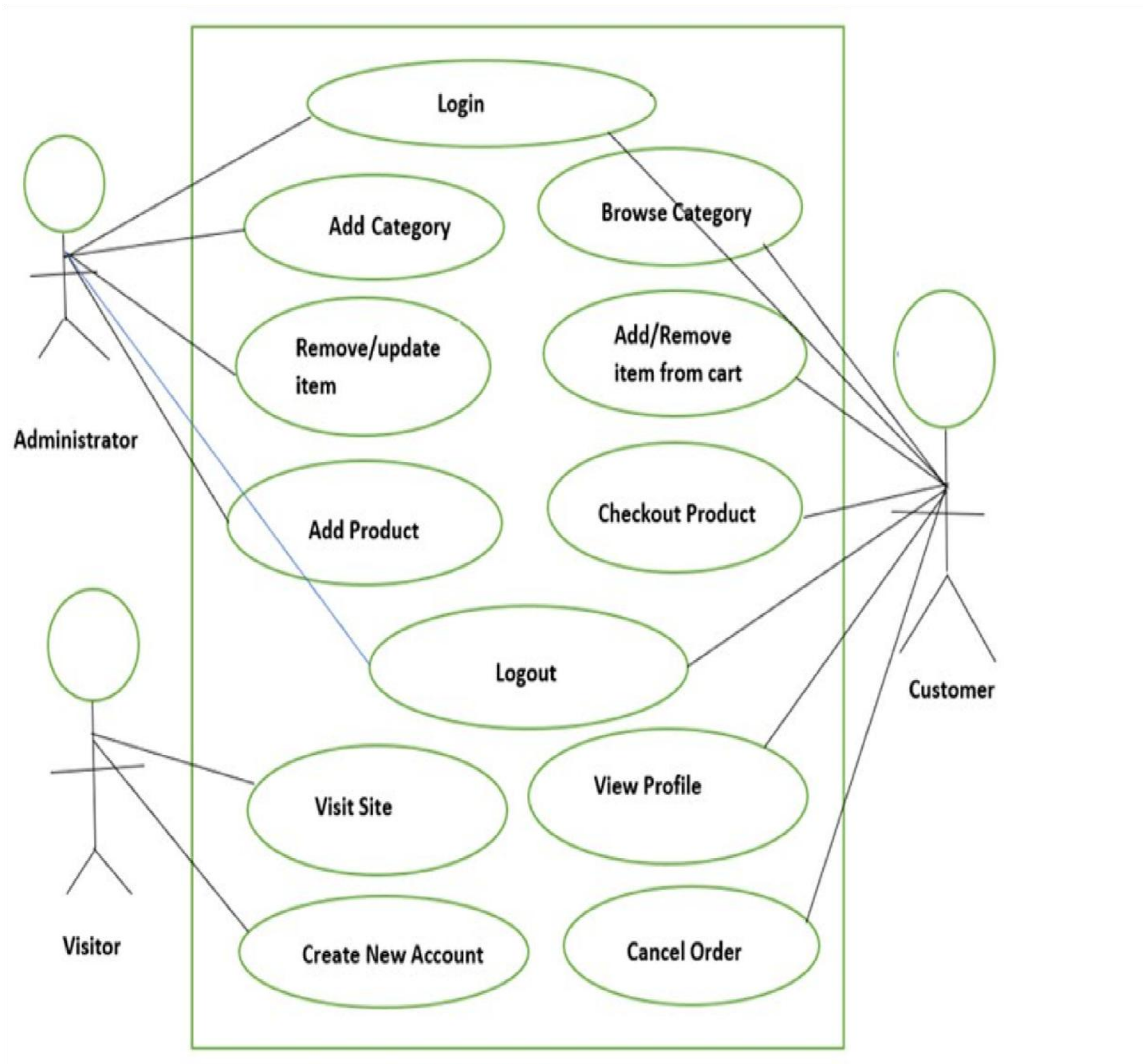
Software Requirements

1. Operating system - Windows
2. Language- Python Programming
3. Framework- Django
4. Database-SQL Lite

5.2 E-R Diagram



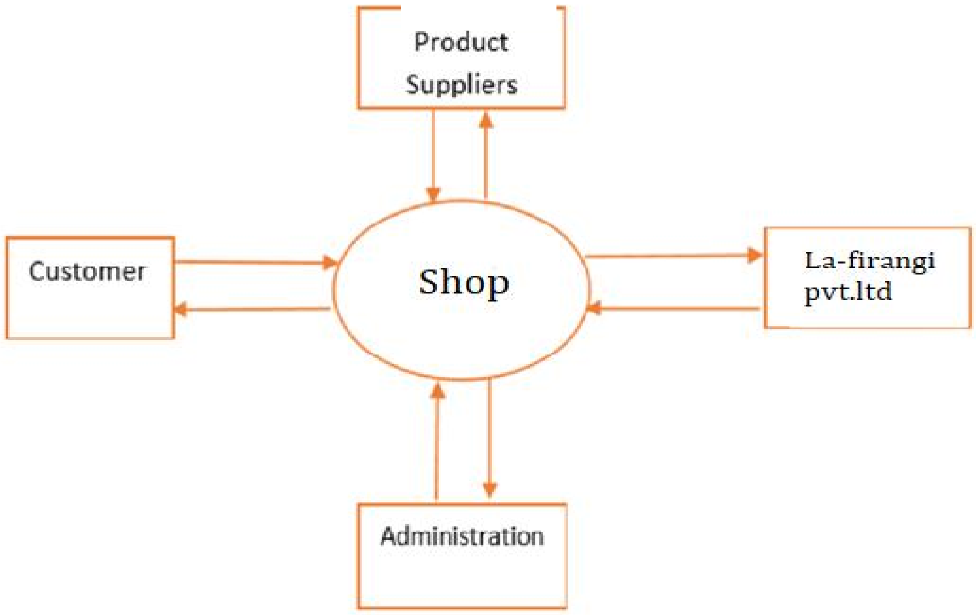
5.3 Use Case Diagram



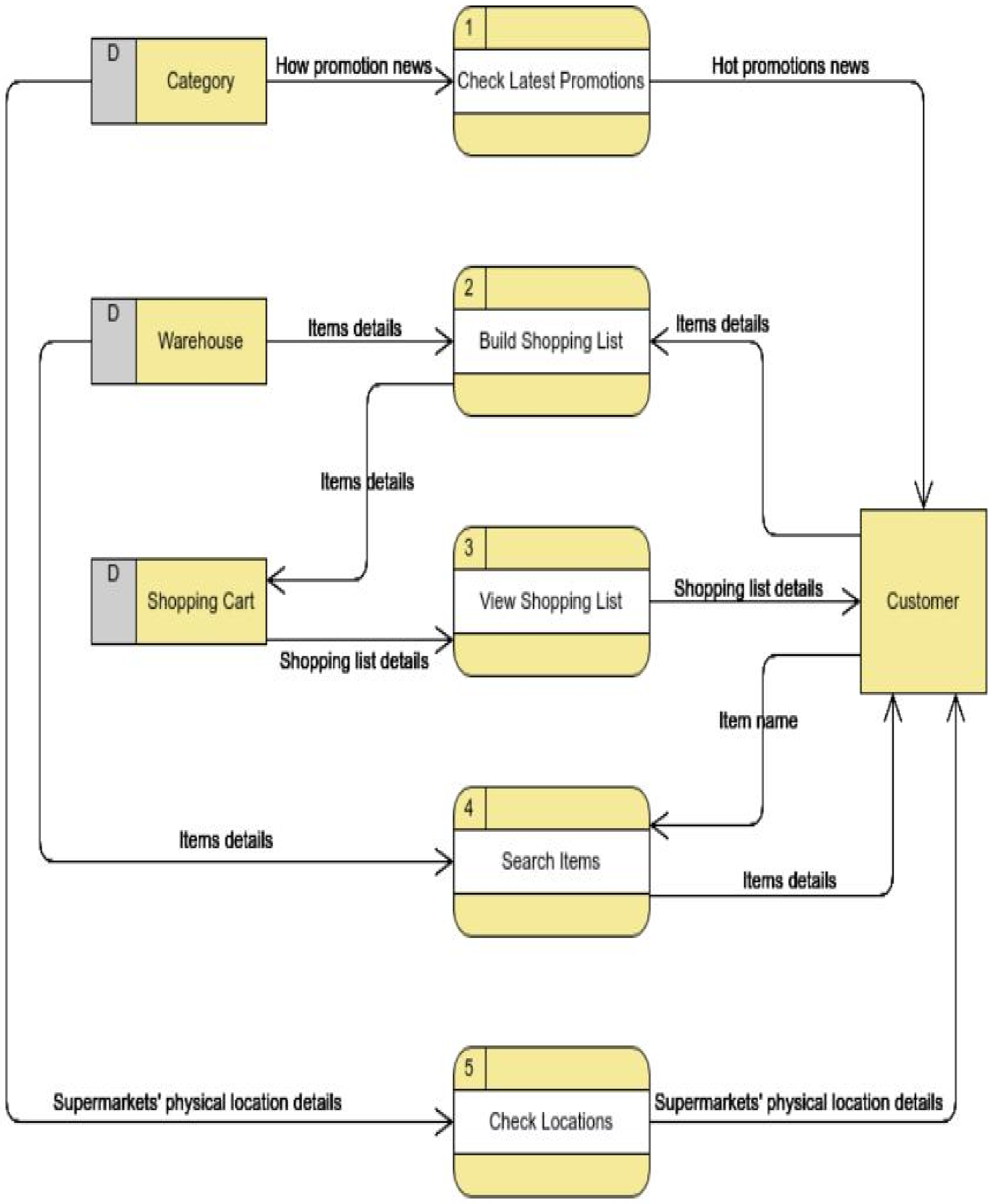
Use Case Diagram

5.4 Data Flow Diagram

Context Level Data Flow Diagram

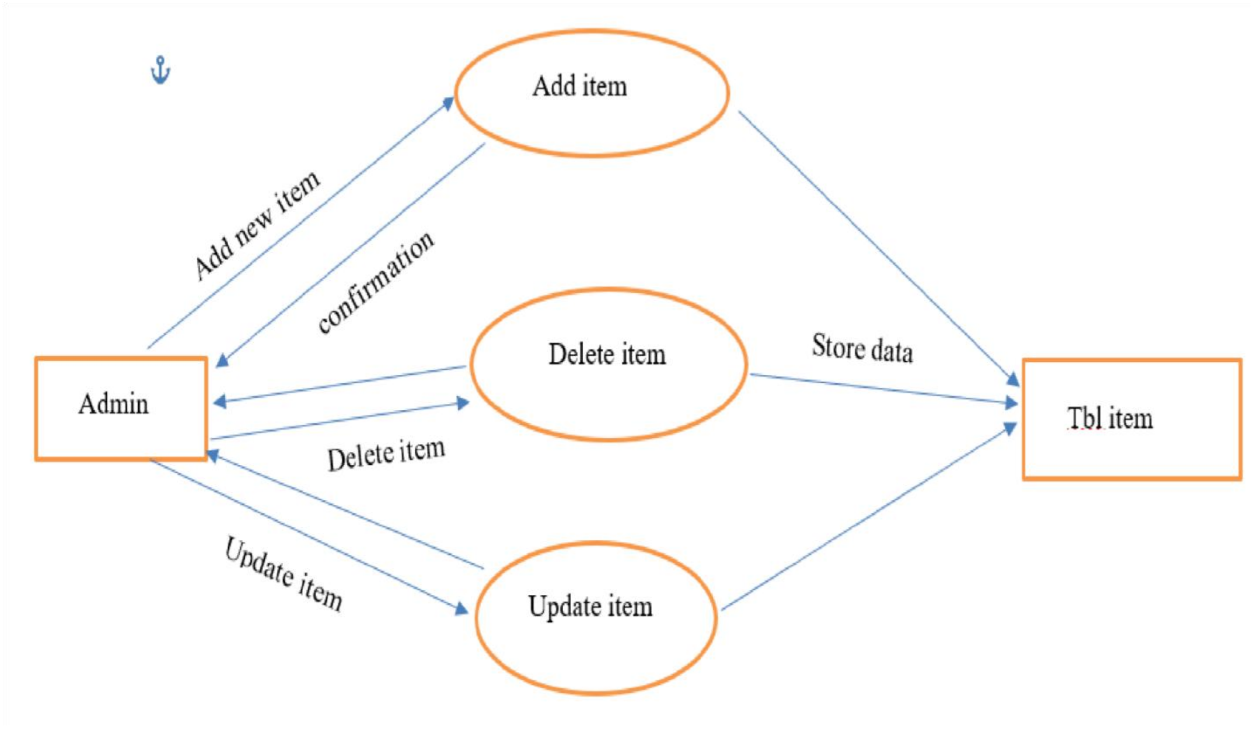


Level 1 Data Flow Diagram

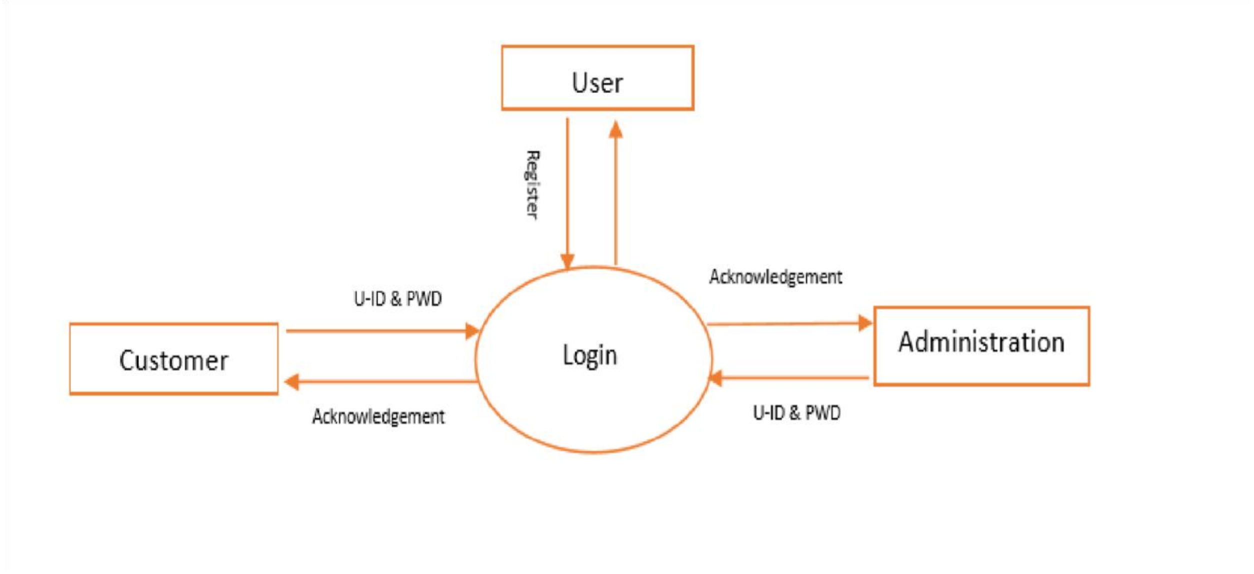


Level 2 Data Flow Diagram

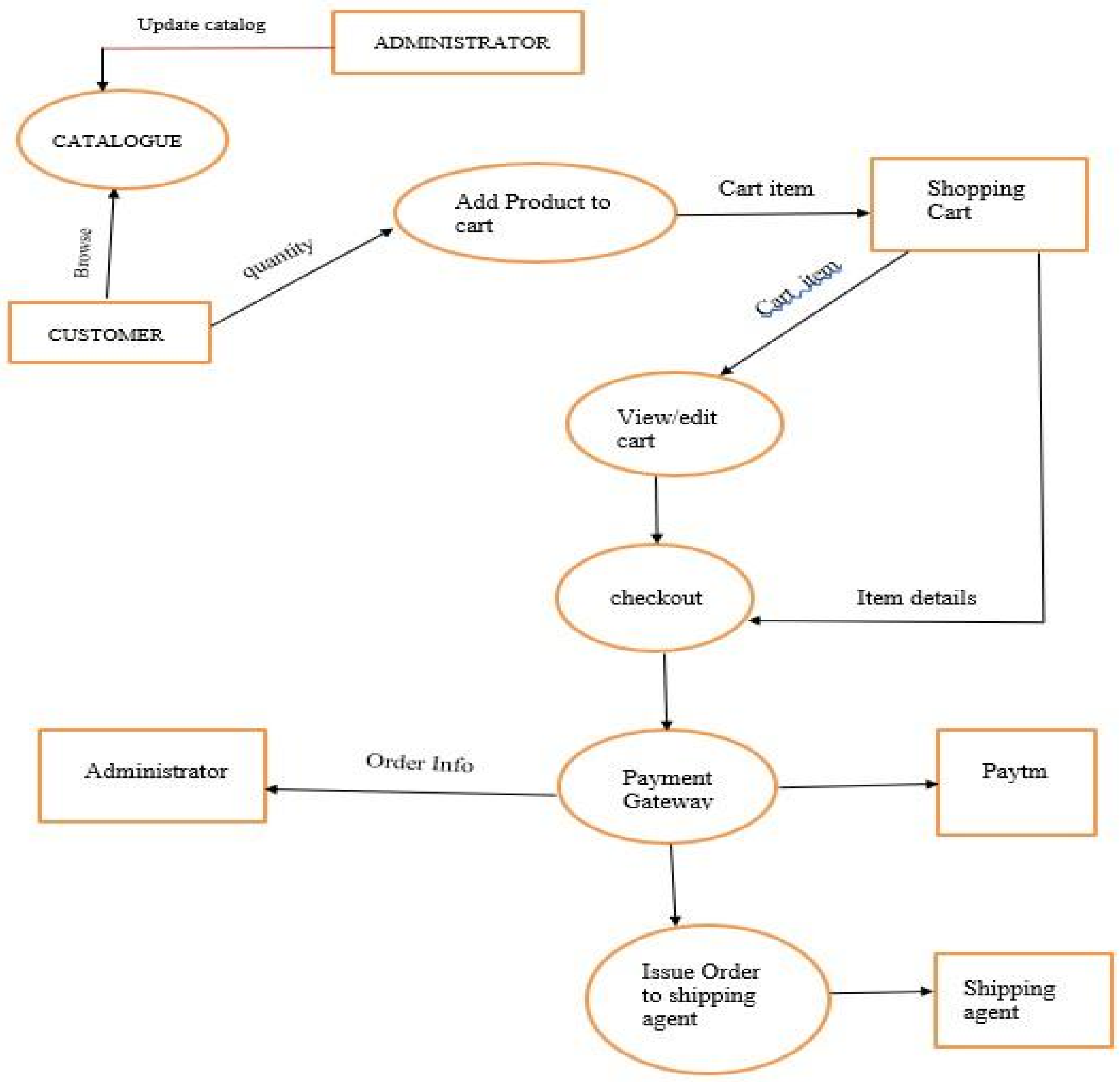
Add, Delete, update item by Admin



Login



Cart and Payment



6.System Design

The systems objectives outlined during the feasibility study serve as the basis from which the work of system design is initiated. Much of the activities involved at this stage are of technical nature requiring a certain degree of experience in designing systems, sound knowledge of computer related technology and thorough understanding of computers available in the market and the various facilities provided by the vendors. Nevertheless, a system cannot be designed in isolation without the active involvement of the user. The users have a vital role to play at this stage too. As we know that data collected during feasibility study will be utilized systematically during the system design. It should, however, be kept in mind that detailed study of the existing system is not necessarily over with completion of the feasibility study. Depending on the plan of feasibility, the level of detailed study will vary and the system design stage will also vary in the amount of investigation that still needs to be done. This investigation is generally an urgent activity during the system design as the designer needs to study minute's details in all aspects of the system. Sometimes, but rarely, this investigation may form a separate stage between Feasibility Study and Computer System Design. Designing a new system is a creative process, which calls for logical as well as lateral thinking. The logical approach involves systematic moves towards the end product keeping in mind the capabilities of the personnel and the equipment at each decision making step. Lateral thought implies encompassing of ideas beyond the usual functions and equipment. This is to ensure that no efforts are being made to fit previous solutions into new situations.

System design considerations

Web development is complicated. In any project, there are a multitude of aspects to consider, from user experience to performance. Development of any enterprise website is a large-scale, complex process, but an eCommerce site presents its own particular challenges. Since users will be coming to the site to learn about and purchase products, developers will want to do everything they can to make this process easy and intuitive. But alongside the considerations of usability, here are a few other aspects of an eCommerce site that developers will want to be sure to consider during the development process:

1. Responsive Design

The use of mobile devices to access websites is continually growing. This means that it is incredibly important to make sure your eCommerce website is optimized for mobile, providing the best experience for users no matter what device they are using to access your site.

Implementing Responsive Design to make a website accessible and usable on every device is important for the success of an eCommerce site. Whatever platform you use, you'll need to be sure you keep mobile users in mind for every aspect of the site, from basic navigation to checkout and payment, since this ever-growing group of users can't be ignored.

2. Support Guest Checkouts

Companies with eCommerce sites will often want to require users to create an account in order to make a purchase, since this allows for follow-up communication that encourages future sales, as well as tracking customers' demographic information to analyze sales. However, it's important to remember that not everybody wants to go through the process of creating an account in order to buy a product. Repeat customers will want to register and get the benefits of having an account, such as saving their information for future purchases and receiving notifications about upcoming sales, but it's still a good idea to provide an option for people who just want to make a one-time order.

3. Site Search Is Important

Statistics show that 30% of visitors to eCommerce sites use search to find the products they are looking for, so it's important to make sure the search functionality is available and easy to use. In addition, it's a good idea to utilize features like autocomplete to help users find popular products or items related to their searches.
Faceted search is another important way to help users find products. This functionality allows them to narrow their search in a variety of ways, including by department, size, price range, manufacturer, etc. Providing this functionality gives users more power to find what they need, letting them limit their searches to exactly what they are looking for.

4. Security Is Essential

All eCommerce sites should support SSL to encrypt information that needs to remain secure. This is especially true for credit card and payment information, but also any customer information like address, phone number, email, etc. Customers have an expectation that their personal information will remain secure when they make a purchase online, so ensuring that SSL is implemented is not just a good idea, but something that is absolutely essential for ensuring that your customers trust that their information will remain secure. In addition, security is required to meet PCI compliance for any business which accepts credit card payments.

5. Optimize Site Performance

If your site is slow, you're likely to lose customers. Statistics show that 40% of users will abandon a website that takes more than 3 seconds to load. This is especially true for mobile users, who are often multi-tasking as they access websites and are more likely to move on to something else if a site is too slow.
In order to keep from losing customers due to slow load times, you'll want to make sure your site is optimized to run as quickly as possible. Here are a few ways to help your site

run more smoothly:
Combining a site’s JavaScript or CSS resource files into single files will speed up their interaction with the site, because users will only have to download one JavaScript file or style sheet rather than five or ten.
Compress images, which will allow them to provide the best visuals at the smallest possible size, reducing download times.
Use caching to reduce the time spent sending data between the web server and the database server.

The designer normally will work under following constraints:

Hardware: The existing hardware will obviously affect the system design

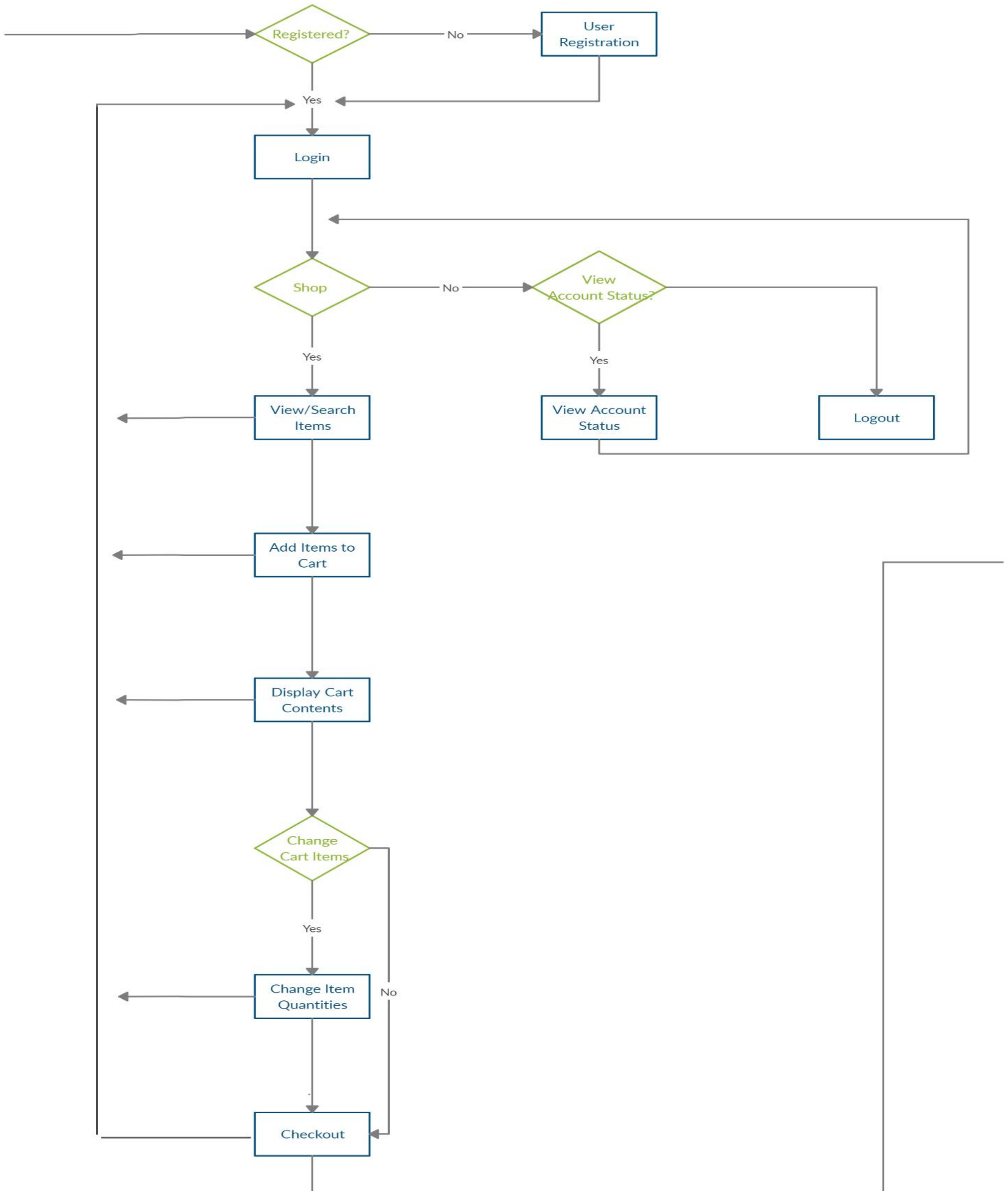
Software: The available software (operating system, utilities, language, etc.) in the market will constrain the design.

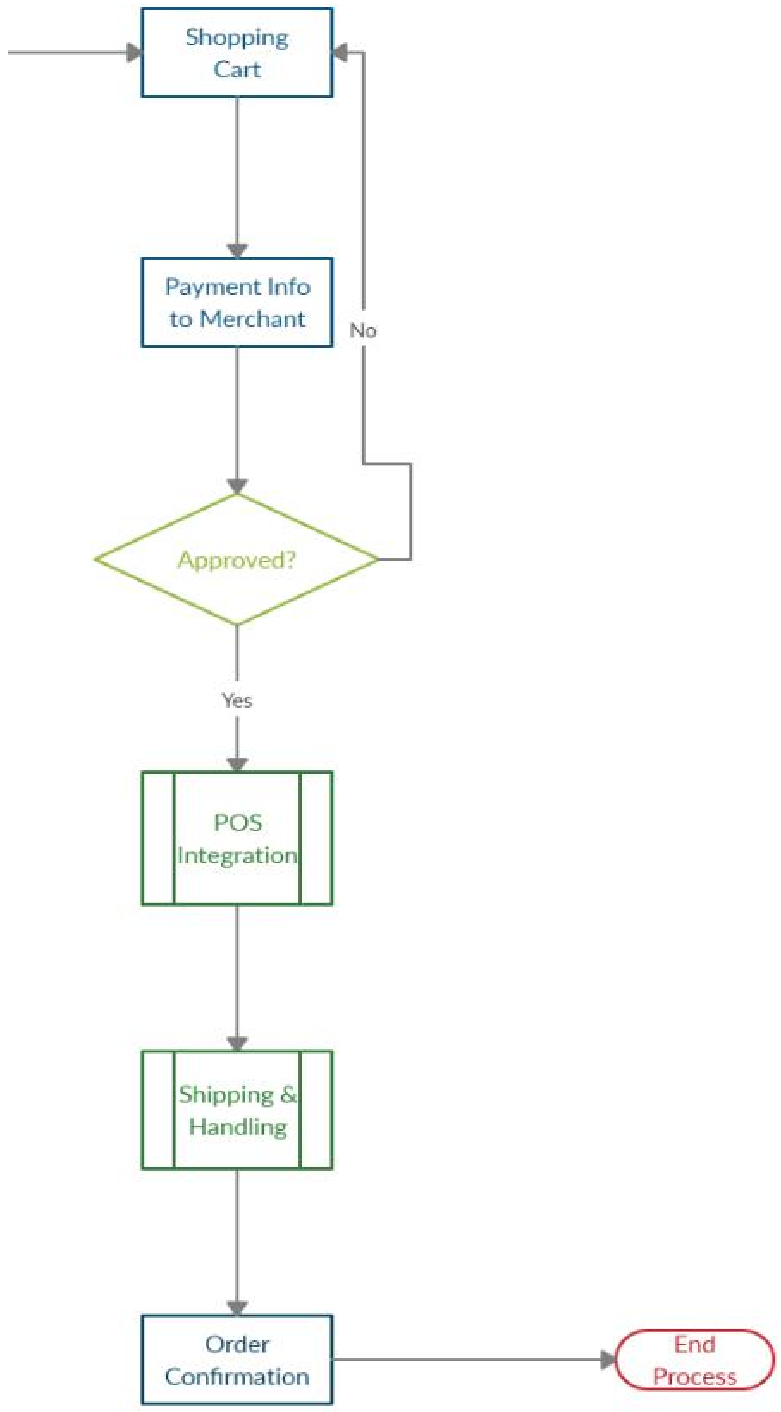
Budget: The budget allocated for the project will affect the scope and depth of design.

Time-scale: The new system may be required by a particular time (e.g. the start of a financial year). This may put a constraint on the designer to find the best design.

Interface with other systems: The new system may require some data from another computerized system or may provide data to another system in which case the files must be compatible in format and the system must operate with a certain processing cycle.

Flowchart


Registered?
No
User Registration
Yes
Login
Shop
No
View Account Status?
Yes
View/Search Items
Yes
View Account Status
Logout
Add Items to Cart
Display Cart Contents
Change Cart Items
Yes
Change Item Quantities
No
Checkout


Shopping Cart
Payment Info to Merchant
No
Approved?
Yes
POS Integration
Shipping & Handling
Order Confirmation
End Process

7. Testing

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software.

Principles of Testing:-

(i) All the test should meet the customer requirements
(ii) To make our software testing should be performed by third party
(iii) Exhaustive testing is not possible.As we need the optimal amount of testing based on the risk assessment of the application.
(iv) All the test to be conducted should be planned before implementing it
(v) It follows pareto rule(80/20 rule) which states that 80% of errors comes from 20% of program components.
(vi) Start testing with small parts and extend it to large parts.

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are –

1. Functional Testing
2. Non-functional Testing

Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

- **Bottom-up integration**

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

- **Top-down integration**

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)

- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Testing E-commerce Websites

Testing E-commerce Websites requires knowledge of web testing techniques and the e-commerce domain.

Most E-commerce Websites share a general common theme and structure, e.g:

- Homepage
- Search Results Page
- Product Details Page
- Order Form Page
- Order Confirmation Page
- Login Form Page and Accounts Pages

Of course, there are many other pages on a typical e-commerce website, but the main core user journey would entail touching the above pages and that's where testing e-commerce websites should focus on: The Checkout Journey.

These "front-end" pages most likely communicate with "back-end" web services, such as Product Search Service, Content Service, Booking Engine, Payment Services, Accounts Services, etc. Therefore, it is important when testing e-commerce websites that we test individual services in isolation as well as integrated as a whole system.

Testing Shopping Cart

Shopping carts are one of the main features of an e-commerce website and thus form the centerpiece of testing e-commerce websites. It allows for customers to select and store multiple items in the cart and purchase them all at once.

Nowadays, shopping carts have become “intelligent” in a sense that they remember what items you store in them so you can retrieve them at a later date or even from another device.

In most cases, cookies are used to store cart data or if the user has an active account and is logged in, a session id can be stored against the user in the database. Either way, there are some key test cases which should be part of testing a shopping cart.

Add one item to the cart – the cart should be updated with the item with correct name, image, and price.

Increase the quantity of the item from the cart – the price should be updated to reflect the correct figure.

Add the same item multiple times – there should be one item in the cart, but the quantity should reflect the number of additions and the total price should reflect the sum of the price of each item.

Add multiple items of different types – For each item added, we should see a corresponding name, image, and price and total price of all items.

Remove some items from the cart – the cart should update showing the existing items in the cart, total price should reflect the new sum.

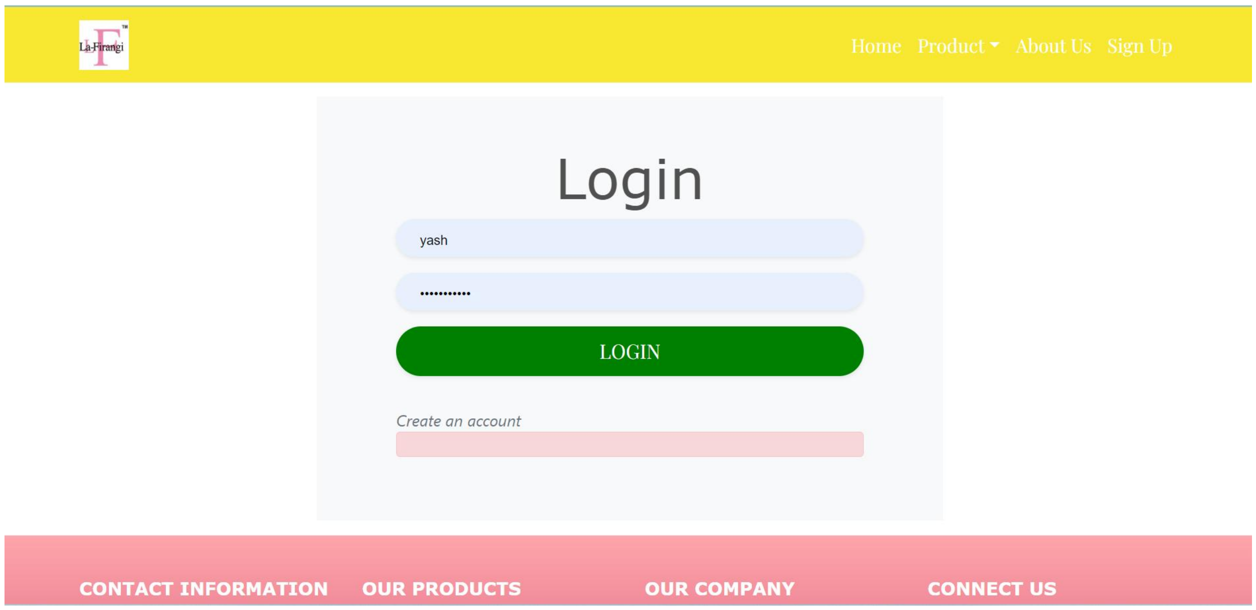
Remove all items from the cart – cart balance should be zero, no items should be displayed in the cart.

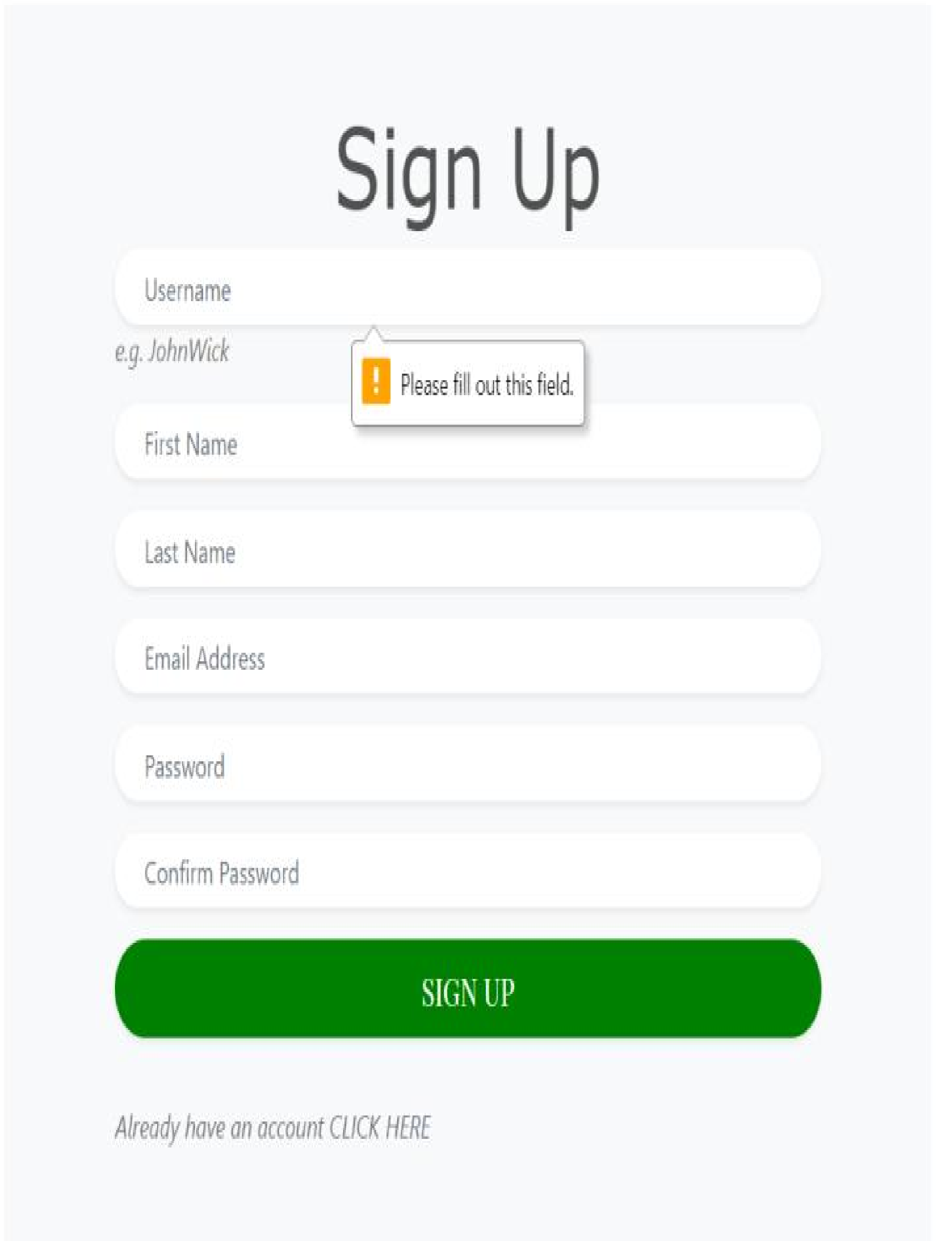
Add item(s) to the cart, close the browser and reopen the same site – ideally, the cart should still hold your items. N.B this particularly depends on the requirements on how the cart should behave.

8. Validation

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.
Validation is the **Dynamic Testing**.
Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing




Sign Up

Username

e.g. JohnWick

Please fill out this field.

First Name

Last Name

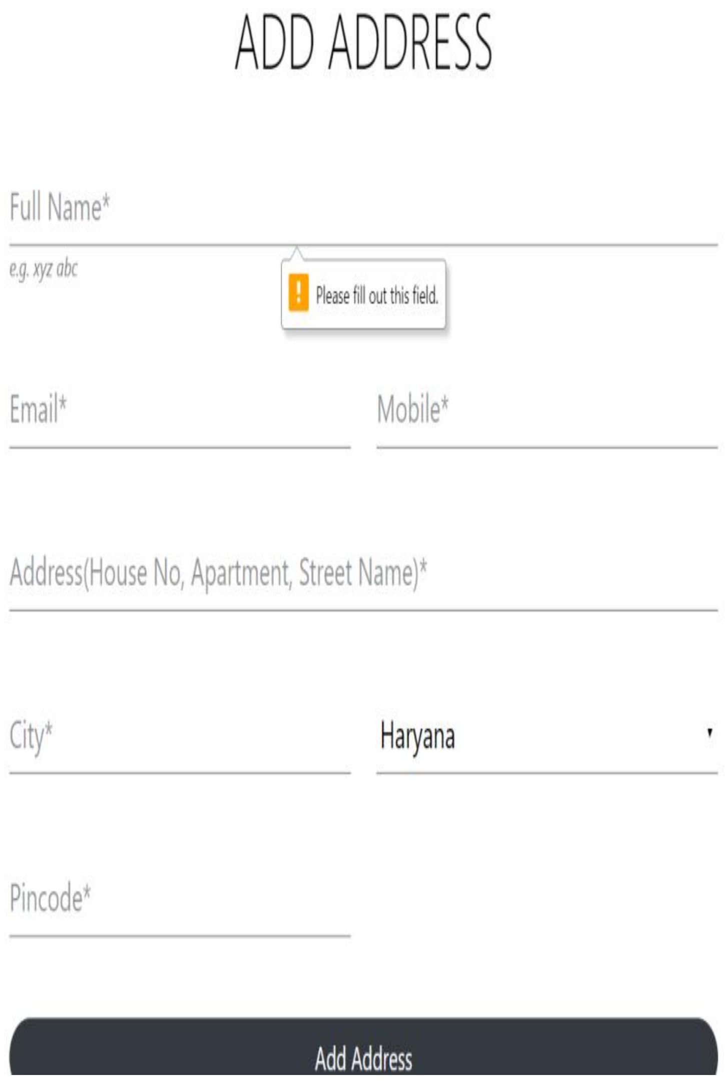
Email Address

Password

Confirm Password

SIGN UP

Already have an account CLICK HERE



ADD ADDRESS

Full Name*

e.g. xyz abc

Please fill out this field.

Email*

Mobile*

Address(House No, Apartment, Street Name)*

City*

Haryana

Pincode*

Add Address

9.Code

Models.py (DataBase Mapping)

```
from django.db import models
from django.contrib.auth.forms import User
from django.contrib.auth.models import User
# Create your models here.
class Category(models.Model):
    cid = models.AutoField
    cname = models.CharField(max_length=30)
    def __str__(self):
        return self.cname

class Size(models.Model):
    sid = models.AutoField
    sname = models.CharField(max_length=5)
    def __str__(self):
        return self.sname

class Product(models.Model):
    id = models.CharField(max_length=30,primary_key=True)
    pid = models.CharField(max_length=30)
    brand = models.CharField(max_length=30,default=None)
    cat = models.ForeignKey(Category,on_delete=models.CASCADE,default=None)
    name = models.CharField(max_length=100)
    description = models.TextField()
    basicPrice = models.IntegerField()
    discount = models.IntegerField()
    price = models.IntegerField()
    size = models.ForeignKey(Size,on_delete=models.CASCADE,default=None)

    img1 = models.ImageField(upload_to='images')
    img2 = models.ImageField(upload_to='images',default=None)
    date = models.DateTimeField(auto_now_add=True)
    update = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.id
```

```
class Cart(models.Model):
   cartid=models.AutoField
   cart_user=models.ForeignKey(User,on_delete=models.CASCADE,default=None)

cart_product=models.ForeignKey(Product,on_delete=models.CASCADE,default=None)
   count=models.IntegerField(default=1)
   total=models.IntegerField()
   date=models.DateTimeField(auto_now_add=True)
   update=models.DateTimeField(auto_now_add=True)

   def __str__(self):
      return self.cart_product.id

class Checkout(models.Model):
   checkid=models.CharField(max_length=30,primary_key=True,default=None)
   checkout_user      =      models.ForeignKey(User,      on_delete=models.CASCADE,
default=None)
   chname=models.CharField(max_length=30)
   mobile=models.IntegerField()
   email=models.EmailField(max_length=50)
   state=models.CharField(max_length=30)
   city=models.CharField(max_length=30)
   address=models.CharField(max_length=50)
   pin=models.CharField(max_length=10)

   def __str__(self):
      return self.checkid

class Order(models.Model):
   orderid =  models.AutoField
   ordernumber = models.IntegerField()
   order_user=models.ForeignKey(User,on_delete=models.CASCADE,default=None)

order_product=models.ForeignKey(Product,on_delete=models.CASCADE,default=None)
   count=models.IntegerField(default=1)

order_address=models.ForeignKey(Checkout,on_delete=models.CASCADE,default=No
ne)
   def __str__(self):
      return self.order_address.chname
```

```
class PreviousOrder(models.Model):
   orderid =  models.AutoField
   ordernumber = models.IntegerField()
   order_user=models.ForeignKey(User,on_delete=models.CASCADE,default=None)

order_product=models.ForeignKey(Product,on_delete=models.CASCADE,default=None)
   count=models.IntegerField(default=1)

order_address=models.ForeignKey(Checkout,on_delete=models.CASCADE,default=No
ne)
   def __str__(self):
      return self.order_address.chname

class CancelOrder(models.Model):
   orderid =  models.AutoField
   ordernumber = models.IntegerField()
   order_user=models.ForeignKey(User,on_delete=models.CASCADE,default=None)

order_product=models.ForeignKey(Product,on_delete=models.CASCADE,default=None)
   count=models.IntegerField(default=1)

order_address=models.ForeignKey(Checkout,on_delete=models.CASCADE,default=No
ne)

   def __str__(self):
      return self.order_address.chname

class ReturnOrder(models.Model):
   orderid =  models.AutoField
   ordernumber = models.IntegerField()
   order_user=models.ForeignKey(User,on_delete=models.CASCADE,default=None)

order_product=models.ForeignKey(Product,on_delete=models.CASCADE,default=None)
   count=models.IntegerField(default=1)

order_address=models.ForeignKey(Checkout,on_delete=models.CASCADE,default=No
ne)
```

```
    def __str__(self):
        return self.order_address.chname
```

Url.py (Url Mapping)

```
from django.contrib import admin
from django.urls import path
from automobile import settings
from django.contrib.staticfiles.urls import staticfiles_urlpatterns,static
from vendor import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.home,name='home'),
    path('addproduct/', views.addProduct),
    path('addcategory/', views.addCategory),
    path('shop/<str:cn>/',views.Shop,name='shop'),
    path('shop2/<str:si>/',views.Shop2),
    path('productdetails/<str:num>/', views.ProductDetails,name='productdetail' ),
    path('productdetails2/<str:num>/<str:s>/',
views.ProductDetails2,name='productdetail2'),
    path('register/', views.register,name='register'),
    path('signup/', views.SignUp),
    path('logout/',views.logOut),
    path('adminpage/',views.AdminPage,name="adminpage"),
    path('sizeavailable/<str:num>/', views.Sizeavi),
    path('sizeadd/<str:num>/', views.SizeAdd),
    path('delete/<str:num>/',views.DeleteProduct,name='deleteproduct'),
    path('delete2/<str:num>/',views.DeleteAddress,name='deleteaddress'),
    path('edit/<str:num>/',views.editProduct,name='edit'),
    path('cart/',views.CartDetails,name='cart'),
    path('orderplaced/<str:num>/',views.OrderPlaced,name='cart'),
    path('selectaddress/',views.SelectAddress,name='cart'),
    path('cartdelete/<str:num>/',views.CartDelete),
    path('cartedit1/<str:num>/',views.CartEdit1),
    path('cartedit/<str:num>/',views.CartEdit),
    path('checkout/<str:num>/',views.CheckoutForm,name='checkout'),
    path('addaddress/',views.AddAddress),
    path('addaddress2/',views.AddAddress2),
    path('pastorder/',views.PastOrders),
    path('Previousorders/',views.PastOrders2),
    path('cancelorder/<int:num>/',views.Cancelorder),
    path('cancelorder/',views.CancelOrderAdmin,name='cart'),
    path('about/',views.About,name = 'about'),
    path('orderadmin/',views.OrderAdmin),
    path('deletecart/',views.deletecart),
    path('deleteshop/',views.deleteshop),
    path('deleteabout/',views.deleteabout),
    path('deletelogout/',views.deletelogout),
    path('deletepastorder/',views.deletepastorder),
    path('deleteadminpage/',views.deleteadminpage),
    path('deletePreviousorders/',views.deletePreviousorders),
    path('dispatchedorder/<int:num>/',views.DispatchedOrder),
    path('homedelete/',views.homedelete),
    path('returnorder/<int:num>/',views.Returnorder),
    path('returnorder/',views.ReturnOrderAdmin),
    path('cancelreturn/<int:num>/',views.CancelReturn),
    path('confirmreturn/<int:num>/',views.ConfirmReturn),
    path('deletecancel/<int:num>/',views.Deletecancel),

]



urlpatterns=urlpatterns+staticfiles_urlpatterns()
urlpatterns=urlpatterns+static(settings.MEDIA_URL,document_root=settings.MEDIA_R
OOT)
```

Views.py (Business Logic)

```
from django.shortcuts import render
from . import forms
from django.contrib.messages import success,error
from django.shortcuts import HttpResponseRedirect
from django.db.models import Q
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
import pandas as pd
```

```
from django.contrib.auth.forms import User
from django.contrib.auth.decorators import login_required
from django.contrib import auth
from django.core.mail import send_mail
from django.core.paginator import Paginator,EmptyPage,PageNotAnInteger
from vendor.models import *
from mart import settings

product_df = pd.read_csv('products.csv')

# Create a bag-of-words vectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(product_df['description'])

# Create an SVM object with a linear kernel
clf = svm.SVC(kernel='linear')

# Create a feature matrix 'X' from the input variables
X = product_df[['basicPrice', 'discount', 'price']].values
clf.fit(X, product_df['recommended'])

def home(request):
    cat = Category.objects.all()
    siz = Size.objects.all()

    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr)|
Q(pid__contains=sr),size__sname=sr)
        datacount=data.count()
        noData = ""
        if(datacount == 0):
            noData="No Such product found"
            data=Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return
render(request,'shop.html',{"Data":data,"Cat":cat,"Siz":siz,"posts":product_list,"No
":noData})

    abc = [None] * 6
    count=0
    data = Product.objects.all()

    for i in data:
        abc[count]= i.id
        count=count+1
    cat = Category.objects.all()
    return render(request, 'index.html', {"Data":data,"Cat":cat,"Abc":abc})

def response(pid):
    product_description = product_df.loc[product_df['pid'] == pid, 'description'].values[0]

    # Preprocess the data using the vectorizer
    X_new = vectorizer.transform([product_description])

    # Make a prediction using the trained SVM model
    predictions = clf.predict(X_new)

    # Get the indices of the recommended products
    recommended_indices = predictions.nonzero()[1]

    # Get the recommended products
    recommended_products = product_df.loc[recommended_indices]

    # Convert the recommended products to a JSON response
    response_data = {
        'products': recommended_products.to_dict(orient='records')
    }
    return response_data


def addProduct(request):
    if(request.method=='POST'):
        data = forms.ProductForm(request.POST, request.FILES)
        if(data.is_valid()):
            data.save()
            success(request,"Product Added")
            return HttpResponseRedirect('/addproduct/')
        else:
            error(request,'Invalid Product Detail')
            return HttpResponseRedirect('/addproduct/')
    else:
        return render(request,'addproduct.html', {'Form' : forms.ProductForm})

def addCategory(request):
    if(request.method=='POST'):
        data = forms.ProductForm(request.POST, request.FILES)
        if(data.is_valid()):
            data.save()
            success(request,"Product Added")
            return HttpResponseRedirect('/addproduct/')
        else:
            error(request,'Invalid Product Detail')
            return HttpResponseRedirect('/addproduct/')
    else:
        return render(request,'addproduct.html', {'Form' : forms.ProductForm})




def email_send(request,email,name):
    subject = 'Thanks '+name+' for registering to our site'
    message = ' it  means a lot to us '
    email_from = settings.EMAIL_HOST_USER
    recipient_list = [email,]
    send_mail( subject, message, email_from, recipient_list )

def dispatch_email(request,data):

    subject = 'Order Dispached'
    message = 'Dear '+data.order_address.chname+',\n        Your Product is being
dispatched for our side and will reached soon.\nAt address:
```

```
\n'+data.order_address.address+"\n"+data.order_address.pin
    email_from = settings.EMAIL_HOST_USER
    recipient_list = [data.order_address.email,]
    send_mail( subject, message, email_from, recipient_list )


def Cancel_email(request,data):

    subject = 'Order Dispached'
    message = 'Dear '+data.order_address.chname+",\n        Your Request of Product
Cancelation is registered "
    email_from = settings.EMAIL_HOST_USER
    recipient_list = [data.order_address.email,]
    send_mail( subject, message, email_from, recipient_list )

def register(request):
    if(request.method=='POST'):
        lname=request.POST.get('usernam')
        lpward=request.POST.get('passwrd')
        user=auth.authenticate(username=lname,password=lpward)
        if(user is not None):
            auth.login(request,user)
            if(user.is_superuser):
                return HttpResponseRedirect('/')
            else:
                return HttpResponseRedirect('/')
        else:
            error(request,"Invalid User")
    return render(request,'Login.html')

def SignUp(request):
    if(request.method=='POST'):
        unam=request.POST.get('uname')
        try:
            match = User.objects.get(username=str(unam))
            if (match):
                error(request, "Username Already Exist")

        except:
            fnam = request.POST.get('first_name')
            lnam = request.POST.get('last_name')
            mail = request.POST.get('email')
            pward = request.POST.get('pward')
            cpward = request.POST.get('cpward')
            if (pward == cpward):
                User.objects.create_user(username=str(unam),
                                         first_name=str(fnam),
                                         last_name=str(lnam),
                                         email=mail,
                                         password=pward
                                         )
                success(request, "Account is created")
                try:
                    email_send(request, mail, unam)
                except:
                    error(request, "Mail not send")
                return HttpResponseRedirect('/register/')
            else:
                error(request, "Password and Confirm Password not Matched")
    return render(request, "signup.html")


def Shop(request,cn):

    cat = Category.objects.all()
    siz = Size.objects.all()
    noData = ""
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr)|
Q(pid__contains=sr),size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return
render(request,'shop.html',{"Data":data,"Cat":cat,"Siz":siz,"posts":product_list,"No
":noData})

    if (cn == "sample"):
        data = Product.objects.all()
    else:
        data = Product.objects.filter(cat__cname=cn)

    if(len(data)==0):
        noData="Product In This Category Is Not Available"
        data = Product.objects.all()
   # post = Product.objects.all()
    paginator = Paginator(data, 8)
    page = request.GET.get('page')
    try:
        product_list = paginator.page(page)
    except PageNotAnInteger:
        product_list = paginator.page(1)
    except EmptyPage:
        product_list = paginator.page(paginator.num_pages)
    return render(request,"shop.html",{"Cat":cat,"Data":data,"Siz":siz,
                                       "No":noData,"posts":product_list})

def Shop2(request,si):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list,"No":noData})

    noData = ""
    if (si == "sample"):
        data = Product.objects.all()
    else:
        data = Product.objects.filter(size__sname=si)
    if (len(data)==0):
        noData ="Product In This Size Is Not Available"
        data = Product.objects.all()
    paginator = Paginator(data, 8)
    page = request.GET.get('page')
    try:
        product_list = paginator.page(page)
    except PageNotAnInteger:
        product_list = paginator.page(1)
    except EmptyPage:
        product_list = paginator.page(paginator.num_pages)
    return
render(request,"shop.html",{"Cat":cat,"Data":data,"Siz":siz,"No":noData,"posts":pr
oduct_list})

def ProductDetails(request,num):
    data = Product.objects.get(id=num)
    dat = Product.objects.filter(pid=data.pid)
    car = Cart.objects.all();
    avail = ""
    sicount = dat.count()
    if (sicount == 0):
        avail = "Out Of Stock"
    else:
        avail = "In Stock"

    if (request.method == 'POST'):
        print("yash")
        form = forms.CartForm(request.POST)
        print(form.is_valid())
        q = request.POST['count']
        z=0
        if (request.user == None):
            HttpResponseRedirect('/register/')
        if (form.is_valid()):
            print("yash")
            for x in car:

                if (data.id == x.cart_product.id):
                    x.count = x.count + 1
                    z = 1
                    x.save()
                    return HttpResponseRedirect('/cart/')
            if (z == 0):
                f = form.save(commit=False)
                f.cart_user = request.user
                f.cart_product = data
                f.count = 1
                f.total = int(data.price) * float(q)
                f.save()
                return HttpResponseRedirect('/cart/')
    else:
        form = forms.CartForm()
    car = Cart.objects.all();
    siz = Size.objects.all();

    return render(request, 'productdetails.html', {"Data": data,"Siz":siz,"Form":form,"Dat":dat,"Avail":avail,"Count":sicount})

@login_required(login_url='/register/')
def ProductDetails2(request,num,s):
    dat = Product.objects.filter(pid=num)
    avail = ""
    sicount = dat.count()
    if (sicount == 1):
        avail = "Out Of Stock"
    else:
        avail = "In Stock"
    data = Product.objects.get(id=num)
    num=str(num)+"-"+str(s)
    dat = Product.objects.filter(pid=s)
    car = Cart.objects.all();
    siz = Size.objects.filter(sname=s)
    if (request.method == 'POST'):
        form = forms.CartForm(request.POST)
        q = request.POST['count']
        z=0
        if(request.user == None):

            HttpResponseRedirect('/register/')
        if (form.is_valid()):
            for x in car:

                if (data.id == x.cart_product.id):
                    x.count = x.count + 1
                    z = 1
                    x.save()
                    return HttpResponseRedirect('/cart/')
            if (z == 0):

                f = form.save(commit=False)
                f.cart_user = request.user
                f.cart_product = data
                f.count = q
                f.total = int(data.price) * float(q)
                f.save()
                return HttpResponseRedirect('/cart/')
    else:
        form = forms.CartForm()
    return render(request, 'productdetails.html', {"Data": data, "Siz": siz,"Dat":dat,
"Form": form,"Avail":avail})
```

```
def logOut(request):
    auth.logout(request)
    return HttpResponseRedirect('/')

def addProduct(request):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        try:

            data = Product()
            s = request.POST.get('size')
            st = Size.objects.get(sname=s)
            data.size = st
            data.pid = request.POST.get('id')

            data.id = request.POST.get('id')+"_"+str(st)
            cn = request.POST.get('cat')
            ct = Category.objects.get(cname=cn)
            data.cat = ct



            data.name = request.POST.get('name')
            data.description = request.POST.get('description')
            data.basicPrice = request.POST.get('basicPrice')
            data.discount = request.POST.get('discount')
            data.brand = request.POST.get('brand')

            bp = int(data.basicPrice)
            d = int(data.discount)

            data.price = int(bp - (bp * d / 100))
            data.img1 = request.FILES.get('img1')
            data.img2 = request.FILES.get('img2')

            data.save()
            success(request, 'Product Inserted')
            return HttpResponseRedirect('/addproduct/')
        except:
            error(request, "Invalid Record")
    return render(request, "addproduct.html", {"Cat": cat,"Siz": siz})

def AdminPage(request):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list,"No":noData})

    data=Product.objects.all()
    return render(request,"admins.html",{"Data":data})

def Sizeavi(request,num):
    data=Product.objects.filter(pid=num)
    return render(request,"sizeavi.html",{"Data":data})

def addCategory(request):
    if (request.method == 'POST'):
        try:

            data = Category()
            data.cname= request.POST.get('name')
            data.save()
            success(request, 'Product Inserted')
            return HttpResponseRedirect('/addcategory/')
        except:
            error(request, "Invalid Record")
    return render(request, "addcategory.html")


def SizeAdd(request,num):
    data = Product.objects.get(id=num)
    siz = Size.objects.all()
    cat = Category.objects.all()

    if (request.method == 'POST'):
        try:

            s = request.POST.get('size')
            st = Size.objects.get(sname=s)
            data.size = st

            data.id=str(request.POST.get('id'))+"_"+str(st)
            data.name = request.POST.get('name')
            data.description = request.POST.get('description')
            data.basicPrice = request.POST.get('basicPrice')
            data.discount = request.POST.get('discount')

            bp = int(data.basicPrice)
            d = int(data.discount)

            data.price = bp - bp * d / 100
            data.color = request.POST.get('color')
            data.save()
            success(request, 'Size Is Added')
            data = Product.objects.get(id=num)
        except:
            error(request, "Invalid Record")
    return render(request, "sizeadd.html", {"Data": data,"Siz":siz})
```

```
def editProduct(request,num):
    data=Product.objects.get(id=num)
    cat=Category.objects.all()
    if (request.method == 'POST'):
        try:

            data.name = request.POST.get('name')
            data.description = request.POST.get('description')
            data.basicPrice = request.POST.get('basicPrice')
            data.discount = request.POST.get('discount')
            bp = int(data.basicPrice)
            d = int(data.discount)

            data.price = bp - bp * d / 100
            data.color = request.POST.get('color')
            data.save()
            success(request, 'Product Edited')
            data = Product.objects.get(id=num)
        except:
            error(request, "Invalid Record")
    return render(request,"edit.html",{"Data":data})

def DeleteProduct(request,num):
    data=Product.objects.filter(pid=num)
    for i in data:
        i.delete()
    return HttpResponseRedirect("/admins/")
def DeleteAddress(request,num):
    adata=Checkout.objects.get(checkid=num)
    adata.delete()
    adata = Checkout.objects.filter(checkout_user=request.user)
    if (len(adata) == 0):
        return HttpResponseRedirect('/addaddress/')
    return render(request, "selectaddress.html", {"Data": adata})


@login_required(login_url='/register/')
def CartDetails(request):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.filter(size__sname=sr)
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list})

    data=Cart.objects.filter(cart_user=request.user)
    coun=data.count();
    t=0
    for i in data:
        t=t+i.cart_product.price*i.count
    return render(request,"cart.html",{"Data":data,"Total":t,"length":coun})

@login_required(login_url='/register/')
def PastOrders(request):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
            paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list,"No":noData})

    data=Order.objects.filter(order_user=request.user)
    return render(request,"placedorder.html",{"Data":data})

@login_required(login_url='/register/')
def PastOrders2(request):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)
        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list,"No":noData})

    data=PreviousOrder.objects.filter(order_user=request.user)
    return render(request,"pastorders.html",{"Data":data})

def CartDelete(request,num):
    data=Cart.objects.get(cart_product__id=num)
    data.delete()
    return HttpResponseRedirect('/cart/')

def CartEdit1(request,num):
    data=Cart.objects.get(cart_product__id=num)
    data.count= int(data.count)-1
    if(int(data.count)==0):
        data.count=1
    data.save()
    return HttpResponseRedirect("/cart/")

def OrderPlaced(request,num):
    cat = Category.objects.all()
    siz = Size.objects.all()
    if (request.method == 'POST'):
        sr = request.POST.get('search')
        data = Product.objects.filter(Q(description__icontains=sr) | Q(name__icontains=sr) |
Q(pid__contains=sr),
                                      size__sname=sr)
        datacount = data.count()
        noData = ""
        if (datacount == 0):
            noData = "No Such product found"
            data = Product.objects.all()
        paginator = Paginator(data, 8)
        page = request.GET.get('page')
        try:
            product_list = paginator.page(page)
        except PageNotAnInteger:
            product_list = paginator.page(1)
        except EmptyPage:
            product_list = paginator.page(paginator.num_pages)

        return render(request, 'shop.html', {"Data": data, "Cat": cat, "Siz": siz, "posts":
product_list,"No":noData})

    data = Cart.objects.filter(cart_user=request.user)
    adata = Checkout.objects.filter(checkid=num)
    t=0
    for i in data:
        t = t + i.cart_product.price * i.count

    return render(request,"orderplace.html",{"Data":data,"Adata":adata,"total":t})
def homedelete(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/')
def deletecart(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/cart/')
def deleteabout(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/about/')
def deleteshop(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/shop/sample/')

def deletePreviousorders(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/Previousorders/')

def deletepastorder(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/pastorder/')
def deletelogout(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/logout/')
def deleteadminpage(request):
    data = Cart.objects.filter(cart_user=request.user)
    data.delete()
    return HttpResponseRedirect('/adminpage/')

def OrderPlaced2(request):
    data=PreviousOrder.objects.filter(order_user=request.user)
    return render(request,"orderplace.html",{"Data":data})

def CartEdit(request,num):
    data=Cart.objects.get(cart_product__id=num)
    data.count= int(data.count)+1
    data.save()
    return HttpResponseRedirect("/cart/")

@login_required(login_url='/register/')
def AddAddress(request):
    i=None
    x=""
    if (request.method == 'POST'):
        try:
            check = Checkout()
            check.checkid= request.user
            x=request.user
            check.chname = request.POST.get('name')
            check.checkout_user = request.user
            check.mobile = request.POST.get('mobile')
            check.email = request.POST.get('email')
            check.state = request.POST.get('state')
            check.city = request.POST.get('city')
            check.address = request.POST.get('address')
            check.pin = request.POST.get('pin')
            check.save()
            y = "/checkout/" + str(x) + "/"
            return HttpResponseRedirect(y)
        except:
            error(request, "Invalid Record")
    return render(request, "addaddress.html")
```

```
@login_required(login_url='/register/')
def AddAddress2(request):
    i=None
    nam=None
    x=""
    data = Checkout.objects.filter(checkout_user=request.user)
    for i in data:
        nam=i.checkid
    x=int(len(nam))
    if (request.method == 'POST'):
        try:
            check = Checkout()
            names=str(request.user)
            y = x-int(len(names))
            y=y+1
            subname=names[:y]
            check.checkid= names+str(subname)
            x= check.checkid
            check.chname = request.POST.get('name')
            check.checkout_user = request.user
            check.mobile = request.POST.get('mobile')
            check.email = request.POST.get('email')
            check.state = request.POST.get('state')
            check.city = request.POST.get('city')
            check.address = request.POST.get('address')
            check.pin = request.POST.get('pin')
            check.save()
            success(request, "Address is added")
            y="/checkout/"+x+"/"
            return HttpResponseRedirect(y)
        except:
            error(request, "Invalid Record")
    return render(request, "addaddress.html")
@login_required(login_url='/register/')
def SelectAddress(request):
    adata=Checkout.objects.filter(checkout_user=request.user)
    if(len(adata)==0):
        return HttpResponseRedirect('/addaddress/')
    return render(request,"selectaddress.html",{"Data":adata})
```

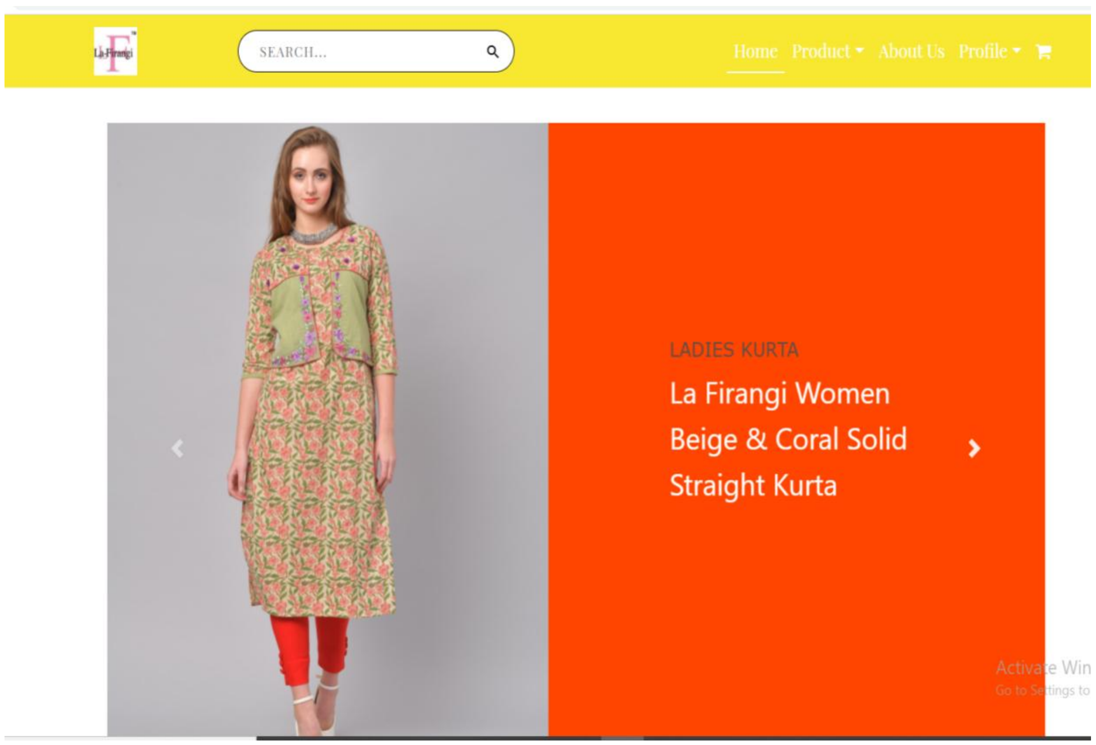
```
MERCHANT_KEY='sample'
@login_required(login_url='/register/')
def CheckoutForm(request,num):
    data = Cart.objects.filter(cart_user=request.user)
    temp=0
    t = 0
    orderr=0
    for i in data:
        t = t + i.cart_product.price * i.count

    adata=Checkout.objects.filter(checkid=num)
    orde=Order.objects.all()
    if(orde != None):
        for z in orde:
            orderr= int(z.ordernumber)
    O = Order()
    print(request.user)
    if(request.method=='POST'):
        try:
            choice = request.POST.get('choice')
            if(choice=='COD'):
                for i in data:
                    O = Order()
                    O.ordernumber = orderr+1
                    orderr=orderr+1
                    O.order_user = request.user
                    d = Product.objects.get(id=i.cart_product.id)
                    O.order_product = d
                    for x in adata:
                        O.order_address = x
                    O.count = i.count
                    O.save()
                x='/orderplaced/'+num+"/"
                return HttpResponseRedirect(x)
            # elif\
            #         (choice=='Paypal'):
            #     success(request, "pay with paypal")
            #     return HttpResponseRedirect('/payment/process/')
        except:
            error(request, "Invalid Record")

    return render(request,"checkout.html",{"Total":t,"Data":adata})

def About(request):
    return render(request,'about.html')

def OrderAdmin(request):
    count=len(PreviousOrder.objects.all())
    data=Order.objects.all()
    return render(request,'orderadmin.html',{"Data":data})

def CancelOrderAdmin(request):
    data=CancelOrder.objects.all()
    return render(request,'canceladmin.html',{"Data":data})

def DispatchedOrder(request,num):

    data = Order.objects.get(ordernumber=num)
    try:
        p = PreviousOrder()
        p.ordernumber = data.ordernumber
        p.order_user = data.order_user
        p.order_product = data.order_product
        p.count=data.count
        p.order_address = data.order_address
        p.save()
        try:
            dispatch_email(request,data)
        except:
            error(request, "mail not send")
        data.delete()
    except:
        error(request, "Invalid Record")
    data=Order.objects.all()
    return render(request,'orderadmin.html',{"Data":data})

def Cancelorder(request,num):

    data = Order.objects.get(ordernumber=num)
    try:
        p = CancelOrder()
        p.ordernumber = data.ordernumber
        p.order_user = data.order_user
        p.order_product = data.order_product
        p.count=data.count
        p.order_address = data.order_address
        p.save()
        try:
            Cancel_email(request,data)
        except:
            error(request,"Mail not send")
        data.delete()
    except:
        error(request, "Invalid Record")
    data=Order.objects.filter(order_user=request.user)
    return render(request,'placedorder.html',{"Data":data})

def Returnorder(request,num):

    data = PreviousOrder.objects.get(ordernumber=num)
    try:
        p = ReturnOrder()
        p.ordernumber = data.ordernumber
        p.order_user = data.order_user
        p.order_product = data.order_product
        p.count=data.count
        p.order_address = data.order_address
        p.save()
        try:
            Return_email(request, data)
        except:
            error(request,"Mail not send")

    except:
        error(request, "Invalid Record")
    data=PreviousOrder.objects.filter(order_user=request.user)
    return render(request,'pastorders.html',{"Data":data})

def Return_email(request,data):
    subject = 'Return Request'
    message = 'Dear '+data.order_address.chname+",\n        Your Request of Product
Return is registered "
    email_from = settings.EMAIL_HOST_USER
    recipient_list = [data.order_address.email,]
    send_mail( subject, message, email_from, recipient_list )

def ReturnOrderAdmin(request):
    data=ReturnOrder.objects.all()
    return render(request,'returnorder.html',{"Data":data})

def CancelReturn(request,num):
    data=PreviousOrder.objects.get(ordernumber=num)
    try:
        subject = 'Return Request'
        message = 'Dear ' + data.order_address.chname + ",\n         Your Request of
Product Return is Decline "
        email_from = settings.EMAIL_HOST_USER
        recipient_list = [data.order_address.email, ]
        send_mail(subject, message, email_from, recipient_list)
    except:
        error(request, "Mail not send")
    adata=ReturnOrder.objects.get(ordernumber=num)
    try:
        adata.delete()
    except:
        error(request, "Invalid Record")
    adata = ReturnOrder.objects.all()
    return render(request, 'returnorder.html', {"Data": adata})

def ConfirmReturn(request,num):
    data=PreviousOrder.objects.get(ordernumber=num)
    try:
        subject = 'Return Request'
        message = 'Dear ' + data.order_address.chname + ",\n         Your Request of
Product Return is confirmed and your order will be picked soon by our executive
from your delivered address "
        email_from = settings.EMAIL_HOST_USER
        recipient_list = [data.order_address.email, ]
        send_mail(subject, message, email_from, recipient_list)
    except:
        error(request,"Mail not send")
    adata=ReturnOrder.objects.get(ordernumber=num)
    try:
        adata.delete()
        data.delete()
    except:
        error(request, "Invalid Record")
    adata = ReturnOrder.objects.all()
    return render(request, 'returnorder.html', {"Data": adata})
```

Models.py

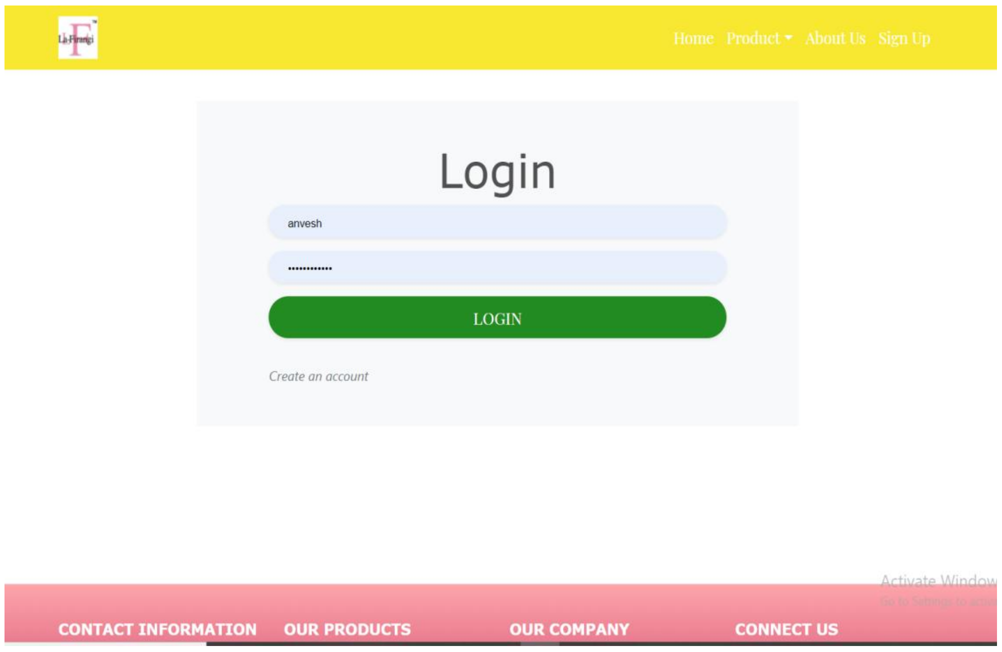
```
from django.db import models
from django.contrib.auth.forms import User
# Create your models here.

class Category(models.Model):
    cid = models.AutoField
    cname = models.CharField(max_length=30)
    def __str__(self):
        return self.cname

class Size(models.Model):
    sid = models.AutoField
    sname = models.CharField(max_length=5)
    def __str__(self):
        return self.sname

class Product(models.Model):
    id = models.CharField(max_length=30,primary_key=True)
    pid = models.CharField(max_length=30)
    brand = models.CharField(max_length=30,default=None)
    cat = models.ForeignKey(Category,on_delete="CASCADE",default=None)
    name = models.CharField(max_length=100)
    description = models.TextField()
    basicPrice = models.IntegerField()
    discount = models.IntegerField()
    price = models.IntegerField()
    size = models.ForeignKey(Size,on_delete="CASCADE",default=None)

    img1 = models.ImageField(upload_to='images')
    img2 = models.ImageField(upload_to='images',default=None)
    date = models.DateTimeField(auto_now_add=True)
    update = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name

class Cart(models.Model):
    cartid=models.AutoField
    cart_user=models.ForeignKey(User,on_delete="CASCADE",default=None)
    cart_product=models.ForeignKey(Product,on_delete="CASCADE",default=None)
    count=models.IntegerField(default=1)
    total=models.IntegerField()
    date=models.DateTimeField(auto_now_add=True)
    update=models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.cart_user

class Checkout(models.Model):
    checkid=models.CharField(max_length=30,primary_key=True,default=None)
    checkout_user = models.ForeignKey(User, on_delete="CASCADE", default=None)
    chname=models.CharField(max_length=30)
    mobile=models.IntegerField()
    email=models.EmailField(max_length=50)
    state=models.CharField(max_length=30)
    city=models.CharField(max_length=30)
    address=models.CharField(max_length=50)
    pin=models.CharField(max_length=10)

    def __str__(self):
        return self.chname

class Order(models.Model):
    orderid =  models.AutoField
    ordernumber = models.IntegerField()
    order_user=models.ForeignKey(User,on_delete="CASCADE",default=None)
    order_product=models.ForeignKey(Product,on_delete="CASCADE",default=None)
    count=models.IntegerField(default=1)
    order_address=models.ForeignKey(Checkout,on_delete="CASCADE",default=None)
    def __str__(self):
        return self.order_user

class PreviousOrder(models.Model):
    orderid =  models.AutoField
    ordernumber = models.IntegerField()
    order_user=models.ForeignKey(User,on_delete="CASCADE",default=None)
    order_product=models.ForeignKey(Product,on_delete="CASCADE",default=None)
    count=models.IntegerField(default=1)
    order_address=models.ForeignKey(Checkout,on_delete="CASCADE",default=None)
    def __str__(self):
        return self.order_user

class CancelOrder(models.Model):
    orderid =  models.AutoField
    ordernumber = models.IntegerField()
    order_user=models.ForeignKey(User,on_delete="CASCADE",default=None)
    order_product=models.ForeignKey(Product,on_delete="CASCADE",default=None)
    count=models.IntegerField(default=1)
    order_address=models.ForeignKey(Checkout,on_delete="CASCADE",default=None)

    def __str__(self):
        return self.order_user

class ReturnOrder(models.Model):
    orderid =  models.AutoField
    ordernumber = models.IntegerField()
    order_user=models.ForeignKey(User,on_delete="CASCADE",default=None)
    order_product=models.ForeignKey(Product,on_delete="CASCADE",default=None)
    count=models.IntegerField(default=1)
    order_address=models.ForeignKey(Checkout,on_delete="CASCADE",default=None)

    def __str__(self):
        return self.order_user
```

La-firangi Website Snapshot

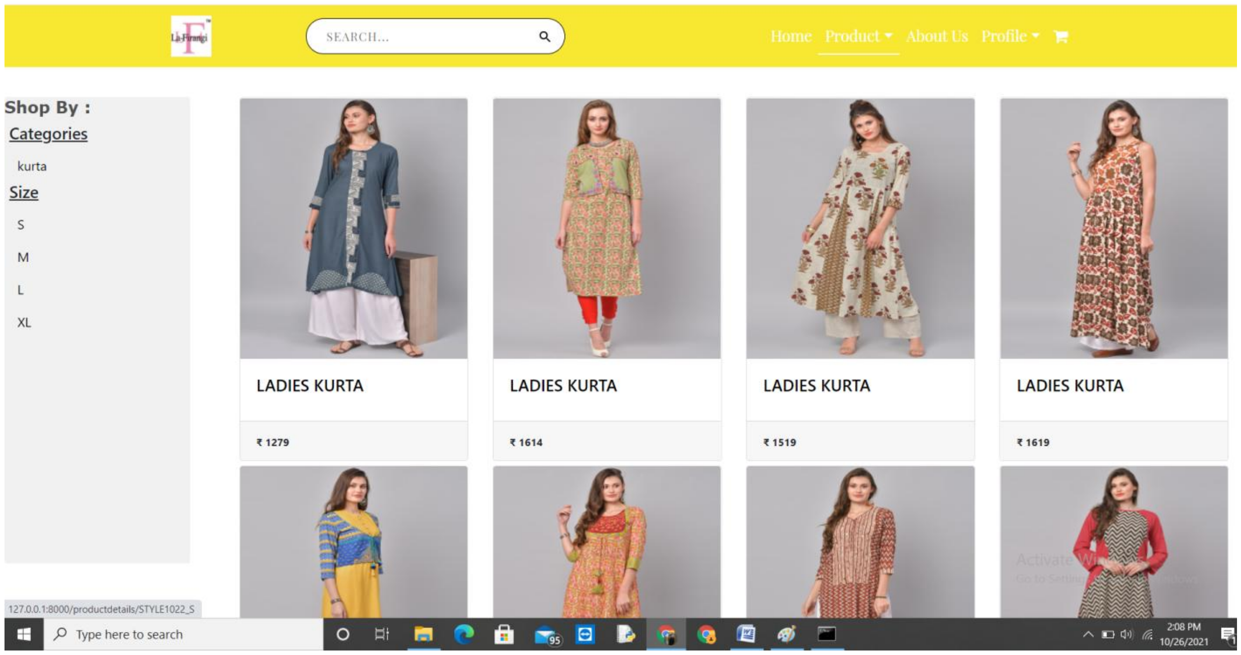
Home Page



Login Page



Product Page



Shop By :

Categories

kurta

Size

S

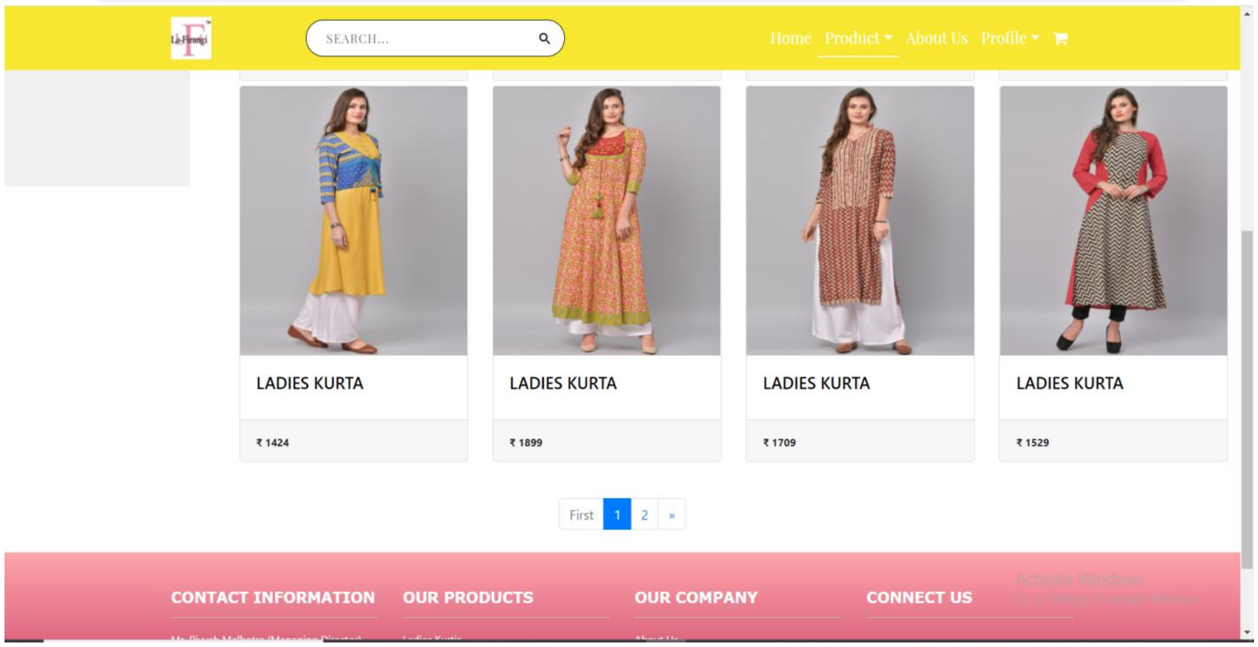
M

L

XL

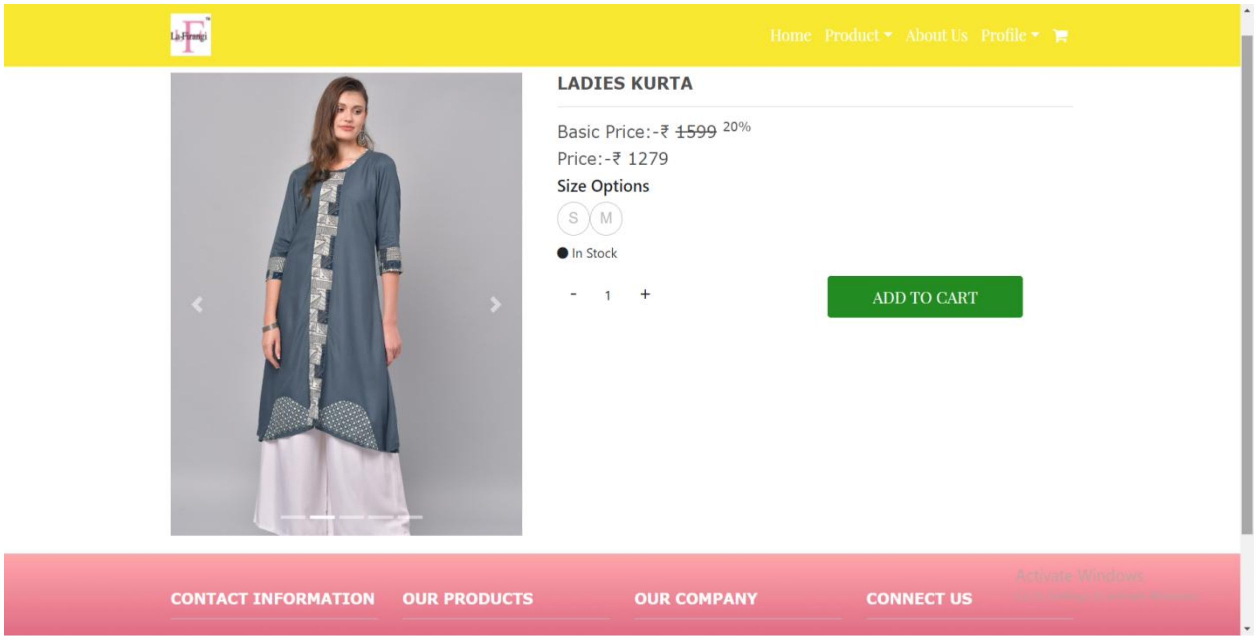
LADIES KURTA	LADIES KURTA	LADIES KURTA	LADIES KURTA
₹ 1279	₹ 1614	₹ 1519	₹ 1619

127.0.0.1:8000/productdetails/STYLE1022_S



LADIES KURTA	LADIES KURTA	LADIES KURTA	LADIES KURTA
₹ 1424	₹ 1899	₹ 1709	₹ 1529

CONTACT INFORMATION OUR PRODUCTS OUR COMPANY CONNECT US



LADIES KURTA

Basic Price:-₹ ~~1599~~ 20%

Price:-₹ 1279

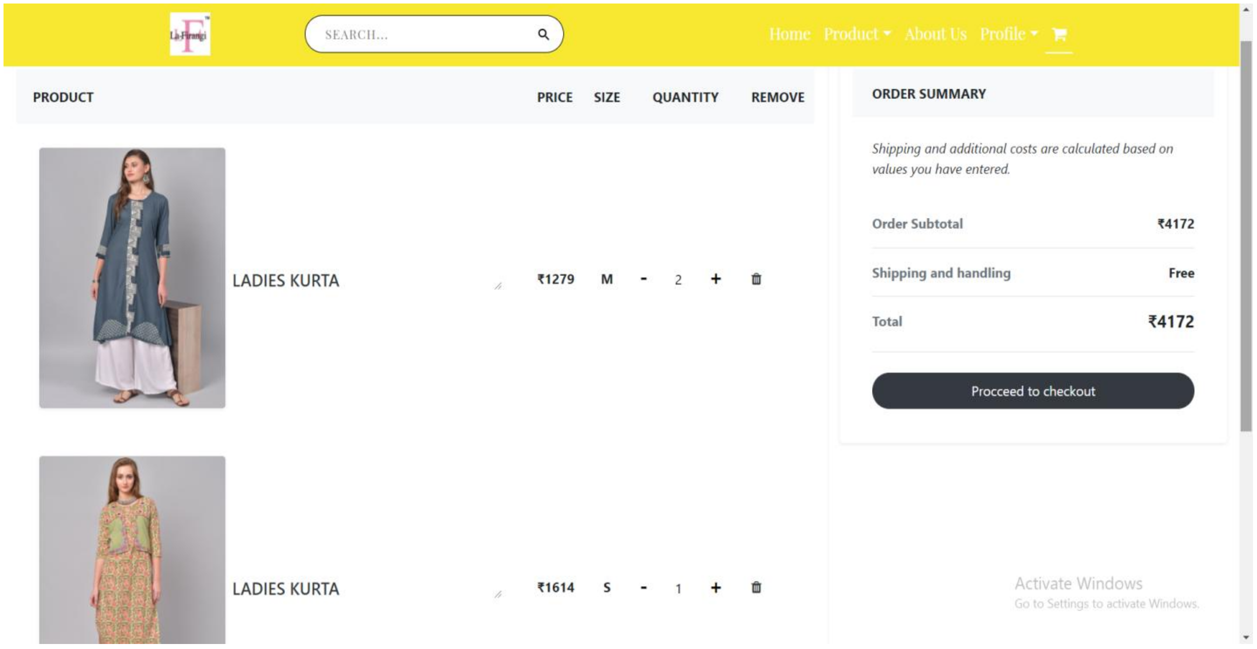
Size Options

S M

● In Stock

- 1 +

Cart



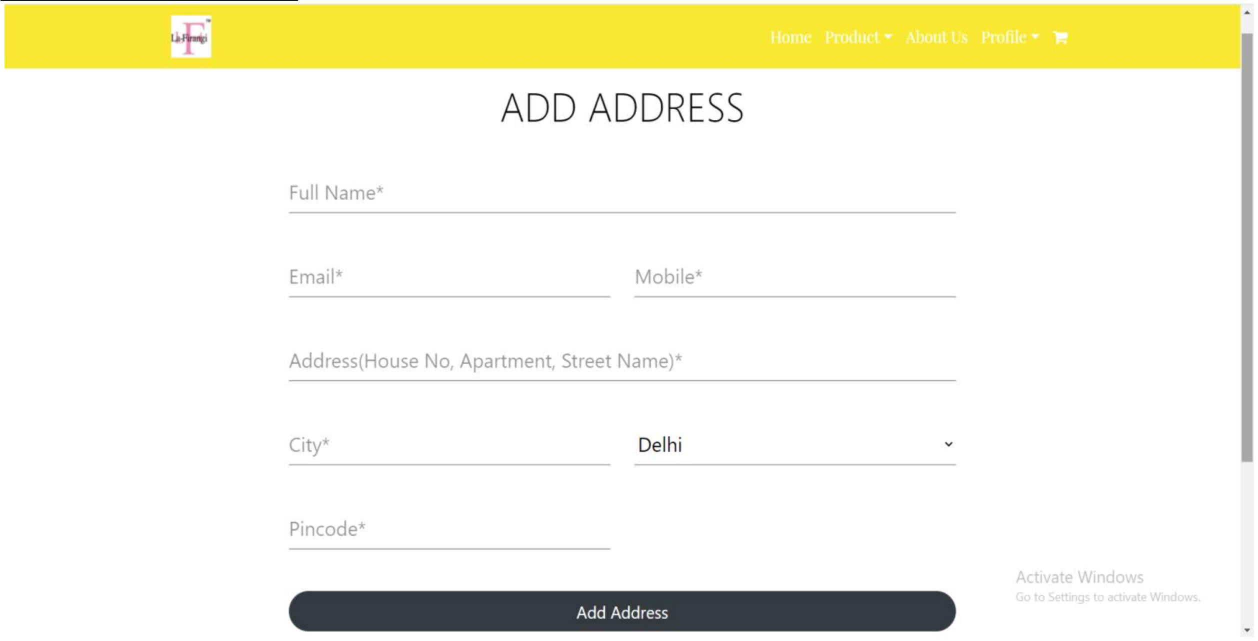
PRODUCT	PRICE	SIZE	QUANTITY	REMOVE
LADIES KURTA	₹1279	M	- 2 +	
LADIES KURTA	₹1614	S	- 1 +	

ORDER SUMMARY

Shipping and additional costs are calculated based on values you have entered.

Order Subtotal	₹4172
Shipping and handling	Free
Total	₹4172

Add Address



ADD ADDRESS

Full Name*

Email*

Mobile*

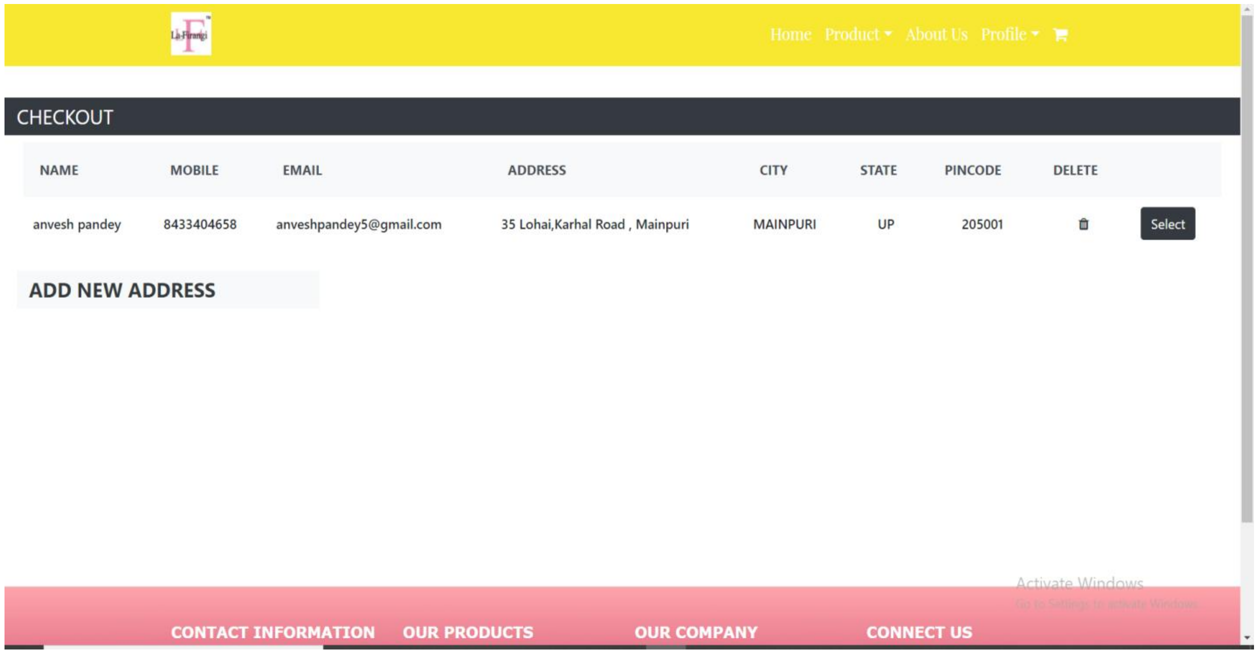
Address(House No, Apartment, Street Name)*

City*

Delhi

Pincode*

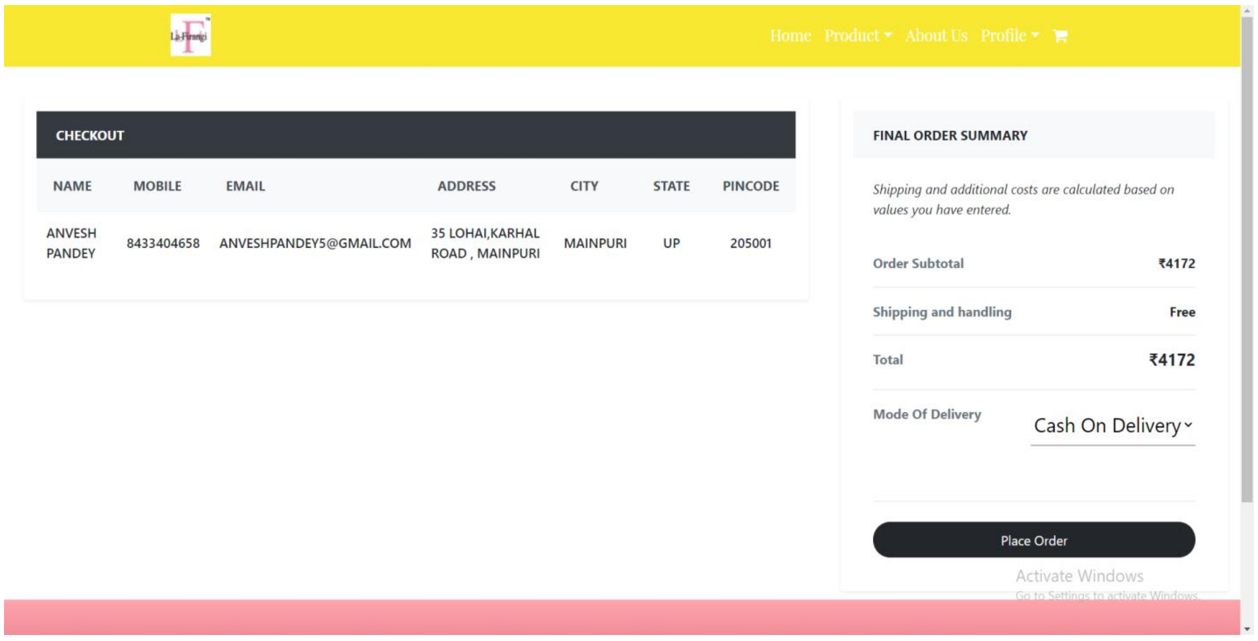
Add Address



CHECKOUT

NAME	MOBILE	EMAIL	ADDRESS	CITY	STATE	PINCODE	DELETE	
anvesh pandey	8433404658	anveshpandey5@gmail.com	35 Lohai,Karhal Road , Mainpuri	MAINPURI	UP	205001	🗑	Select

ADD NEW ADDRESS


La-Firangi
Home Product About Us Profile
Place Order
Activate Windows
Go to Settings to activate Windows.

CHECKOUT

NAME	MOBILE	EMAIL	ADDRESS	CITY	STATE	PINCODE
ANVESH PANDEY	8433404658	ANVESHPANDEY5@GMAIL.COM	35 LOHAI,KARHAL ROAD , MAINPURI	MAINPURI	UP	205001

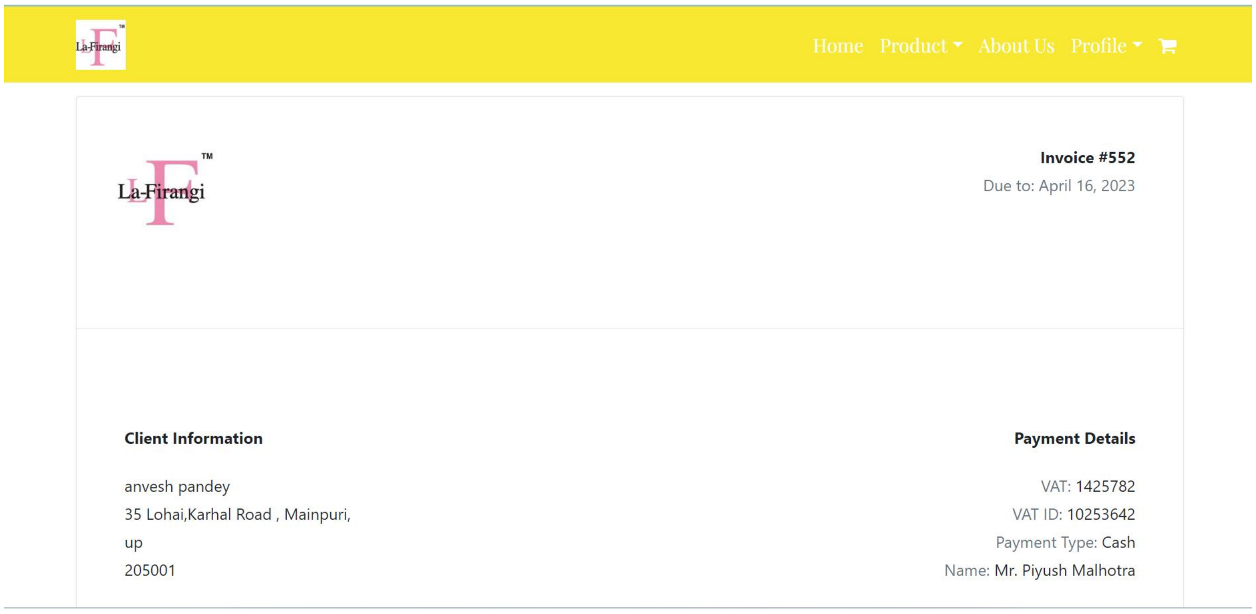
FINAL ORDER SUMMARY

Shipping and additional costs are calculated based on values you have entered.

Order Subtotal	₹4172
Shipping and handling	Free
Total	₹4172
Mode Of Delivery	Cash On Delivery

Order placed

Invoice



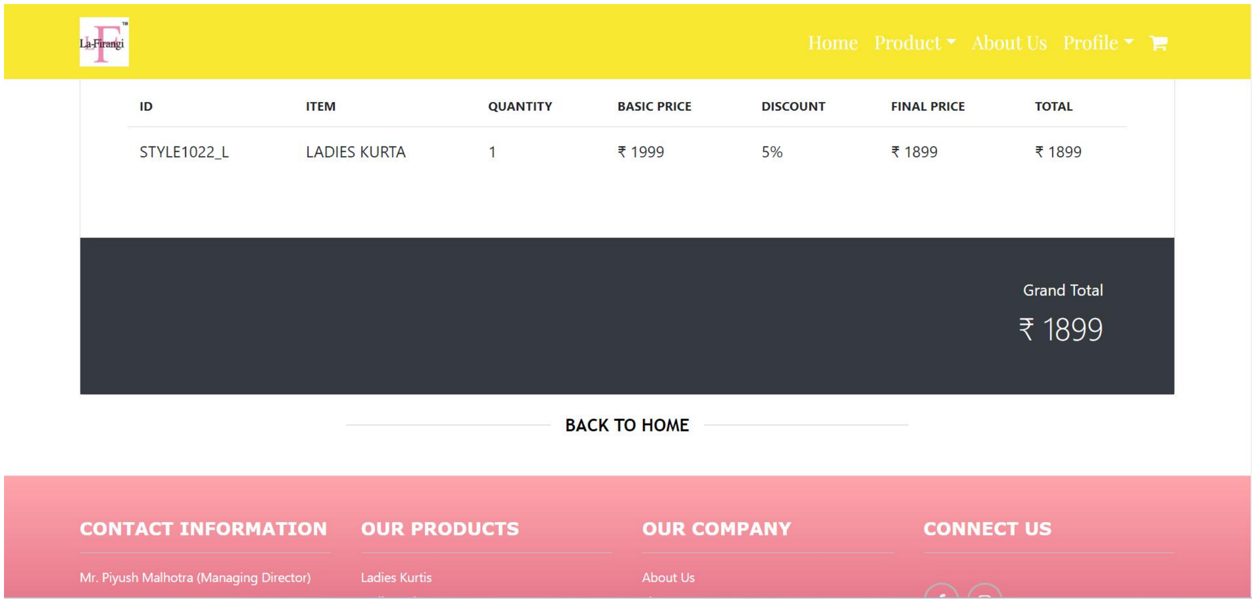
Invoice #552
Due to: April 16, 2023

Client Information

anvesh pandey
35 Lohai,Karhal Road , Mainpuri,
up
205001

Payment Details

VAT: 1425782
VAT ID: 10253642
Payment Type: Cash
Name: Mr. Piyush Malhotra



ID	ITEM	QUANTITY	BASIC PRICE	DISCOUNT	FINAL PRICE	TOTAL
STYLE1022_L	LADIES KURTA	1	₹ 1999	5%	₹ 1899	₹ 1899

Grand Total
₹ 1899

BACK TO HOME

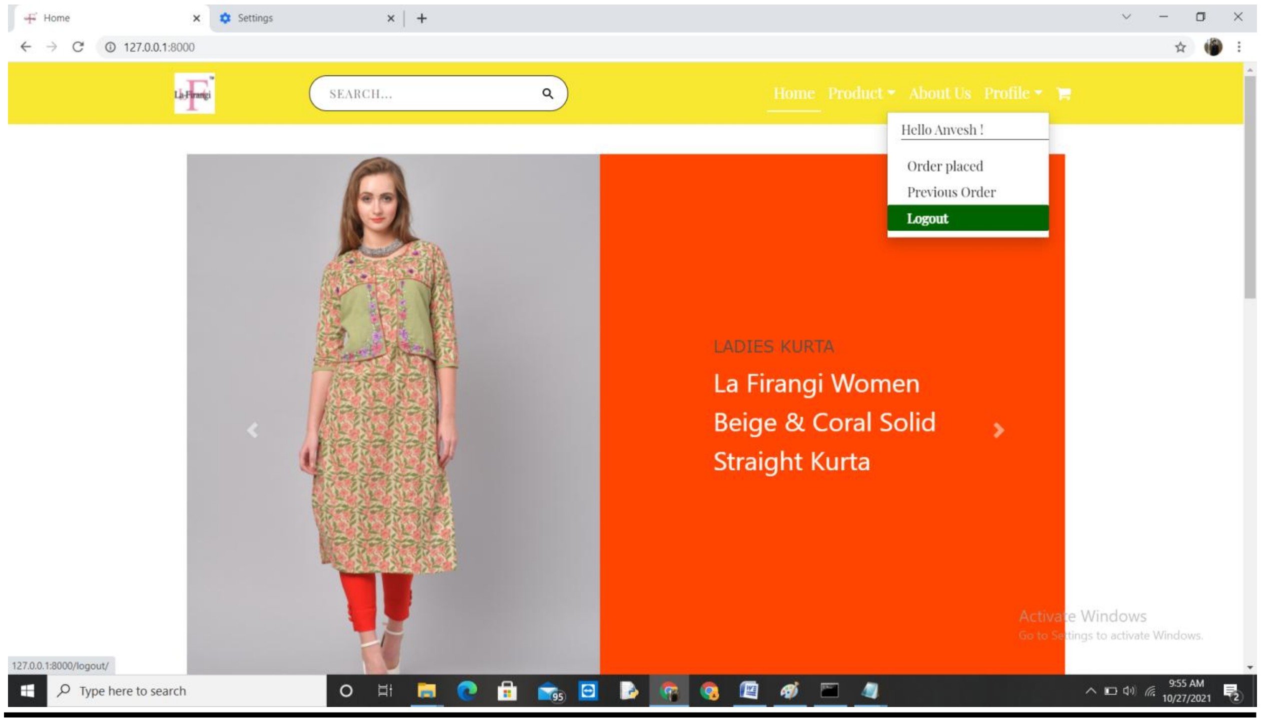
CONTACT INFORMATION
Mr. Piyush Malhotra (Managing Director)

OUR PRODUCTS
Ladies Kurtis

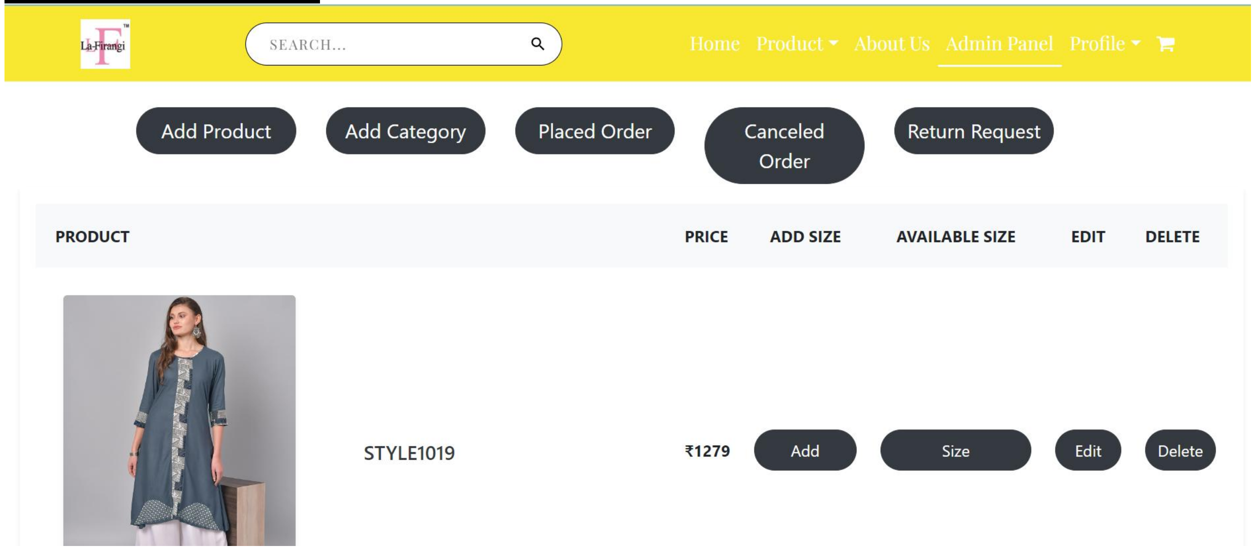
OUR COMPANY
About Us

CONNECT US

Logout

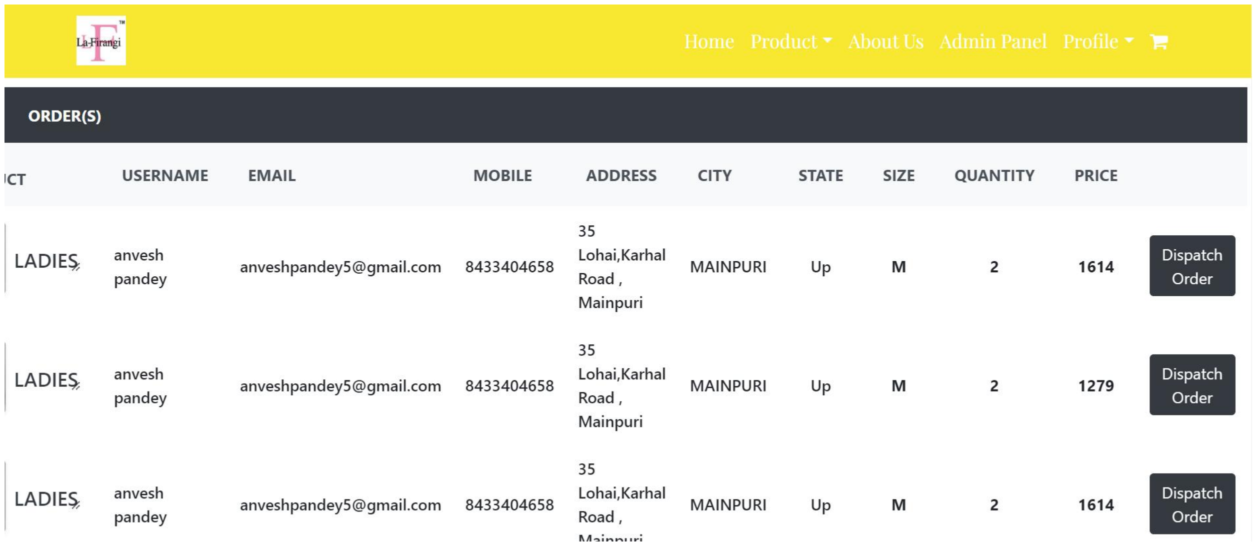


Admin Panel:



PRODUCT		PRICE	ADD SIZE	AVAILABLE SIZE	EDIT	DELETE
	STYLE1019	₹1279	Add	Size	Edit	Delete

Admin Confirm Order:



ORDER(S)

CT	USERNAME	EMAIL	MOBILE	ADDRESS	CITY	STATE	SIZE	QUANTITY	PRICE	
LADIES	anvesh pandey	anveshpandey5@gmail.com	8433404658	35 Lohai,Karhal Road , Mainpuri	MAINPURI	Up	M	2	1614	Dispatch Order
LADIES	anvesh pandey	anveshpandey5@gmail.com	8433404658	35 Lohai,Karhal Road , Mainpuri	MAINPURI	Up	M	2	1279	Dispatch Order
LADIES	anvesh pandey	anveshpandey5@gmail.com	8433404658	35 Lohai,Karhal Road ,	MAINPURI	Up	M	2	1614	Dispatch Order

10.Conclusion

The website developed is working satisfactorily under the due constraints. This application will be of great help for the organization, who now feels more confident as now there is not any paper work left. The application is been implemented in almost every organization in the country or states and is working successfully.

"La-firangi" will facilitate the clients with the following features:

- Admin can easily add items or categories for the expansion of business
- Automized recommendation of product.
- Data Security.
- Authenticity of Users.
- Search and delete items.
- Easy management of data.
- User defined data entry.
- User friendly environment.
- Easy to operate.

11.Scope of Project

In the future, I would like to add some extra features in this project, which are as follows:

- The project can be made more user friendly by adding some more GUI features.

- A module for printing various reports generated can also included in this project.

- To secure the data from any malicious misuse, more sophisticated methodologies like digitized signature for data accessing can be employed to restrict access and minimize the threat to the data.

- Some provision for suggestion and complaints can be made.

- More filtered and detailed enquiry can be given to the user.

12. References

- **Mastering Django: core**
- **The definite guide to Django: Web Development Done Right**
- **www.testingexcellence.com**
- **www.freetutes.com**