A

Minor Project Report

ON

E-COMMERCE SITE

for the Partial fulfilment of the award of

Degree of

Master of Computer Application (MCA

Under the Guidance of:
Miss. Vartika Shah

Submitted by:
Kana Ram Yadav
Shikha Gupta
Vaibhav Rathor

Department of Information Technology & Computer Application

JECRC UNIVERSITY, JAIPUR

DECEMBER 2020

Candidate’s Declaration

I hereby declare that the project work, which is being presented in the Project Report, entitled **“E – COMMERCE SITE”** in partial fulfillment for the award of Degree of “**Master of Computer Application**” in Dep’t. of Information Technology, **JECRC University** is a record of my own investigations carried under the Guidance of **Miss. Vartika Shah.**

I have not submitted the matter presented in this Project Report anywhere for the award of any other Degree.

Name of Candidate: Kana Ram Yadav
Shikha Gupta
Vaibhav Rathore

Enrolment No: 19MCAL079
19MCAL071
19MCAL087

Name(s) of Supervisor(s)/Guide	**Signature:**
Miss. Vartika Shah	Kana Ram Yadav
	Shikha Gupta
	Vaibhav Rathor

Countersigned By: HoD-IT & CA

ACKNOWLEDGEMENT

We had a great experience working on this project and we got to learn a plethora of new skills through this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them. We are highly indebted to the teachers and especially **Miss. Vartika Shah** for their guidance and constant supervision as well as providing necessary information regarding the project and also for their support in completing the project. We would like to express our gratitude towards our parents and friends for their kind cooperation and encouragement which help us in the completion of the project.

Name: Kana Ram Yadav
Shikha Gupta
Vaibhav Rathor
Place: Jaipur

ABSTRACT

E-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.   E-Commerce which was started in early 1990's has taken a great leap in the world of computers, but the fact that has hindered the growth of e-commerce is security. Security is the challenge facing e-commerce today & there is still a lot of advancement made in the field of security.

The main advantage of e-commerce over traditional commerce is the user can browse online shops, compare prices and order merchandise sitting at home on their PC.

For increasing the use of e-commerce in developing countries the B2B e-commerce is implemented for improving access to global markets for firms in developing countries. For a developing country advancement in the field of e-commerce is essential. The research strategy shows the importance of the e-commerce in developing countries for business applications.

INDEX PAGE

Introduction to Project

“E-Commerce site” is a Python program. The E-Commerce site widely used for online shopping.

1. **There are Use Payment API in this application:**
   Paytm API

2. **Language to be used:**

   Front end

   HTML, CSS, JavaScript

   Back end

   Python, Django

   DataBase

   Sqlite3

SYSTEM REQUIREMENT ANALYSIS (SRS DIAGRAMS)

Hardware Requirements

- ➢ RAM: 4 GB
- ➢ HDD: 10 GB or more (Free space excluding data size)
- ➢ Processor: i3 or onwards

Software Requirements

- ➢ PyCharm
- ➢ Python Shell

There are used many Python Libraries in this application

1. argon2-cffi==20.1.02.
2. asgiref==3.2.10
3. bcrypt==3.1.7
4. certifi==2020.6.20
5. cffi==1.14.0
6. chardet==3.0.4
7. cryptography==3.0
8. defusedxml==0.7.0rc1
9. Django==3.0.8
10. django-crispy-forms==1.9.2
11. djangorestframework==3.11.1
12. Faker==4.1.1
13. idna==2.10
14. importlib-metadata==1.7.0
15. Markdown==3.2.2
16. oauthlib==3.1.0
17. Pillow==7.2.0
18. pycparser==2.20
19. PyJWT==1.7.1
20. python-dateutil==2.8.1
21. python-social-auth
22. python3-openid==3.2.0
23. pytz==2020.1
24. requests==2.24.0
25. requests-oauthlib==1.3.0
26. six==1.15.0
27. social-auth-app-django==4.0.0
28. social-auth-core==3.3.3

29. sqlparse==0.3.1
30. text-unidecode==1.3
31. urllib3==1.25.10
32. wincertstore==0.2
33. zipp==3.1.0

Feasibility Study of the Project

Whatever we think need not be feasible. It is wise to think about the feasibility of any problem we undertake. Feasibility is the study of impact, which happens in the organization by the development of a system. The impact can be either positive or negative. When the Positives nominate the negatives, then the system is considered feasible. Here the feasibility study can be performed in two ways such as technical feasibility and Economical Feasibility.

- **Technical Feasibility:**

We can strongly say that it is technically feasible, since there will not be much difficulty in getting required resources for the development and maintaining the system as well. All the resources needed for the development of the software as well as the maintenance of the same is available in the organization here we are utilizing the resources which are already available.

- **Economic Feasibility**:

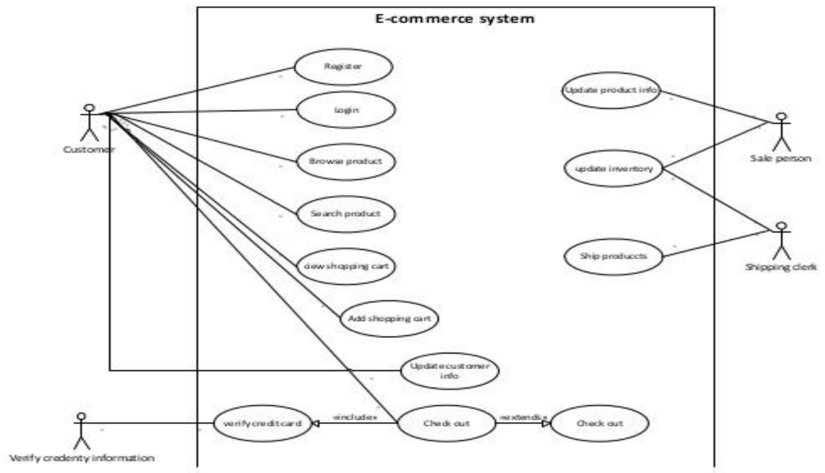
Development of this application is highly economically feasible. The organization needed not spend much money for the development of the system already available. The only thing is to be done is making an environment for the development with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources. Even after the development, the organization will not be in condition to invest more in the organization. Therefore, the system is economically feasible.

SYSTEM DESIGN

4.1 - UML Diagrams: -

Use Case Diagram - A use case is a set of scenarios that describe an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

Use case Diagram of e-Commerce system



Use case description:

Register:

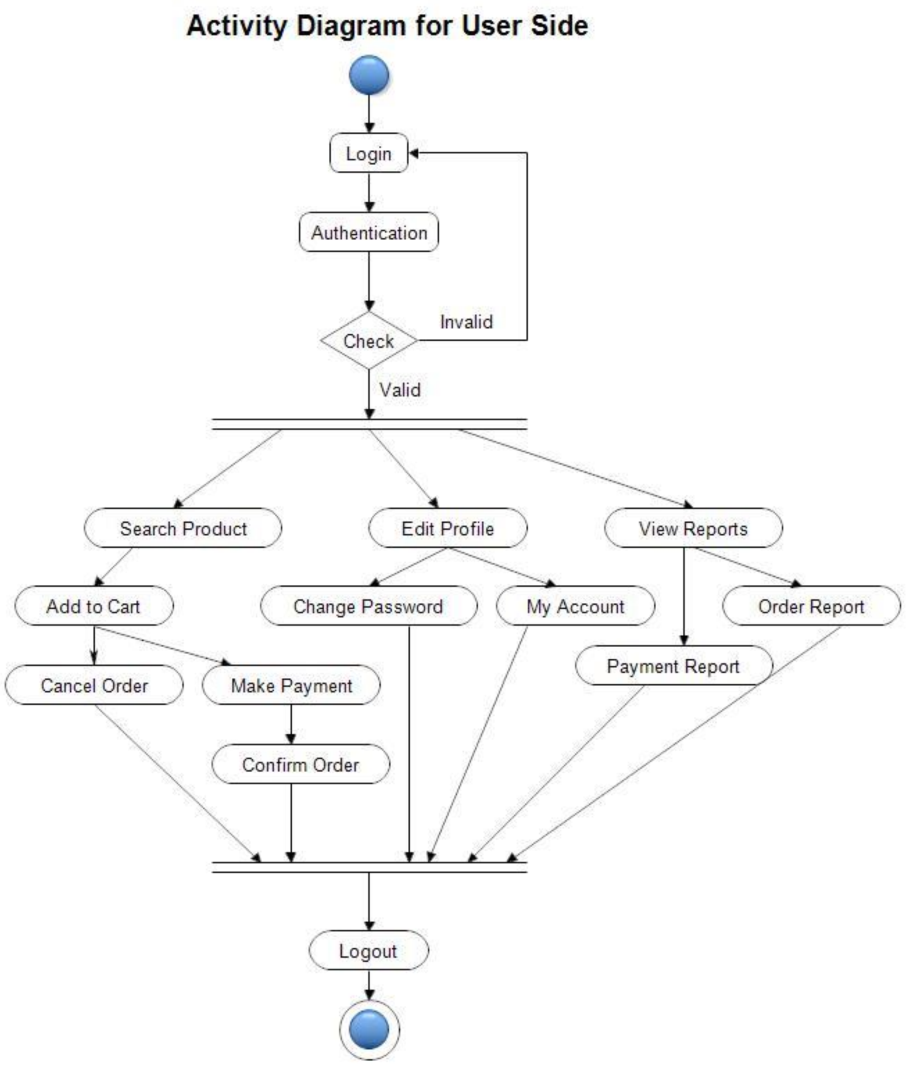
If a customer is new user, he can request to register page. A register page open and asks total information about customer and also asks to customer to choose login (email address) and password.

Login:

The customer can login by enter name and password. The system verifies the name and password matches. If not matches, error messages shows to the customer.

ActivityDiagram:

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc



TECHNOLOGY USED/PLATFORM

4.2.1 - Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

4.2.2 - API Uses:

1. Paytm Developer API

Paytm Payment Gateway provides a secure, PCI-compliant way to accept Debit/Credit card, Net-Banking, UPI and Paytm wallet payments from your customers.

Understanding account credentials

Account credentials are available in your dashboard for both staging and production environment. These credentials consist of the following:

- **MID (Merchant ID)** - This is a unique identifier provided to every merchant by Paytm. MID is part of your account credentials and is different on staging and production environment. Your staging MID is available here and production MID will be available once your activation is complete.
- **Merchant Key** - This is a unique secret key used for secure encryption of every request. This needs to be kept on server-side and should not be shared with anyone.

Detailed Payment Flow Explained

Transaction Creation - When a transaction request is received at Paytm's server, there are multiple validations carried out like valid source of the request, the structure of request, uniqueness of request etc. Once these validations are passed, a transaction is created.

Successful Transaction - Customer fills basic payment details to authorize the payment. Once the authorization is successful, money is debited from the customer’s account. This transaction is a successful transaction.

Failed Transaction - If the customer drops out from the payment process, or in the event of payment authorization failure, money is not deducted from the customer's account. This is marked as a failed transaction.

Pending Transaction - Sometimes Paytm doesn't receive real-time transaction status from the bank. This can be due to many reasons such as network issues, technical errors at customer's/bank's end etc. This is marked as a pending transaction. Refer this for detailed handling of the pending transaction.

Settled Transaction - Payments received against successful transactions are credited into your bank account on T+1, where T is the date of a successful transaction. Once the payment is credited, the corresponding transaction is marked as Settled.

Refund Transaction - Sometimes there are use cases where you need to reverse payments for successful or settled transactions. The reversal transaction of a successful or settled payment is called a refund transaction. Refer Refunds for more details.

4.2.4 - Python Libraries

1. **Django:-**

   Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

2. **Oauthlib:-**

   OAuthLib is a framework which implements the logic of OAuth1 or OAuth2 without assuming a specific HTTP request object or web framework.

3. **Pillow:-**

   Pillow is the “friendly” PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors

4. **python-social-auth:-**

   Python Social Auth aims to be an easy-to-setup social authentication and authorization mechanism for Python projects supporting protocols like OAuth (1 and 2), OpenID and others.

5. **requests:-**

   Requests is a Python HTTP library, released under the Apache License 2.0. The goal of the project is to make HTTP requests simpler and more human-friendly.

6. **requests-oauthlib:-**

   OAuth 1 can seem overly complicated and it sure has its quirks. Luckily, requests_oauthlib hides most of these and let you focus at the task at hand.

7. **social-auth-core:-**

   Python Social Auth is an easy to setup social authentication/registration mechanism with support for several frameworks and auth providers.

8. **urllib3:-**

   urllib3 is a powerful, user-friendly HTTP client for Python. Much of the Python ecosystem already uses urllib3 and you should too. urllib3 brings many critical features that are missing from the Python standard libraries:

   - Thread safety.
   - Connection pooling.
   - Client-side SSL/TLS verification.
   - File uploads with multipart encoding.
   - Helpers for retrying requests and dealing with HTTP redirects.
   - Support for gzip, deflate, and brotli encoding.
   - Proxy support for HTTP and SOCKS.
   - 100% test coverage.

CODING

1. **Login.html**

```
•  {% extends 'store/main.html' %}
•  {% load crispy_forms_tags %}
•  {% block content %}
•      <h1 class="display-5">Log In!</h1>
•      <hr class="my-4">
•      <form method="POST">
•          {% csrf_token %}
•          {{ form|crispy }}
•          <br>
•          <input type="submit" class="btn btn-primary mb-
   2" name="submit" , value='Login'>
•      </form>
•      <br>
•      <a class="align-
   middle" href="{% url 'password_reset' %}">Forgotten Your password?</a><br>
•      <h6 class="align-middle">OR</h6>
•      <a class="align-
   middle" href="{% url 'register' %}">Create a new account!</a>
•      <br><br>
•      <div>
•          <h3>Login with :</h3>
•          <a href="{% url 'social:begin' 'google-oauth2' %}">
            <button type="button" name="button" class="btn btn-
danger">Google</button>
•          </a>
•      </div>
•
•  {% endblock %}
```

2. **Registration.html**

```
{% extends 'store/main.html' %}
{% load crispy_forms_tags %}
{% block content %}
<h1 class="display-5">Create a new account!</h1>
<br>
<form method="POST">
    {% csrf_token %}
    {{ form|crispy }}
    <br>
    <input type="submit" name="submit" class="btn btn-primary mb-
2" value="Register">
    <input type="reset" name="reset" class="btn btn-secondary mb-
2" value="Reset">
    <br>
    <a href="{% url 'login' %}">Or Log into an existing account!</a>
</form>

{% endblock %}
```

3. Account/models.py

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

class Profile(models.Model):

    user = models.OneToOneField(User,on_delete=models.CASCADE)
    dob = models.DateField(null=True,blank=True)
    photo = models.ImageField(null=True,blank=True)

    def __str__(self):
        return "profile {}".format(self.user.username)

```

4. Account/urls.py

```
from django.urls import path
from . import views




urlpatterns = [
    path('login/',views.user_login,name = 'user_login'),
    path('logout/',views.user_logout ,name = 'user_logout'),
    path('register/',views.register,name = 'register'),
    path('edit_profile/',views.edit_profile , name = 'edit_profile'),
    path('profilepage/<str:username>/',views.profilepage,name='profilepage'),
]
```

4. **Ecommerce.urls.py**

```python
"""ecommerce URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path , include
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('store.urls'),name='app_store'),
    path('',include('accounts.urls'),name='auth'),
    path('',include('store.api.urls'),name='api'),
    path('', include('django.contrib.auth.urls')),
    path('', include('django.contrib.auth.urls')),
    path(r'^oauth/', include('social_django.urls', namespace='social')),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

1. **Templaate/store.Main.html**

```html
<!DOCTYPE html>
{% load static %}
```

```html
<html>
<head>
    <title>Ecommerce</title>

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1" />

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
    <script src="https://maxscdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.min.css" integrity="sha512-1PKOgIY59xJ8Co8+NE6FZ+LOAZKjy+KY8iq0G4B3CyeY6wYHN3yt9PW0XpSriVlkMXe40PTKnXrLnZ9+fkDaog==" crossorigin="anonymous" />

    <link rel="stylesheet" type="text/css" href="{% static 'store/css/main.css' %}">
    <script type="text/javascript">
        var user = '{{request.user}}'
    </script>
    <style>

        .dropdown {
          position: relative;
          display: inline-block;
        }

        .dropdown-content {
          display: none;
          position: absolute;
          background-color: #343A40;
          min-width: 180px;
          box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
          z-index: 1;
        }

        .dropdown-content a {
          color: white;
          padding: 8px 16px;
          text-decoration: none;
          display: block;
        }

        .dropdown-content a:hover {background-color: rgb(138, 136, 136);}

        .dropdown:hover .dropdown-content {display: block;}

</style>

</head>
<body>

    <nav class="navbar navbar-expand-lg navbar-dark badge-dark">
      <h5><strong><a class="navbar-brand" href="{% url 'store' %}" style="color: white">Ecommerce</a></strong></h5>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>

      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
          <li class="nav-item active">
      <h5><strong><a class="nav-link" href="{% url 'store' %}" style="color: white">Store <span class="sr-only">(current)</span></a></strong></h5>
     </li>

    </ul>
    <div class="form-inline my-2 my-lg-0">

      <form method="get" action="{% url 'search' %}" class="form-inline my-2 my-lg-0">
          <input type="search" name="search" aria-label="Search" placeholder="Search" class="form-control mr-sm-2">
          <button class="btn btn-success my-2 mr-sm-3" type="submit">Search</button>
      </form>
      <h5 class="mr-sm-3"><strong><a href="#" style="color: white">Tracker</a></strong></h5>

      <div class="dropdown mr-sm-3">
           <h5><strong><a style="color: white">Categories</a></strong></h5>
           <div class="dropdown-content">
            {% for product in product_categories %}
              <a href="{% url 'show_items' product.id %}">{{product.name}}</a>
            {% endfor%}
           </div>
      </div>
      {% if request.user.is_authenticated %}
        <h5 class="mr-sm-3"><strong><a href="{% url 'order_details' %}" style="color: white">Orders</a></strong></h5>
        <h5 class="mr-sm-3"><strong><a href="{% url 'user_logout' %}" style="color: white">Logout</a></strong></h5>
        {% if request.user.profile.photo %}
          <a href="{% url 'profilepage' username=user.username %}"><img class="mr-sm-1" src="{{ request.user.profile.photo.url }}" width="35" height="35"></a>
        {% else %}
          <a href="{% url 'profilepage' username=user.username %}"><img class="mr-sm-1" src="/static/images/default.png" width="35" , height="35"></a>
        {% endif %}
```

```
              <h5><strong><a href="{% url 'profilepage' username=request.user.username %}" style="color: white">{{request.user.username}}</a></strong></h5>

          {% else %}
              <h5 class="mr-sm-3"><strong><a href="{% url 'user_login' %}" style="color: white">Login</a></strong></h5>
              <div>
                  <h5 class="mr-sm-3"><strong><a href="{% url 'register' %}" style="color: white">SignUp</a></strong></h5>
              </div>
          {% endif %}
                <a href="{% url 'cart' %}">
                    <img  id="cart-icon" src="{% static 'images/cart.png' %}">
                </a>
                <p id="cart-total">{{total_item_cart}}</p>

            </div>
          </div>
        </nav>

        <!-- <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script> -->
        <div class="jumbotron jumbotron-fluid">
            <div class="container container-fluid">
                {% block content %}
                {% endblock %}
            </div>
        </div>
        <footer class="container-fluid badge-dark my-0 py-3 text-light footer">
            <p class="mb-0 text-center">&copy; 2020-2021 MyCart.com</p>
            <p class="mb-0 text-center">
                <a href="#">Back to top |</a>
                <a href="#">Privacy |</a>
                <a href="#">Terms</a>
            </p>
        </footer>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
        <script src="https://unpkg.com/ionicons@5.1.2/dist/ionicons.js"></script>
        <script src="https://code.jquery.com/jquery-3.4.1.js" integrity="sha256-WpOohJOqMqqyKL9FccASB9O0KwACQJpFTUBLTYOVvVU=" crossorigin="anonymous"></script>

        <script type="text/javascript" src="{% static 'store/js/main.js' %}"></script>

        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

    </body>
    </html>
```

7. **Template/store.payment_success.html**

```
<blockquote class="blockquote text-center" style="text-align: center; padding-top: 100">
    <h1><p class="mb-0">** Payment Successful **</p></h1>
    <h4><p>Go Back ! Check your orders - </p></h4>
    <br>
    <a href="{% url 'order_details' %}" class="btn btn-secondary mb-2">View Order</a>
</blockquote>
```

8. Template/store.items_details.html

```
{% extends 'store/main.html' %}
{% load static%}
{% block content %}
    <div class="row my-2">
        <div class="col-lg-4 order-lg-1 text-center">
            <img src="{{product.get_imageURL}}" width="200" height="200">
        </div>
        <div class="col-lg-8 order-lg-2">
            <div class="tab-content py-4">
                <div class="tab-pane active" id="profile">
                    <h3 class="mb-3">{{product.name}}</h3>
                    <h4 class="mb-3">${{product.price}}</h4>
                    <div class="row">
                        <div class="col-md-6">

                            <p><strong>Description: </strong>{{product.des
cription}}</p>
                            <!-- <p><strong>E-
Mail: </strong>{{user.email}}</p> -->
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="row my-2">
        <div class="col-lg-4 order-lg-1 text-center">
            <button  class="update_cart btn btn-outline-secondary add-
btn" product_id ="{{product.id}}">Add to Cart</button>
        </div>
        <div class="col-lg-6 order-lg-2">
            <div class="quantity">
                <img action ="add" product_id ="{{product.id}}"  class="up
date_cart_quantity chg-quantity" src="{% static  'images/arrow-
up.png' %}">

                <img action ="remove" product_id ="{{product.id}}" class="
update_cart_quantity chg-quantity " src="{% static  'images/arrow-
down.png' %}">
            </div>
        </div>

    </div>
{% endblock %}
```

9. Template/store.order_status.html

```
{% extends 'store/main.html' %}
{% block content %}
    <h3>Ordered Items : </h3>
    <br>
    <div>
        {% if request.user.is_authenticated %}
            {% for order in ordered %}
                <div class="box-element" style="background-color: rgb(235, 233, 233)" ">
                        <div class="cart-row">
                            <div style="flex:1">
                                <strong>Order Placed</strong>
                                <h6>{{ order.order.date_ordered | date:"d M Y"}} </h6>
                            </div>
                            <div style="flex:1">
                                <strong>Transaction Id</strong>
                                <h6>{{order.order.transaction_id}}</h6>
                            </div>
                            <div style="flex:1">
                                <strong>Total</strong>
                                <h6>&#8377;{{order.order.amount}}</h6>
                            </div>
                            <div style="flex:1">
                                <strong>Shipped to :</strong>
                                <div>{{order.order.recepient_fullname}}({{order.order.phone_no}})</div>
                                <div>{{order.order.address_line1}}, {{order.order.address_line2}}</div>
                                <div>{{order.order.city}},{{order.order.state}},{{order.order.country}}</div>
                                <div>{{order.order.zipcode}}</div>
                            </div>
                        </div>
                    {% for item in order.items %}
                        <div class="cart-row" style="background-color: rgb(210, 223, 228);height: 80px">
```

```
                            <div style="flex:1"><img style="width: 80px; height: 80px;margin-left: 20px" class="row-image" src="{{item.get_imageURL}}"></div>
                            <div style="flex:1"><p>Product :{{item.name}}</p></div>
                            <div style="flex:1"><p>Price :&#8377;{{item.price}}</p></div>
                            <div style="flex:1">
                                <p class="quantity">Quantity :{{item.quantity}}</p>
                            </div>
                            <div style="flex:1"><p>Total :${{item.get_total}}</p></div>
                        </div>
                    {% endfor %}
                </div>
                <br>
            {% endfor %}
            <br>
        {% endif %}
    </div>
    <br><br>
    <blockquote class="blockquote text-center">
        <p class="mb-0">** That's it! **</p>
        <footer class="blockquote-footer"> You've scrolled to the end of the List!</footer>
    </blockquote>
{% endblock %}
```

11. urls.py

```
      from django.urls import path
from . import views

# app_name = 'app_store'

urlpatterns = [
    path('',views.store,name='store'),
    path('cart/',views.cart,name='cart'),
    path('checkout/',views.checkout,name='checkout'),
    path('insert_cart/',views.insert_into_cart,name='insert_cart'),
    path('cart/update_item/',views.update_item_quantity,name='update_item'),
    path('order_details/',views.order_details,name='order_details'),
    # path('tracker/',views.tracker,name='tracker'),
    path('item_detail/<int:id>',views.item_detail,name='item_detail'),
    path('make_payment/<int:id>',views.make_payment,name='make_payment'),
    path('delete_address/<int:id>', views.delete_address, name='delete_address'),
    path('show_items/<int:id>',views.show_items,name='show_items'),
    path('search/',views.search,name='search'),
    path('update_address/<int:id>',views.update_address,name='update_address')
]
```

12. models.py

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


class ProductCategories(models.Model):

    name = models.CharField(max_length=100,blank=False,null=True)
    image = models.ImageField(null=True,blank=False)

    @property
    def get_imageURL(self):
        try:
            url = self.image.url
        except:
            url = ''
        return url

    def __str__(self):
        return self.name



class Product(models.Model):

    category = models.ManyToManyField(ProductCategories)
    name = models.CharField(max_length=100,blank=False,name=False)
    price = models.FloatField(blank=False,null=False)
    image = models.ImageField(null=True,blank=True)
    description = models.TextField(null=True,blank=True)

    def __str__(self):
        return self.name
    @property
    def get_imageURL(self):
        try:
            url = self.image.url
        except:
            url = ''
        return url



class OrderItem(models.Model):

    user = models.ForeignKey(User,on_delete=models.CASCADE,null=True,blank=False)
    product = models.ForeignKey(Product,on_delete=models.CASCADE,null=True,blank=
False)
    date_ordered = models.DateTimeField(auto_now_add=True)
    quantity = models.IntegerField(default=0,blank=True,null=True)

    @property
    def get_total(self):
        total = self.product.price * self.quantity
        return total

    def __str__(self):
        return '{}-{}'.format(self.user.username,self.product.name)



class ShippingAddress(models.Model):

    user = models.ForeignKey(User,on_delete=models.CASCADE,null=True,blank=False)
    recepient_fullname = models.CharField(max_length=100,null=True,blank=False)
    phone_no = models.IntegerField(null=False,blank=False)
    address_line1 = models.CharField(max_length=200, null=True,blank=False)
    address_line2 = models.CharField(max_length=100,null=True,blank=True)
    city = models.CharField(max_length=200, null=False)
    state = models.CharField(max_length=200, null=False)
    country = models.CharField(max_length=100,null=True,blank=False)
    zipcode = models.CharField(max_length=200, null=False)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return '{}-{}'.format(self.address_line1, self.address_line2)
```

```
class FullOrder(models.Model):

    user = models.ForeignKey(User,on_delete=models.CASCADE,null=True)
    recepient_fullname = models.CharField(max_length=100, null=True, blank=False)
    phone_no = models.IntegerField(null=True, blank=False)
    address_line1 = models.CharField(max_length=200, null=True, blank=False)
    address_line2 = models.CharField(max_length=100, null=True, blank=True)
    city = models.CharField(max_length=200, null=True,blank=False)
    state = models.CharField(max_length=200, null=True,blank=False)
    country = models.CharField(max_length=100, null=True, blank=False)
    zipcode = models.CharField(max_length=200, null=True,blank=False)
    amount = models.IntegerField(null=True,blank=True)
    transaction_id = models.CharField(max_length=100,null=True,blank=False)
    date_ordered = models.DateTimeField(auto_now_add=True,null=True,blank=True)

    def __str__(self):
        return '{}-{}'.format(self.recepient_fullname,self.id)



class Purchased_item(models.Model):

    user = models.ForeignKey(User,on_delete=models.CASCADE,null=True)
    order = models.ForeignKey(FullOrder,on_delete=models.CASCADE,null=True)
    quantity = models.IntegerField(default=0, blank=True, null=True)
    name = models.CharField(max_length=100, blank=False, name=False)
    price = models.FloatField(blank=False, null=True)
    image = models.ImageField(null=True, blank=True)
    description = models.TextField(null=True, blank=True)

    @property
    def get_imageURL(self):
        try:
            url = self.image.url
        except:
            url = ''
        return url

    @property
    def get_total(self):
        total = self.price * self.quantity
        return total

    def __str_ (self):
        return self.name
```

13. views.py

```
from django.shortcuts import render
from .models import Product , OrderItem , ShippingAddress , FullOrder , Purchased
_item
from .models import ProductCategories
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from .forms import ShippingForm , ShippingUpdateForm
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse , HttpResponseRedirect ,Http404
from django.urls import reverse
import datetime
import json


# Create your views here.
def store(request):

    total_item_cart = 0

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity


    product_categories = ProductCategories.objects.all()

    context = {
        'product_categories' : product_categories,
        'total_item_cart' : total_item_cart,
    }
    return render(request, 'store/store.html', context)



def checkout(request):
    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('user_login'))

    items = []
    total_cost_cart = 0
    total_item_cart = 0

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

        for item in items:
            total_cost_cart += item.get_total

    if total_item_cart == 0:
        return Http404

    form = ShippingForm()
    if request.method == 'POST':
        form = ShippingForm(request.POST)
        if form.is_valid():
            adr = form.save(commit=False)
            adr.user = request.user
            adr.save()
        return HttpResponseRedirect(reverse('checkout'))

    addresses = ShippingAddress.objects.filter(user = request.user)

    product_categories = ProductCategories.objects.all()

    context = {
        'product_categories' : product_categories,
        'items': items,
        'total_item_cart': total_item_cart,
        'total_cost_cart': total_cost_cart,
        'form' : form,
        'addresses' : addresses,
    }
    return render(request, 'store/checkout.html', context)



@csrf_exempt
```

```
def insert_into_cart(request):

    total_item_cart = 0
    about = 'item_not_added'
    if request.user.is_authenticated:
        print('sparsh')
        about = 'Item Added'
        product_id = request.POST.get('product_id')
        product = Product.objects.get(id = product_id)
        if OrderItem.objects.filter(product=product,user = request.user).exists()
:
            item = OrderItem.objects.get(product=product,user = request.user)
            item.quantity += 1
            item.save()
        else:
            item = OrderItem.objects.create(product=product,user = request.user,q
uantity =1)
            item.save()

        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity


    dic = {
        'data' : about,
        'total_item_cart' : total_item_cart,
    }
    return JsonResponse(dic, safe=False)



@csrf_exempt
def update_item_quantity(request):
    about = 'Some Error Occurred'
    if request.user.is_authenticated:
        about = 'Item Updated'
        product_id = request.POST.get('product_id')
        action = request.POST.get('action')
        product = Product.objects.get(id=product_id)
        item = OrderItem.objects.get(product=product, user=request.user)

        if action == 'add':
            item.quantity+=1
        else:
            item.quantity-=1
        item.save()
        if item.quantity <= 0 :
            item.delete()

    dic = {
        'data': about,
    }
    return JsonResponse(dic,safe=False)



def cart(request):

    items = []
    total_cost_cart=0
    total_item_cart=0

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user = request.user)
        for item in items:
            total_item_cart += item.quantity

        for item in items:
            total_cost_cart += item.get_total

    if total_item_cart==0:
        check = False
    else:
        check = True

    product_categories = ProductCategories.objects.all()

    context = {
        'items' : items ,
        'total_item_cart' : total_item_cart,
        'total_cost_cart' : total_cost_cart,
        'check':check,
        'product_categories': product_categories,
    }
    return render(request, 'store/cart.html', context)
```

```python
def item_detail(request,id):

    total_item_cart = 0

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

    product = Product.objects.get(id=id)

    product_categories = ProductCategories.objects.all()

    context = {
        'product_categories': product_categories,
        'product' : product,
        'total_item_cart' : total_item_cart,
    }

    return render(request,'store/item_detail.html',context)



def order_details(request):

    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('user_login'))

    total_item_cart = 0
    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

    orders = FullOrder.objects.filter(user=request.user).order_by('-
date_ordered')

    ordered = []
    for order in orders:
        tt = []
        items = Purchased_item.objects.filter(order=order)
        for item in items:
            tt.append(item)
        ordered.append({'order': order, 'items': tt})
    product_categories = ProductCategories.objects.all()

    context = {
        'product_categories': product_categories,
        'ordered': ordered,
        'total_item_cart': total_item_cart,
    }
    return render(request,'store/order_details.html',context)



def make_payment(request,id):

    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('user_login'))

    dt = datetime.datetime.now()
    seq = int(dt.strftime("%Y%m%d%H%M%S"))

    adr = ShippingAddress.objects.get(id = id)
    obj = FullOrder.objects.create(user = request.user)

    obj.recepient_fullname = adr.recepient_fullname
    obj.phone_no = adr.phone_no
    obj.address_line1 = adr.address_line1
    obj.address_line2 = adr.address_line2
    obj.city = adr.city
    obj.state = adr.state
    obj.country = adr.country
    obj.zipcode = adr.zipcode
    obj.transaction_id = seq
    obj.save()

    total_amount = 0

    items = OrderItem.objects.all()
    for item in items:
        item_purchased = Purchased_item.objects.create(order = obj)
        item_purchased.user = request.user
        item_purchased.quantity = item.quantity
        item_purchased.name = item.product.name
        item_purchased.price = item.product.price
        item_purchased.image = item.product.image
        item_purchased.description = item.product.description
        item_purchased.save()
        total_amount += item.product.price * item.quantity

        item.delete()

    obj.amount = total_amount
    obj.save()

    return render(request,'store/payment_success.html')



def delete_address(request,id):

    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('user_login'))

    adr = ShippingAddress.objects.get(id=id)

    if adr.user != request.user:
        return Http404

    adr.delete()
    return HttpResponseRedirect(reverse('checkout'))



def show_items(request,id):

    total_item_cart = 0

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

    product_category = ProductCategories.objects.get(id=id)
    products = Product.objects.filter(category=product_category)

    product_categories = ProductCategories.objects.all()

    context = {
        'product_categories' : product_categories,
        'product_category' : product_category,
        'products': products,
        'total_item_cart': total_item_cart,
    }
    return render(request, 'store/show_items.html', context)



def search(request):
    total_item_cart = 0

    query = request.GET['search']

    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

    product_categories = ProductCategories.objects.all()
    products_temp = Product.objects.all()

    products =[]

    for p in products_temp:
        if query.lower() in p.name.lower() or query.lower() in p.description.lowe
r():
            products.append(p)

    context = {
        'products' : products,
        'product_categories': product_categories,
        'total_item_cart': total_item_cart,
    }

    return render(request, 'store/search.html', context)


def update_address(request,id):

    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('user_login'))

    total_item_cart = 0
    if request.user.is_authenticated:
        items = OrderItem.objects.filter(user=request.user)
        for item in items:
            total_item_cart += item.quantity

    product_categories = ProductCategories.objects.all()

    adr = ShippingAddress.objects.get(id=id)

    if adr.user != request.user:
        return Http404()

    if request.method == 'POST':
        form = ShippingUpdateForm(request.POST,instance=adr)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('checkout'))
    else:
        form = ShippingUpdateForm(instance=adr)

    context = {
        'product_categories' : product_categories,
        'total_item_cart' : total_item_cart,
        'form' : form ,
    }

    return render(request ,'store/update_address.html',context)
```

14. Manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'ecommerce.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

TESTING

This section introduces the concept of testing and how important is, for the successful implementation of the project. Different of testing are along with the level of testing incorporated in this particular project. Testing is vital to the success of any system. Testing is done at different stage within the phase. System testing makes a logical assumption that if all phases of the system are correct, the goals will be achieved successfully. System provide the Functionality that was expected.

System testing contain of the following steps: -

Black box-

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

White box-

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

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.

Alpha Testing-

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application –

Spelling Mistakes Broken Links Cloudy Directions

The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing-

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as pre- release testing. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following –

Users will install, run the application and send their feedback to the project team. Typographical errors, confusing application flow, and even crashes.

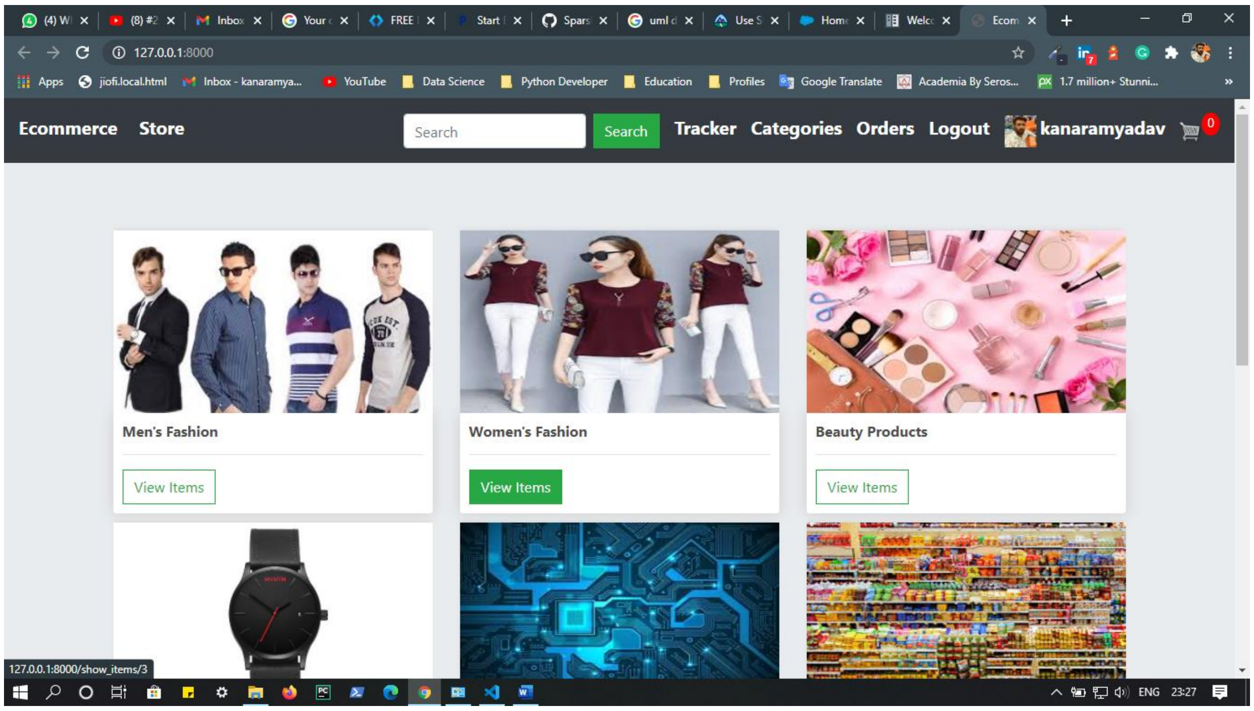
Getting the feedback, the project team can fix the problems before releasing the software to the actual users.

The more issues you fix that solve real user problems, the higher the quality of your application will be.

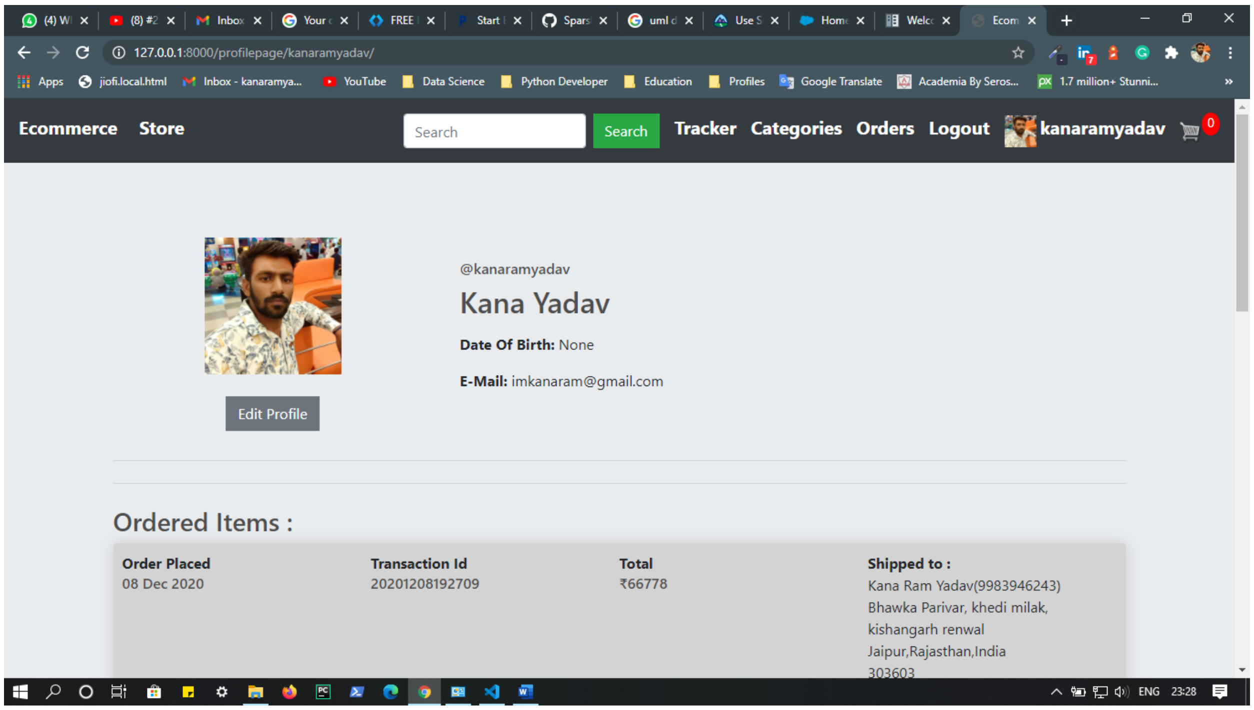
Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Screen Shot

1.



2.



127.0.0.1:8000/profilepage/kanaramyadav/

@kanaramyadav

Kana Yadav

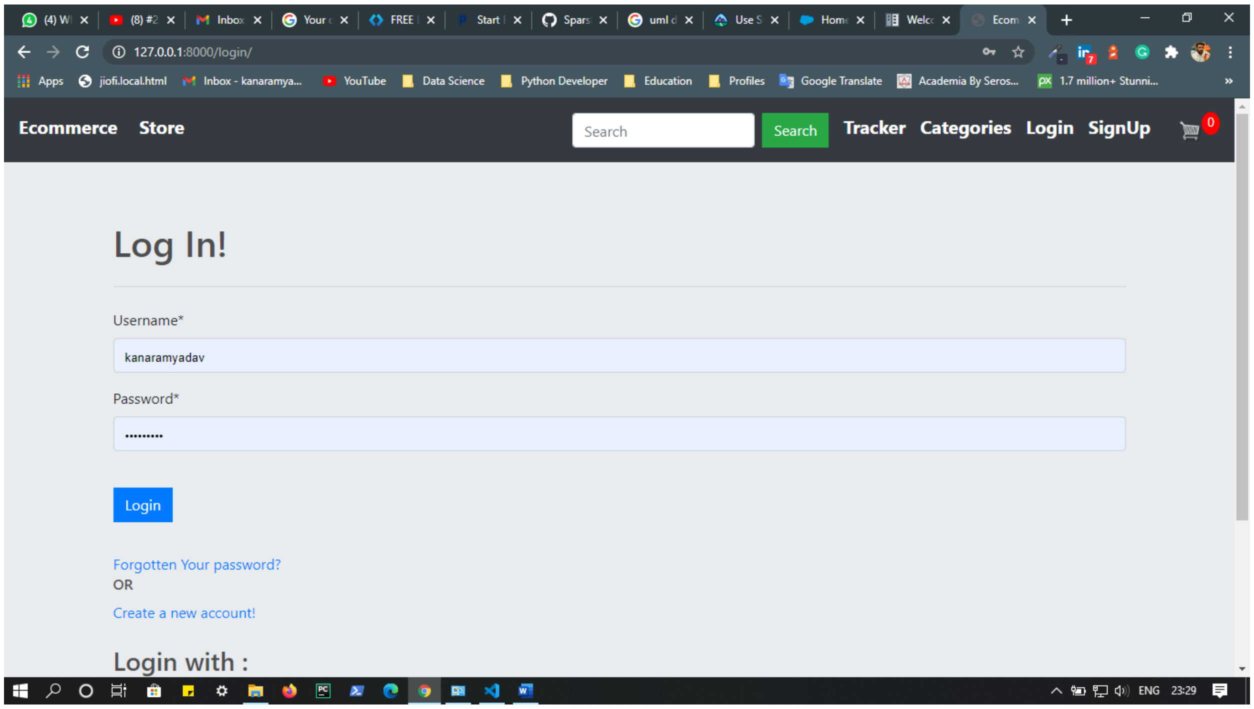
Date Of Birth: None

E-Mail: imkanaram@gmail.com

Ordered Items :

Order Placed	Transaction Id	Total	Shipped to :
08 Dec 2020	20201208192709	₹66778	Kana Ram Yadav(9983946243) Bhawka Parivar, khedi milak, kishangarh renwal Jaipur,Rajasthan,India 303603

3.



Ecommerce Store Search Tracker Categories Login SignUp

Log In!

Username*

kanaramyadav

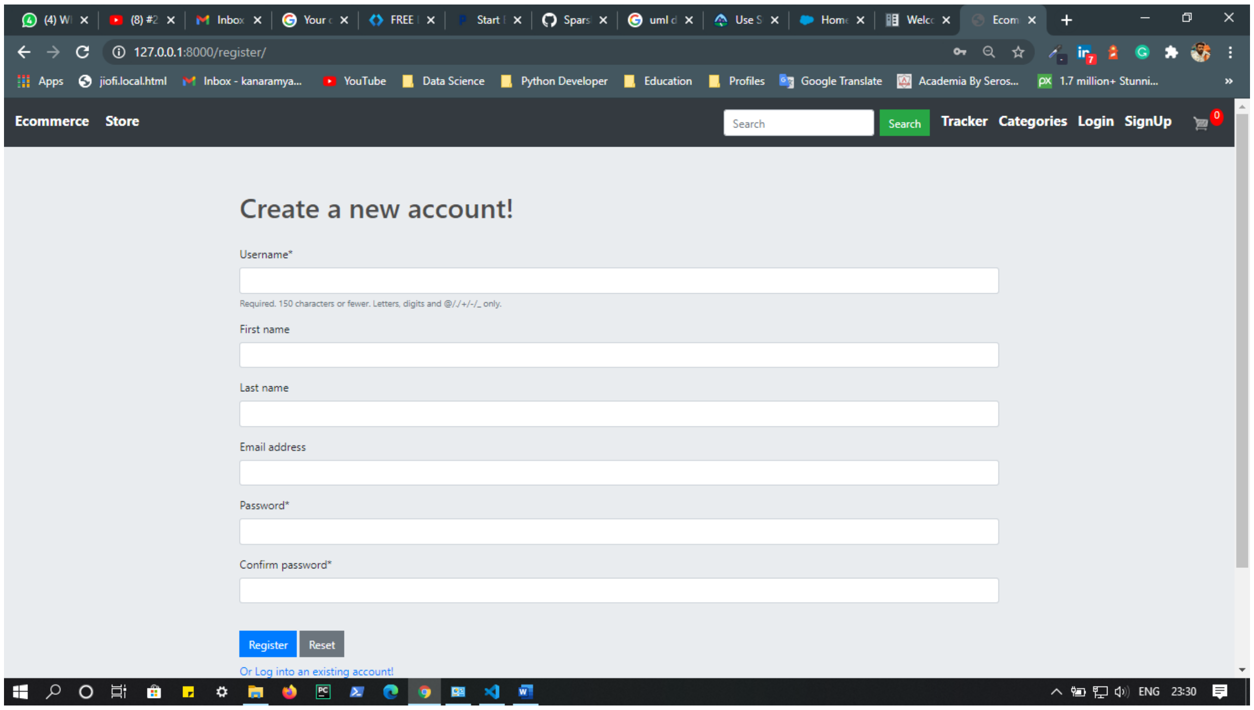
Password*

Login

Forgotten Your password?
OR
Create a new account!

Login with :

4.



Ecommerce Store Search Tracker Categories Login SignUp

Create a new account!

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

First name

Last name

Email address

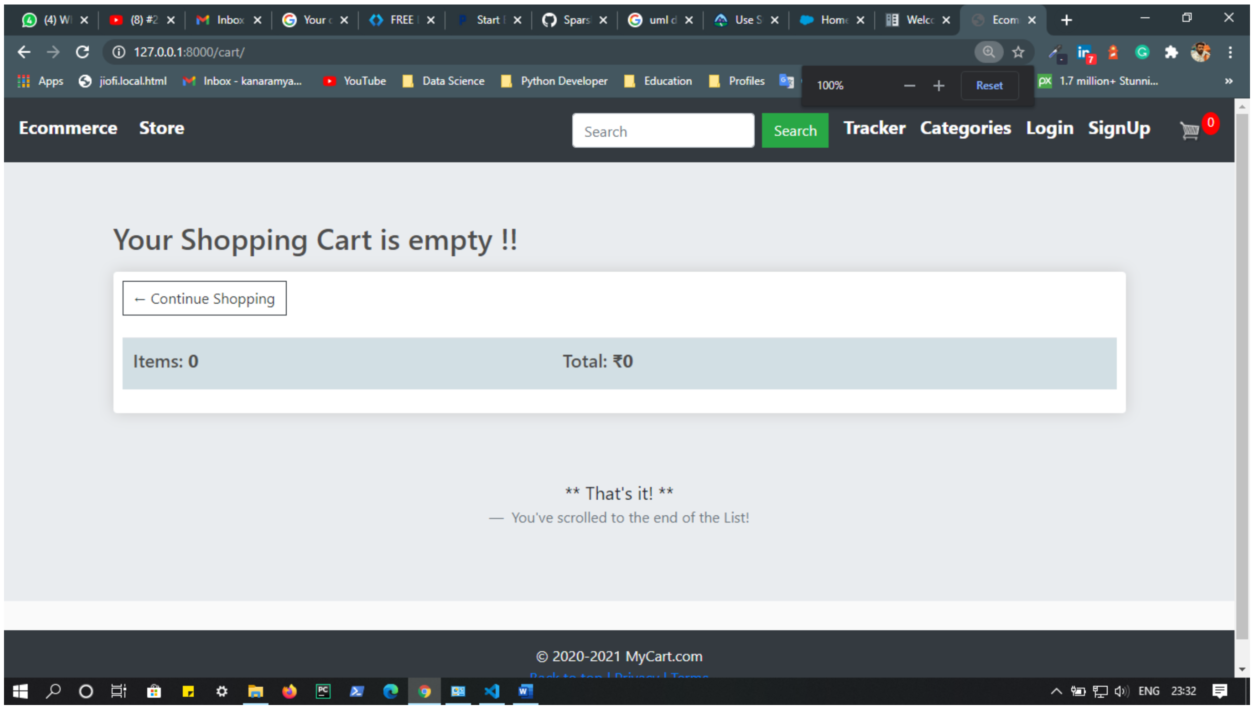
Password*

Confirm password*

Register Reset

Or Log into an existing account!

5.



Your Shopping Cart is empty !!

← Continue Shopping

Items: 0 Total: ₹0

** That's it! **

— You've scrolled to the end of the List!

© 2020-2021 MyCart.com

CONCLUSION

In general, today’s businesses must always strive to create the next best thing that consumers will want because consumers continue to desire their products, services etc. to continuously be better, faster, and cheaper. In this world of new technology, businesses need to accommodate to the new types of consumer needs and trends because it will prove to be vital to their business’ success and survival. E-commerce is continuously progressing and is becoming more and more important to businesses as technology continues to advance and is something that should be taken advantage of and implemented.

From the inception of the Internet and e-commerce, the possibilities have become endless for both businesses and consumers. Creating more opportunities for profit and advancements for businesses, while creating more options for consumers. However, just like anything else, e-commerce has its disadvantages including consumer uncertainties, but nothing that can not be resolved or avoided by good decision-making and business practices.

There are several factors and variables that need to be considered and decided upon when starting an e-commerce business. Some of these include: types of e-commerce, marketing strategies, and countless more. If the correct methods and practices are followed, a business will prosper in an e-commerce setting with much success and profitability.

BIBLIOGRAPHY

Websites: -

- https://docs.djangoproject.com/en/3.1/
- https://docs.python.org/3/

Books: -

- **Learn Python the Hard Way**
- **Python Pocket Reference: Python in Your Pocket**
- **Python - Andrew Johansen**
- **Think python**