**E-COMMERCE WEBSITE**

**Project Report**

*Submitted by*

Sudhanshu Chand 210240101107

Riya Singh 210240101081

Muskan Kumari 720240101007

Under the Supervision

Dr. Lokesh Kumar

Assistant Professor

CSE Department

In Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**ROORKEE INSTITUTE OF TECHNOLOGY, ROORKEE**

**(Affiliated to VMSB Uttarakhand Technical University, Dehradun)**

**APRIL 2024**



**DECLARATION**

I declare that the work embodied in this Minor report is my own original work carried out by me under the supervision of Dr. Lokesh Kumar for the session 2023-24 at **“Roorkee Institute of Technology”.** The matter embodied in this internship report has not been submitted elsewhere for the award of any other degree. I declare that I have faithfully acknowledged, given credit to and referred to the researchers wherever the work has been cited in the text and the body of the thesis. I further certify that I have not willfully lifted up some other’s work, Para, text, data, results, etc. reported in the journals, books, magazines, reports, dissertations, thesis, etc., or available at web-sites and have included them in this internship report and cited as my own work.

Date: Name(s) & Signature(s) of the Student

Place:

**ACKNOWLEDGEMENT**

I am very happy to greatly acknowledge the numerous personalities involved in lending their help to make our project “**E-Commerce Website**” a successful one.

I take this opportunity to express our deep sense of gratitude to our honorable Director “**Dr Parag Jain”** for providing an excellent academic climate in the college that made this endeavor possible.

I give my wholehearted admiration and a deep sense of gratitude to “**Dr. Prashant Verma**”**,** HOD, “**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**”, Roorkee Institute of Technology, Roorkee for his inspiration, valuable guidance, encouragement, suggestion, and overall help throughout.

I express my sincere thanks to supervisor **“Dr. Lokesh Kumar”, “Assistant Professor”,** “**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**”, **“ROORKEE INSTITUTE OF TECHNOLOGY”**, for his keen interest and invaluable help throughout the project.

We would like to express our sincere gratitude to our project coordinator **“Dr. Lokesh Kumar”, “Assistant Professor”,** “**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**”, “**ROORKEE INSTITUTE OF TECHNOLOGY”**, for his kind support and encouragement throughout this course of work.

Finally, we express our gratitude to all the Teaching and Non-Teaching staff of “**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING Roorkee**”, for their timely support and suggestions.

|  |  |
| --- | --- |
| Date: | Sudhanshu Chand 210240101107  Riya Singh 210240101081  Muskan Kumari 720240101007 |
|  |  |
|  |  |

**TABLE OF CONTENTS**

**Contents**

**ABSTRACT**

1. **INTRODUCTION**
2. **ANALYSIS**
3. **SOFTWARE REQUIREMENT SPECIFICATION**
4. **TECHNOLOGY USED AND ITS DESCRIPTION**
5. **CODING**
6. **SCREENSHOTS**
7. **CONCLUSION**
8. **BIBILOGRAPHY**

**ABSTRACT**

Electronic Commerce is process of doing business through computer networks. A person sitting on his chair in front of a computer can access all the facilities of the Internet to buy or sell the products.

Unlike traditional commerce that is carried out physically with effort of a person to go & get products, ecommerce has made it easier for human to reduce physical work and to save time.    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.

1. **INTRODUCTION**

E-commerce (electronic commerce or EC) is the buying and selling of goods and services on the Internet, especially the World Wide Web. In practice, this term and a newer term, e-business, are often used interchangeably . For online retail selling, the term e-tailing is sometimes used Electronic commerce is generally considered to be the sales aspect of e-business. It also consists of the exchange of data to facilitate the financing and payment aspects of e-business. It also consists of the exchange of data to facilitate the financing and payment aspects of business transactions.

E-commerce can be divided into

• E-tailing or "virtual storefronts" on Web sites with online catalogs, sometimes gathered into a "virtual mall"

• The gathering and use of demographic data through Web contacts

• Electronic Data Interchange (EDI), the business-to-business exchange of data

• E-mail and fax and their use as media for reaching prospects and established customers (for example, with newsletters)

• Business-to-business buying and selling

• The security of business transactions

Today Ecommerce is an integral part of business because of various reasons like:

• Ease of use & Accessibility all across the globe

• Great variety & easy compassion of products from different vendors

• Trusted payment channels

• Shopping can be done sitting in the convenience of home shopping, hence it is less time consuming. It is therefore very important for any new entrepreneur to understand the significance of E-Commerce and should know how to utilize this tool for the growth and development of business. So, whether you have an existing business or launching a brand new business, whether the volume of your business is large or small, you can always generate profit by demonstrating your products or services online, thereby acquiring a large amount of viewer exposure. In concise, buying and selling will result in profits and returns.

There are so many factors which makes e-commerce to come to the fore front in today's world. Saving precious time involved in business transactions is really a prominent factor. Like for instance, net banking makes it easy to carry out money and baking transactions in a break neck speed as compared to the real banking scenario. This asserts the fact that Ecommerce is beneficial to both business and consumer wise as payment and documentations can be completed with greater efficiency and reliability. Another important factor determining the flow of whole business is connectivity. Connectivity is very important for both consumers and business. Ecommerce provides better connectivity for all the potential candidates all over the globe, thus helping in enhancing the business without any geographical barriers. From the view point of the customer, Ecommerce is a good platform for hassle free shopping by sitting in your home. The customer can browse through all the products and services available and can review and compare the prices of the similar products available in the online space.

In global market scenario, the emergence of Ecommerce as a forerunner has opened up various windows of opportunities for a variety of online companies and investors. More and more resources are being directed into electronic securities, internet facilities, business plans and new technologies due to the boom in the space of E-commerce. As a result various new markets have emerged from Ecommerce itself giving a boost to the global market.

E-commerce shops have become part of our daily lives. Technological advancement has made it possible for people to sit in the convenience of their homes and still shop online without going to a physical shop. Africans have also joined the trend of e-commerce business, so this project is meant to design an eCommerce online shop so that the people in Ghana (Africa)will be able to purchase their goods and services online.

This project is mainly divided into two main categories: The Administrators and the Customers/Users.

The store manager and the staff members operate as the administrators. They can add, edit, update products or, delete products thus they able to change the names of products, change prices and, add or remove products.

The customer can search for products selection, update the cart, remove products from the cart and check out from the shop. The customer is also able to update his information such as names, address and other data.

The User is only able to browse the online shop and add a product to the cart. The user is limited to the use of the shop.

**2. ANALYSIS**

**Exchange and Share Data Across the World**

Text messages, documents, graphics, photographs, music, video, and much more, can be converted into data and sent as email attachments or presented on a web site. There are no postage, printing and packaging costs with eCommerce communications. So, use this new communications method to work quicker, and in new ways, with customers and suppliers outside of your locality or country.

**Send Messages Faster and Cheaper**

When you send a message, you only pay for the Internet access. As soon as the message arrives at your ISP, it is sent for zero cost to all the people to whom it has been addressed. Moreover, the message is likely to arrive at its destination, in a matter of seconds or minutes, depending on the size of the email and its attachments. Think of the possibilities for saving time and money with email.

**Approve and Proof Work Quickly**

A quotation, purchase order, letter, design drawings, brochure, or advert colour proof, etc. can be emailed to you as an attachment for your approval. Using email in this way can dramatically speed-up the turn-around of work between you and your business contacts.

**Introduce Collaborative Working**

With fast, reliable and cheap eCommerce communications, it is possible for colleagues, customers, suppliers and partners, to use collaborative working practices to manage, share, and enhance the project work, regardless of location where they are at a particular instance.

**Update Employees Instantly With New Policies or Procedures**

Email and messengers etc. works well when communicating with colleagues/pals who are off-site, out of the country, or teleworking.

**Hold Web Meetings (Data Conferencing)**

A great way to have a virtual meeting is to upload a document (spreadsheets, project plans, etc) on to a web site and enable it to be viewed and edited in real-time through the Internet. Holding virtual meetings can save significantly on travel expenses and time.

**Take Advantage of Time Differences**

As eCommerce communications are so fast and low cost, new business relationship have been established where work is exchanged between time zones at the end of one day, which, on the other side of the world, is the beginning of either the same or next day.

**Send Out email Automatically**

Event-triggered emails can be used to acknowledge orders made on web sites, and to update customers on order progress. Alert-emails or SMS messages can also be used to notify technical support of critical issues or maintenance problems. There, and many other similar applications, are effortless, cheap, and efficient ways of enhancing customer service and the brand experience.

**Use the Internet to Improve Business Administration**

With a mobile phone connected to a portable computer, employees working off-site can access and update internal information, such as customer records, price lists, time sheets, schedules, and job reports. Providing remote Internet access for staff makes it possible for the latest information to be at hand, for administrative tasks such as invoicing, employee whereabouts, and expense claims etc.

**Train on the Web**

The multi-media capabilities of web sites make them ideal for creating a virtual learning environment for employees and customers. When training material is updated, the latest version is immediately available. On-line learning facilities enable trainees living all over the world to access courseware when it is convenient to them, and without having to travel to a training centre, which in turn, saves time and expenses.

**Replace the Fax Machine and Save Money**

If a multi-page document has to be faxed to many people, it can work out expensive on telephone bills, especially if a national or international call has to be made. The cost savings can be huge if email is used instead, and the message can also be sent to multiple people for no extra cost.

1. **SOFTWARE REQUIREMENTS SPECIFICATIONS**

3.1 System configurations

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behaviour, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

**Software Requirements:**

● **Hardware System Configuration**

Any device to browse the internet

● **Technology to be used:**

HTML, CSS, JAVASCRIPT, React, Bootstrap MongoDB.

● **Software System Configuration:**

Browsers: Chrome , Firefox ,Microsoft Edge ,Internet Explorer, etc.

1. **TECHNOLOGY USED AND ITS DESCRIPTION**

**Frontend:**

* HTML (Hypertext Markup Language): The standard markup language for creating web pages and web applications.
* CSS (Cascading Style Sheets): Used for styling the appearance of HTML elements on web pages.
* JavaScript: A programming language commonly used for creating interactive effects within web browsers.

**Backend:**

* Spring Framework (Spring Boot): A popular Java framework for building enterprise-level applications. Spring Boot simplifies the setup and configuration of Spring-based applications, enabling rapid development.

**Database:**

* MySQL: An open-source relational database management system (RDBMS) that uses structured query language (SQL). It's commonly used for storing structured data in web applications.

**Programming Language:**

* Python: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

1. **CODING**

**Frontend:-**

**Index.html:-**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

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

<meta name="theme-color" content="#000000" />

<meta name="description" content="Web site created using create-react-app" />

<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />

<!--

manifest.json provides metadata used when your web app is installed on a

user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/

-->

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"

integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l" crossorigin="anonymous" />

<!-- fontawesome -->

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css" />

<!--

Notice the use of %PUBLIC\_URL% in the tags above.

It will be replaced with the URL of the `public` folder during the build.

Only files inside the `public` folder can be referenced from the HTML.

Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running `npm run build`.

-->

<title>React App</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

<!--

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.

To create a production bundle, use `npm run build` or `yarn build`.

-->

</body>

</html>

**manifest.json:-**

{

"short\_name": "React App",

"name": "Create React App Sample",

"icons": [

{

"src": "favicon.ico",

"sizes": "64x64 32x32 24x24 16x16",

"type": "image/x-icon"

},

{

"src": "logo192.png",

"type": "image/png",

"sizes": "192x192"

},

{

"src": "logo512.png",

"type": "image/png",

"sizes": "512x512"

}

],

"start\_url": ".",

"display": "standalone",

"theme\_color": "#000000",

"background\_color": "#ffffff"

}

**index.js:-**

import React from 'react';

import ReactDOM from 'react-dom';

import {Provider} from 'react-redux'

import store from './store'

import App from './App';

import "./App.css"

ReactDOM.render(

<Provider store = {store}>

<App />

</Provider>,

document.getElementById('root')

);

**App.js:-**

import React from 'react'

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'

import ProductListPage from './pages/ProductsListPage'

import ProductDetailsPage from './pages/ProductDetailsPage'

import CheckoutPage from './pages/CheckoutPage'

import NavBar from './components/Navbar'

import PaymentStatus from './components/PaymentStatus'

import Login from './pages/LoginPage'

import Register from './pages/RegisterPage'

import CardUpdatePage from './pages/CardUpdatePage'

import CardDetailsPage from './pages/CardDetailsPage'

import AccountPage from './pages/AccountPage'

import AccountUpdatePage from './pages/AccountUpdatePage'

import DeleteUserAccountPage from './pages/DeleteUserAccountPage'

import AllAddressesOfUserPage from './pages/AllAddressesOfUserPage'

import AddressUpdatePage from './pages/AddressUpdatePage'

import OrdersListPage from './pages/OrdersListPage'

import ProductCreatePage from './pages/ProductCreatePage'

import ProductUpdatePage from './pages/ProductUpdatePage'

import NotFound from './pages/NotFoundPage'

const App = () => {

return (

<div>

<Router>

<NavBar />

<div className="container mt-4">

<Switch>

<Route path="/" component={ProductListPage} exact />

<Route path="/new-product/" component={ProductCreatePage} exact />

<Route path="/product/:id/" component={ProductDetailsPage} exact />

<Route path="/product-update/:id/" component={ProductUpdatePage} exact />

<Route path="/product/:id/checkout/" component={CheckoutPage} exact />

<Route path="/payment-status" component={PaymentStatus} exact />

<Route path="/login" component={Login} exact />

<Route path="/register" component={Register} exact />

<Route path="/account" component={AccountPage} exact />

<Route path="/account/update/" component={AccountUpdatePage} exact />

<Route path="/account/delete/" component={DeleteUserAccountPage} exact />

<Route path="/stripe-card-details" component={CardDetailsPage} exact />

<Route path="/stripe-card-update" component={CardUpdatePage} exact />

<Route path="/all-addresses/" component={AllAddressesOfUserPage} exact />

<Route path="/all-addresses/:id/" component={AddressUpdatePage} exact />

<Route path="/all-orders/" component={OrdersListPage} exact />

<Route path="" component={NotFound} exact />

</Switch>

</div>

</Router>

</div >

)

}

export default App

**App.css:-**

.App {

text-align: center;

}

.App-logo {

height: 40vmin;

pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

.App-logo {

animation: App-logo-spin infinite 20s linear;

}

}

.App-header {

background-color: #282c34;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;

}

.App-link {

color: #61dafb;

}

.searchbox-css {

height: 10px;

}

.justify-description-css {

text-align: justify;

white-space: pre-line;

}

.edit-button-css {

position: relative;

top: -30px;

float: right;

color: grey;

cursor: pointer;

transition: transform .2s; /\* Animation \*/

}

.edit-button-css:hover {

color: #6699ff;

transform: scale(1.3)

}

.delete-button-css {

position: relative;

top: -30px;

float: right;

color: grey;

cursor: pointer;

transition: transform .2s; /\* Animation \*/

}

.delete-button-css:hover {

color: red;

transform: scale(1.3)

}

.saved-cards-css {

color: black;

}

.saved-cards-css:hover {

cursor: pointer;

background-color: #e6ffff;

}

.button-focus-css:focus {

box-shadow: none !important;

}

.footer-css {

position: fixed;

padding: 10px 10px 0px 10px;

bottom: 0;

width: 100%;

/\* Height of the footer\*/

height: 40px;

background: grey;

}

@keyframes App-logo-spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

**Store.js:-**

import {createStore, applyMiddleware} from 'redux'

import { composeWithDevTools} from 'redux-devtools-extension'

import thunk from 'redux-thunk'

import allReducers from './reducers/index'

const middleware = [thunk]

const userInfoFromStorage = localStorage.getItem('userInfo') ? JSON.parse(localStorage.getItem('userInfo')) : null

let initialState = {

userLoginReducer: { userInfo: userInfoFromStorage }

}

const store = createStore(allReducers, initialState, composeWithDevTools(applyMiddleware(...middleware)))

export default store

**Backend:-**

**payments:-views.py**

import stripe

from rest\_framework import status

from rest\_framework import permissions

from rest\_framework.views import APIView

from rest\_framework.response import Response

from account.models import StripeModel, OrderModel

from rest\_framework.decorators import permission\_classes

from datetime import datetime

# stripe secret test key

stripe.api\_key="your secret key here"

def save\_card\_in\_db(cardData, email, cardId, customer\_id, user):

# save card in django stripe model

StripeModel.objects.create(

email = email,

customer\_id = customer\_id,

card\_number=cardData["number"],

exp\_month = cardData["exp\_month"],

exp\_year = cardData["exp\_year"],

card\_id = cardId,

user = user,

)

# Just for testing

class TestStripeImplementation(APIView):

def post(self, request):

test\_payment\_process = stripe.PaymentIntent.create(

amount=120,

currency='inr',

payment\_method\_types=['card'],

receipt\_email='yash@gmail.com'

)

return Response(data=test\_payment\_process, status=status.HTTP\_200\_OK)

# check token expired or not

class CheckTokenValidation(APIView):

permission\_classes = [permissions.IsAuthenticated]

def get(self, request):

return Response("Token is Valid", status=status.HTTP\_200\_OK)

# to create card token (to validate your card)

class CreateCardTokenView(APIView):

permission\_classes = [permissions.IsAuthenticated]

def post(self, request):

card\_invalid = False

data = request.data

email = request.data["email"]

cardStatus = request.data["save\_card"]

card\_info = data["number"]

client\_card = card\_info[slice(12, 16)] # only last 4 digits of card

# checking for valid user (email associated with card will be checked)

customer\_data = stripe.Customer.list().data

user\_data = []

for each in customer\_data:

the\_card = each.sources.data[0].last4

user\_data.append({"user": {"card\_num": the\_card, "card\_holder": each.email}})

for each in user\_data:

user\_info = each["user"]

user\_card\_info = user\_info["card\_num"]

user\_email\_info = user\_info["card\_holder"]

if user\_card\_info == client\_card:

if user\_email\_info != email:

return Response({

"detail": "Your email address does not belong to the provided card." },

status=status.HTTP\_400\_BAD\_REQUEST)

try:

stripeToken = stripe.Token.create(

card = {

"number": data["number"],

"exp\_month": data["exp\_month"],

"exp\_year": data["exp\_year"],

"cvc": data["cvc"]

},

)

except stripe.error.CardError as e:

errorMessage = e.user\_message # as per stripe documentation

return Response({ "detail": errorMessage}, status=status.HTTP\_400\_BAD\_REQUEST)

except stripe.error.APIConnectionError:

return Response({ "detail": "Network error, Failed to establish a new connection."}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

customer\_data = stripe.Customer.list(email=email).data

if len(customer\_data) == 0:

# create customer in stripe (will provide us customer id in response)

customer = stripe.Customer.create(

email = request.data["email"],

description="My new customer"

)

else:

customer = customer\_data[0]

message = "Customer already exists"

actual\_cn = customer.sources.data[0].last4 # holds card number (last four digits)

actual\_em = customer.sources.data[0].exp\_month

actual\_ey = customer.sources.data[0].exp\_year

recieved\_cn = data["number"]

last4\_recieved\_cn = recieved\_cn[-4:]

recieved\_em = data["exp\_month"]

recieved\_ey = data["exp\_year"]

# comparing the last4 digits of card provided by the user with the last4 digits of card present on stripe

if actual\_cn != last4\_recieved\_cn or actual\_em != recieved\_em or actual\_ey != recieved\_ey:

card\_invalid = True

if card\_invalid:

return Response({"detail": "Invalid Card Details Provided."}, status=status.HTTP\_400\_BAD\_REQUEST)

else:

# creating a card on stripe (getting validated also by the stipe token)

create\_user\_card = stripe.Customer.create\_source(

customer["id"],

source=stripeToken.id,

)

# card id got generated at this point

if cardStatus:

try:

save\_card\_in\_db(data, email, create\_user\_card.id, customer["id"], request.user)

message = {"customer\_id": customer["id"], "email": email, "card\_data": create\_user\_card}

return Response(message, status=status.HTTP\_200\_OK)

except:

return Response({

"detail": "Card already in use, please uncheck save card option or select a card from saved card list."},

status=status.HTTP\_400\_BAD\_REQUEST)

else:

try:

message = {"customer\_id": customer["id"], "email": email, "card\_data": create\_user\_card}

return Response(message, status=status.HTTP\_200\_OK)

except:

return Response({ "detail": "Network Error, please check your internet connection."})

# Charge the customer card

class ChargeCustomerView(APIView):

permission\_classes = [permissions.IsAuthenticated]

def post(self, request):

try:

data = request.data

email = request.data["email"]

customer\_data = stripe.Customer.list(email=email).data

customer = customer\_data[0]

customer\_data = stripe.Customer.list(email=request.data["email"]).data

# make stripe payment (charge the customer) (either use charge api or paymentIntent api)

stripe.Charge.create(

customer=customer\_data[0],

amount=int(float(request.data["amount"])\*100),

currency="inr",

description='Software development services', # required for Indian transactions

)

# saving order in django database

new\_order = OrderModel.objects.create(

name = data["name"],

card\_number = data["card\_number"],

address = data["address"],

ordered\_item = data["ordered\_item"],

paid\_status = data["paid\_status"],

paid\_at = datetime.now(),

total\_price = data["total\_price"],

is\_delivered = data["is\_delivered"],

delivered\_at = data["delivered\_at"],

user = request.user

)

return Response(

data = {

"data": {

"customer\_id": customer.id,

"message": "Payment Successfull",

}

}, status=status.HTTP\_200\_OK)

except stripe.error.APIConnectionError:

return Response({

"detail": "Network error, Failed to establish a new connection."},

status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

)

# retrieve card (to get user card details)

class RetrieveCardView(APIView):

permission\_classes = [permissions.IsAuthenticated]

def get(self, request):

card\_details = stripe.Customer.retrieve\_source(

request.headers["Customer-Id"],

request.headers["Card-Id"]

)

return Response(card\_details, status=status.HTTP\_200\_OK)

# update a card

class CardUpdateView(APIView):

permission\_classes = [permissions.IsAuthenticated]

def post(self, request):

data = request.data

update\_card = stripe.Customer.modify\_source(

data["customer\_id"],

data["card\_id"],

exp\_month = data["exp\_month"] if data["exp\_month"] else None,

exp\_year = data["exp\_year"] if data["exp\_year"] else None,

name = data["name\_on\_card"] if data["name\_on\_card"] else None,

address\_city = data["address\_city"] if data["address\_city"] else None,

address\_country = data["address\_country"] if data["address\_country"] else None,

address\_state = data["address\_state"] if data["address\_state"] else None,

address\_zip = data["address\_zip"] if data["address\_zip"] else None,

)

# locating stripe object in django database

obj = StripeModel.objects.get(card\_number=request.data["card\_number"])

# updating stripe object in django database

if obj:

obj.name\_on\_card = data["name\_on\_card"] if data["name\_on\_card"] else obj.name\_on\_card

obj.exp\_month = data["exp\_month"] if data["exp\_month"] else obj.exp\_month

obj.exp\_year = data["exp\_year"] if data["exp\_year"] else obj.exp\_year

obj.address\_city = data["address\_city"] if data["address\_city"] else obj.address\_city

obj.address\_country = data["address\_country"] if data["address\_country"] else obj.address\_country

obj.address\_state = data["address\_state"] if data["address\_state"] else obj.address\_state

obj.address\_zip = data["address\_zip"] if data["address\_zip"] else obj.address\_zip

obj.save()

else:

pass

return Response(

{

"detail": "card updated successfully",

"data": { "Updated Card": update\_card },

}, status=status.HTTP\_200\_OK)

# delete card

class DeleteCardView(APIView):

permission\_classes = [permissions.IsAuthenticated]

def post(self, request):

data = request.data

obj\_card = StripeModel.objects.get(card\_number=request.data["card\_number"])

customerId = obj\_card.customer\_id

cardId = obj\_card.card\_id

# deleting card from stripe

stripe.Customer.delete\_source(

customerId,

cardId

)

# deleting card from django database

obj\_card.delete()

# delete the customer

# as deleting the card will not change the default card number on stripe therefore

# we need to delete the customer (with a new card request customer will be recreated)

stripe.Customer.delete(customerId)

return Response("Card deleted successfully.", status=status.HTTP\_200\_OK)

**urls.py:-**

from django.urls import path

from payments import views

urlpatterns = [

path('test-payment/', views.TestStripeImplementation.as\_view()),

path('create-card/', views.CreateCardTokenView.as\_view()),

path('charge-customer/', views.ChargeCustomerView.as\_view()),

path('update-card/', views.CardUpdateView.as\_view()),

path('delete-card/', views.DeleteCardView.as\_view()),

path('card-details/', views.RetrieveCardView.as\_view()),

path('check-token/', views.CheckTokenValidation.as\_view()),

]

**admin.py:-**

from django.contrib import admin

from .models import StripeModel, BillingAddress, OrderModel

class StripeModelAdmin(admin.ModelAdmin):

list\_display = ("id", "email", "card\_number", "user", "exp\_month", "exp\_year", "customer\_id", "card\_id")

class BillingAddressModelAdmin(admin.ModelAdmin):

list\_display = ("id", "name", "user", "phone\_number", "pin\_code", "house\_no", "landmark", "city", "state")

class OrderModelAdmin(admin.ModelAdmin):

list\_display = ("id", "name", "card\_number", "address", "ordered\_item", "paid\_status", "paid\_at", "total\_price", "is\_delivered", "delivered\_at", "user")

admin.site.register(StripeModel, StripeModelAdmin)

admin.site.register(BillingAddress, BillingAddressModelAdmin)

admin.site.register(OrderModel, OrderModelAdmin)

**serializers.py:-**

from .models import StripeModel, BillingAddress, OrderModel

from rest\_framework import serializers

from django.contrib.auth.models import User

from rest\_framework\_simplejwt.tokens import RefreshToken

class UserSerializer(serializers.ModelSerializer):

admin = serializers.SerializerMethodField(read\_only=True)

class Meta:

model = User

fields = ["id", "username", "email", "admin"]

def get\_admin(self, obj):

return obj.is\_staff

# creating tokens manually (with user registration we will also create tokens)

class UserRegisterTokenSerializer(UserSerializer):

token = serializers.SerializerMethodField(read\_only=True)

class Meta:

model = User

fields = ["id", "username", "email", "admin", "token"]

def get\_token(self, obj):

token = RefreshToken.for\_user(obj)

return str(token.access\_token)

# list of cards

class CardsListSerializer(serializers.ModelSerializer):

class Meta:

model = StripeModel

fields = "\_\_all\_\_"

# billing address details

class BillingAddressSerializer(serializers.ModelSerializer):

class Meta:

model = BillingAddress

fields = "\_\_all\_\_"

# all orders list

class AllOrdersListSerializer(serializers.ModelSerializer):

class Meta:

model = OrderModel

fields = "\_\_all\_\_"

**tests.py:-**

from account import views

from django.http import response

from django.test import TestCase, Client

from django.urls import reverse

from django.utils import timezone

from rest\_framework.test import APITestCase

from rest\_framework.test import force\_authenticate

from rest\_framework.test import APIRequestFactory

from django.contrib.auth.models import User

from rest\_framework.test import force\_authenticate

from .models import BillingAddress, OrderModel, StripeModel

from .views import CardsListView, ChangeOrderStatus, CreateUserAddressView, DeleteUserAddressView, OrdersListView, UpdateUserAddressView, UserAccountDeleteView, UserAccountDetailsView, UserAccountUpdateView, UserAddressDetailsView, UserAddressesListView

class AccountApisSetUp(APITestCase):

def setUp(self):

self.register\_url = reverse("register-page")

self.login\_url = reverse("login-page")

self.user\_data = {

"email": "yashmarmat08@gmail.com",

"username": "yash",

"password": "yash1234"

}

self.empty\_fields = {

"email": "",

"username": "",

"password": ""

}

self.admin\_user = User.objects.create\_superuser(

username = "admin",

email = "admin@gmail.com",

password = "admin1234"

)

self.normal\_user = User.objects.create\_user(

username = "testuser",

email = "testuser@gmail.com",

password = "testuser1234"

)

self.dummy\_address = BillingAddress.objects.create(

name = "testuser",

user = self.normal\_user,

phone\_number = "9123456789",

pin\_code = "110000",

house\_no = "somewhere on earth",

landmark = "near shop",

city = "new delhi",

state = "delhi",

)

self.dummy\_order = OrderModel.objects.create(

name = "testuser",

ordered\_item = "computer chair",

card\_number = "4242424242424242",

address = "somewhere on earth",

paid\_status = "True",

paid\_at = timezone.now(),

total\_price = "5999.99",

is\_delivered = "False",

delivered\_at = "Not Delivered",

user = self.normal\_user

)

# stripe card of testuser

self.testuser\_stripe\_card = StripeModel.objects.create(

email = "testuser@gmail.com",

name\_on\_card = "testuser",

customer\_id = "cus\_1234",

card\_number = "1234123412341234",

exp\_month = "08",

exp\_year = "2024",

card\_id = "card\_1234",

user = self.normal\_user,

address\_city = "New Delhi",

address\_country = "INDIA",

address\_state = "unknown",

address\_zip = "111222"

)

# stripe card of admin

self.admin\_stripe\_card = StripeModel.objects.create(

email = "admin@gmail.com",

name\_on\_card = "admin",

customer\_id = "cus\_1111",

card\_number = "4242424242424242",

exp\_month = "05",

exp\_year = "2026",

card\_id = "card\_5555",

user = self.admin\_user,

address\_city = "New York",

address\_country = "USA",

address\_state = "unknown",

address\_zip = "511111"

)

class AccountApisAuthTest(AccountApisSetUp):

def test\_user\_registration\_without\_user\_data(self):

response = self.client.post(

self.register\_url,

self.empty\_fields,

format="json"

)

# import pdb

# pdb.set\_trace()

self.assertEqual(response.status\_code, 400)

def test\_user\_login\_without\_user\_data(self):

response = self.client.post(

self.login\_url,

self.empty\_fields,

format="json"

)

self.assertEqual(response.status\_code, 400)

def test\_user\_registration\_with\_user\_data(self):

response = self.client.post(

self.register\_url, self.user\_data, format="json"

)

self.assertEqual(response.status\_code, 200)

def test\_user\_login\_with\_user\_data(self):

# register

self.client.post(

self.register\_url, self.user\_data, format="json"

)

# login

res = self.client.post(

self.login\_url, self.user\_data, format="json"

)

# to see request response use below code (then in terminal type response.data to see details)

# import pdb

# pdb.set\_trace()

self.assertEqual(res.status\_code, 200)

def test\_user\_account\_details\_page\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='admin')

view = UserAccountDetailsView.as\_view()

request = factory.get('/accounts/user/1/')

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 200)

def test\_user\_account\_details\_page\_when\_logged\_out(self):

response = self.client.get('/accounts/user/1/')

self.assertEqual(response.status\_code, 404)

def test\_user\_account\_updation\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='admin')

view = UserAccountUpdateView.as\_view()

# only updating the username (you can update anything though)

updates = {"username": "admin22", "email": "", "password": ""}

request = factory.put('/accounts/user\_update/1/', updates)

force\_authenticate(request, user=user)

update\_user = User.objects.get(id='1')

update\_user.username = updates["username"]

update\_user.save()

response = view(request, 1)

#print(User.objects.get(id=1)) # you can check the changes like this

self.assertEqual(response.status\_code, 200)

def test\_user\_account\_updation\_when\_logged\_out(self):

updates = {"username": "admin22", "email": "", "password": ""}

response = self.client.put('/accounts/user\_update/1/', updates)

self.assertEqual(response.status\_code, 404)

def test\_user\_account\_deletion\_with\_wrong\_password(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = UserAccountDeleteView.as\_view()

request = factory.post('/accounts/user\_delete/2/', {"password": "testuser12"})

force\_authenticate(request, user=user)

response = view(request, 2)

self.assertEqual(response.status\_code, 401) # Unauthorized

def test\_user\_account\_deletion\_with\_correct\_password(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = UserAccountDeleteView.as\_view()

request = factory.post('/accounts/user\_delete/2/', {"password": "testuser1234"})

force\_authenticate(request, user=user)

response = view(request, 2)

self.assertEqual(response.status\_code, 204)

def test\_user\_account\_deletion\_without\_login(self):

response = self.client.post(('/accounts/user\_delete/1/', {"password": "admin1234"}))

self.assertEqual(response.status\_code, 404)

def test\_get\_all\_the\_addresses\_of\_user\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username="testuser")

view = UserAddressesListView.as\_view()

request = factory.get('/accounts/all-addresses-details/')

force\_authenticate(request, user=user)

response = view(request)

self.assertEqual(response.status\_code, 200)

def test\_get\_all\_addresses\_of\_user\_when\_logged\_out(self):

response = self.client.get('/accounts/all-addresses-details/')

self.assertEqual(response.status\_code, 404)

def test\_address\_creation\_of\_registered\_user(self):

factory = APIRequestFactory()

user = User.objects.get(username="testuser")

view = CreateUserAddressView.as\_view()

user\_address = {

"name": "testuser",

"user": user,

"phone\_number": "9123456789",

"pin\_code": "110000",

"house\_no": "somewhere on earth",

"landmark": "near shop",

"city": "new delhi",

"state": "delhi",

}

request = factory.post('/account/create-address/', user\_address)

force\_authenticate(request, user=user)

response = view(request)

self.assertEqual(response.status\_code, 200)

def test\_address\_creation\_api\_when\_logged\_out(self):

user = User.objects.get(username="testuser")

user\_address = {

"name": "testuser",

"user": user,

"phone\_number": "9123456789",

"pin\_code": "110000",

"house\_no": "somewhere on earth",

"landmark": "near shop",

"city": "new delhi",

"state": "delhi",

}

response = self.client.post('/account/create-address/', user\_address)

self.assertEqual(response.status\_code, 401) # Unauthorized

def test\_address\_updation\_of\_registered\_user(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = UpdateUserAddressView.as\_view()

# updating the dummy address here

updated\_address = {

"name": "",

"user": user,

"phone\_number": "",

"pin\_code": "111111",

"house\_no": "",

"landmark": "",

"city": "new york",

"state": "",

}

request = factory.put('/account/update-address/1/', updated\_address)

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 200)

def test\_address\_updation\_when\_logged\_out(self):

user = User.objects.get(username='testuser')

# updating the dummy address here

updated\_address = {

"name": "",

"user": user,

"phone\_number": "",

"pin\_code": "111111",

"house\_no": "",

"landmark": "",

"city": "new york",

"state": "",

}

response = self.client.put('/account/update-address/1/', updated\_address)

self.assertEqual(response.status\_code, 401) # Unauthorized

def test\_fetching\_address\_details\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = UserAddressDetailsView.as\_view()

request = factory.get('/accounts/address-details/1/') # 1 is the id of the address (created above)

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 200)

def test\_fetching\_address\_details\_when\_logged\_out(self):

response = self.client.get('/accounts/address-details/1/') # 1 is the id of the address (created above)

self.assertEqual(response.status\_code, 404)

def test\_delete\_user\_address\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = DeleteUserAddressView.as\_view()

request = factory.delete('/account/delete-address/1/')

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 204)

def test\_delete\_user\_address\_when\_logged\_out(self):

response = self.client.delete('/account/delete-address/1/')

self.assertEqual(response.status\_code, 403) # Forbidden

def test\_get\_orders\_list\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = OrdersListView.as\_view()

request = factory.get('/account/all-orders-list/')

force\_authenticate(request, user=user)

response = view(request)

self.assertEqual(response.status\_code, 200)

def test\_get\_orders\_list\_when\_logged\_out(self):

response = self.client.get('/account/all-orders-list/')

self.assertEqual(response.status\_code, 401) # Unauthorized

def test\_changing\_of\_order\_status\_by\_admin(self):

factory = APIRequestFactory()

user = User.objects.get(username='admin')

view = ChangeOrderStatus.as\_view()

request = factory.put('/account/change-order-status/1/', {

"is\_delivered": "True",

"delivered\_at": timezone.now()

})

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 200)

def test\_changing\_of\_order\_status\_by\_normal\_user(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser') # normal user (not an admin)

view = ChangeOrderStatus.as\_view()

request = factory.put('/account/change-order-status/1/', {

"is\_delivered": "True",

"delivered\_at": timezone.now()

})

force\_authenticate(request, user=user)

response = view(request, 1)

self.assertEqual(response.status\_code, 403) # Forbidden

def test\_changing\_of\_order\_status\_when\_logged\_out(self):

response = self.client.put('/account/change-order-status/1/', {

"is\_delivered": "True",

"delivered\_at": timezone.now()

})

self.assertEqual(response.status\_code, 401) # Unauthorized

def test\_fetching\_of\_user\_stripe\_card\_when\_logged\_in(self):

factory = APIRequestFactory()

user = User.objects.get(username='testuser')

view = CardsListView.as\_view()

request = factory.get('/account/stripe-cards/')

force\_authenticate(request, user=user)

response = view(request)

self.assertEqual(response.status\_code, 200)

self.assertContains(response, "testuser") # confirming ===> getting only testuser card

self.assertNotContains(response, "admin") # confirming ===> not getting admin card

self.assertContains(response, "1234123412341234") # confirming ===> getting testuser card number only

self.assertNotContains(response, "4242424242424242") # confirming ===> not getting admin card number

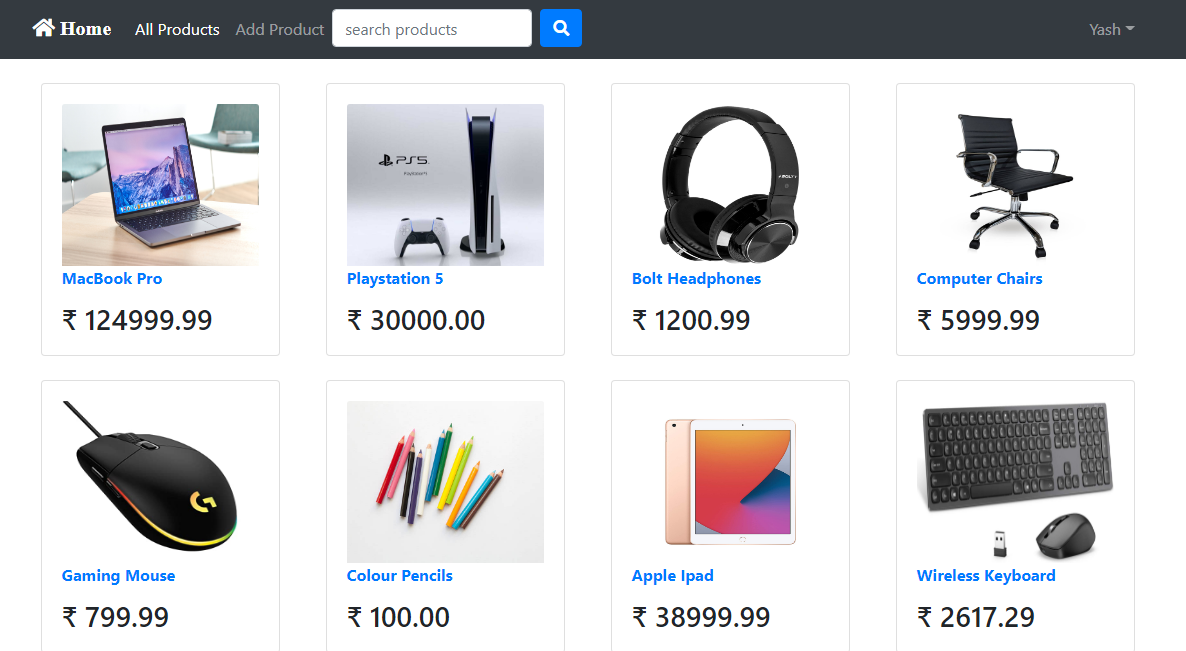
def test\_fetching\_of\_user\_stripe\_card\_when\_logged\_out(self):

response = self.client.get('/account/stripe-cards/')

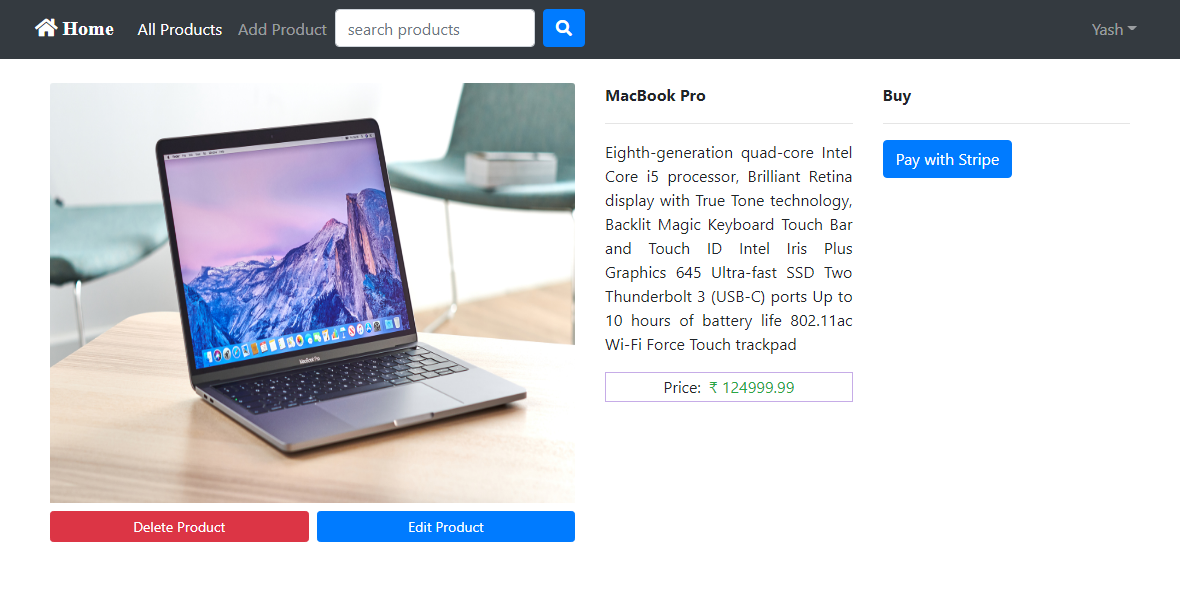
self.assertEqual(response.status\_code, 401)

**6. SCREENSHOTS**

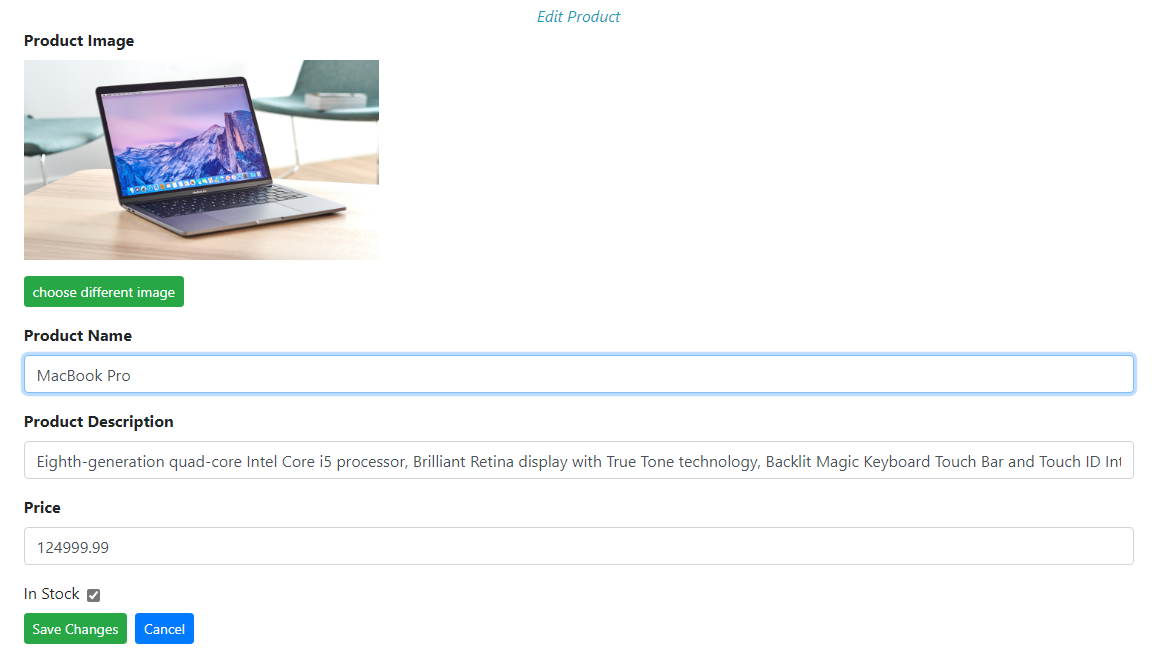
**Products\_List\_Page:-**

****

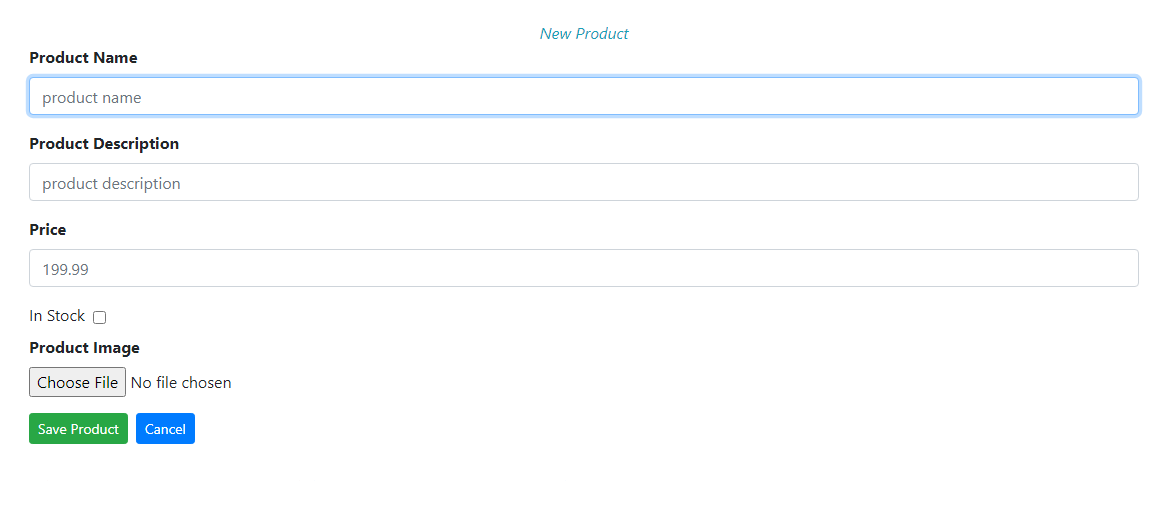
**Products\_Details\_Page:-**

****

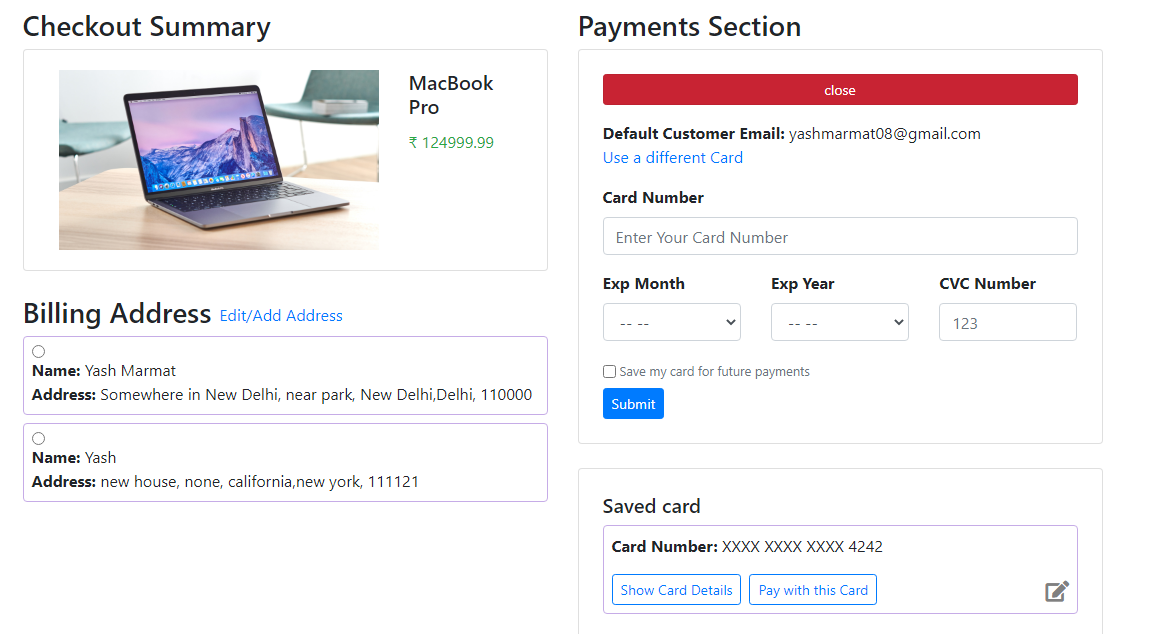
**Product\_Edit\_Page:-**

****

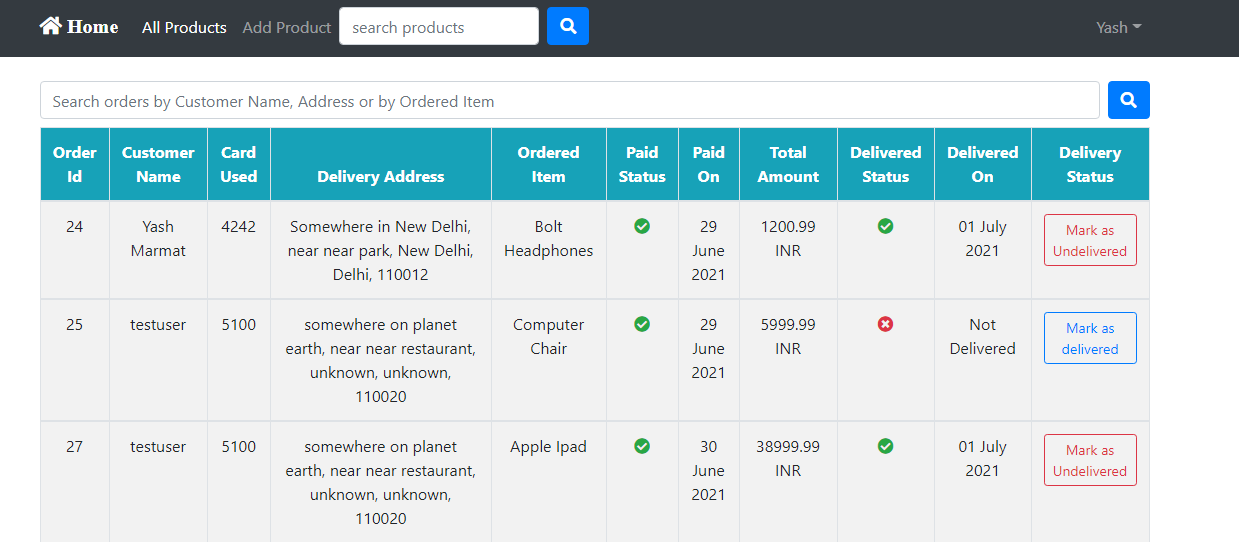
**Add\_Product\_Page:-**

****

**Checkout\_Page:-**

****

**Orders\_Page\_For\_Admin:-**

****

1. **CONCLUSION**

In conclusion, the development of the incident management web application, titled "E-Commerce Website," marks a significant step towards enhancing emergency response and management capabilities in our modern society.

By leveraging HTML, CSS, and JavaScript for the frontend interface, and Spring Framework (Spring Boot) for the backend logic, we have created a user-friendly platform that streamlines the reporting process and facilitates real-time communication among stakeholders. The integration of MySQL as the database management system ensures secure storage and retrieval of incident data, while Java serves as the backbone programming language, enabling robust and scalable application development.

Here’s a structured conclusion for an e-commerce website project:

**Achievements:-**

**1.Website Launch**: The successful launch of the e-commerce website marked a significant achievement. This included a fully functional shopping cart, secure payment integration, and a user-friendly interface, which were developed and deployed within the projected timeline.

**2.Product Range:** Successfully listing a wide range of products with detailed descriptions and high-quality images, which has proven essential in attracting and retaining customers.

**3.Marketing and Outreach:** Implementing an effective digital marketing strategy that increased brand awareness and drove traffic to the site through SEO, social media campaigns, and email marketing.

**Challenges:-**

**1.Technical Issues:** Encountering and resolving unexpected bugs and glitches, especially in mobile responsiveness and checkout processes.

**2.Supply Chain Management:** Managing inventory and logistics, particularly during peak times, posed a challenge that required dynamic strategies and solutions.

**3.Customer Service:** Establishing a reliable customer service framework that efficiently handles inquiries, returns, and complaints was more resource-intensive than initially anticipated.

**Performance Metrics:-**

**1.User Engagement**: Analysis of user engagement metrics indicated that users appreciated the site’s navigation and layout, leading to higher than average time-on-site and page views per session.

**2.Sales Growth**: The site achieved a steady month-on-month growth in sales and customer base, indicating effective market penetration and customer satisfaction.

**3.Feedback and Reviews:** Customer feedback has been overwhelmingly positive with some constructive criticism highlighting areas for improvement.

**Lessons Learned:**

**1.User Experience**: The importance of continuous testing and optimization of the user interface to ensure a seamless shopping experience.

**2.Scalability:** The need for scalable infrastructure became evident as traffic increased, teaching the importance of planning for growth in early stages.

**3.Security:** Enhancing security measures continuously to protect customer data and build trust.

**Future Directions:**

**1.Technology Upgrades**: Implementing AI and machine learning for better product recommendations, personalized marketing, and enhanced customer service.

**2.Expansion:** Expanding the product line and exploring new markets, both domestically and internationally.

**3.Sustainability**: Focusing on sustainability in packaging and shipping methods to appeal to environmentally conscious consumers.

The proposed system's key features, including unified incident reporting, real-time communication, and geospatial mapping capabilities, are designed to enhance coordination, optimize resource allocation, and ultimately, improve public safety outcomes. By providing a centralized platform for incident reporting and management, "E-Commerce Website" empowers citizens to contribute actively to emergency response efforts while enabling authorities to make data-driven decisions and respond promptly to critical situations.

In essence, "E-Commerce Website" represents not only a technological solution but also a testament to our collective dedication to serving the greater good and safeguarding the well-being of our communities. As we embark on this transformative journey, let us remain steadfast in our commitment to building a safer, more connected, and resilient future for all.

**8. BIBLIOGRAPHY**

1. ECMA International. "JavaScript (ECMAScript) Language Specification," 2021. [Online]. Available: https://www.ecma-international.org/ecma-262/.
2. MySQL. "MySQL Documentation," 2022. [Online]. Available: https://dev.mysql.com/doc/.
3. W3Schools. "HTML Geolocation." [Online]. Available: https://www.w3schools.com/html/html5\_geolocation.asp.
4. Spring Team. "Building REST services with Spring." Spring Boot Documentation. Accessed on Apr. 21, 2024. [Online]. Available: https://spring.io/guides/gs/spring-boot.
5. W3Schools. "Python Introduction." Available: https://www.w3schools.com/python/python\_intro.asp.