PHASE-5

E-Commerce Application on IBM Cloud Foundry

**Problem Statement**:

Build a artisanal e-Commerce platform using IBM foundry. Connect the skilled artisans with the global audience. Showcase handmade products from exquisite jewelry to artistic home decor. Implement secure shopping carts, smooth payment gateway and an intuitive checkout process. Nurture creativity and support small business through an artisan's dream marketplace.

**PROJECT OBJECTIVE:**

Connect Artisans and Global Audience: The main goal is to create a platform that facilitates the connection between skilled artisans and a global audience of potential customers.

1. Showcase Handmade Products: The platform should effectively showcase a wide range of handmade products, including jewelry and artistic home decor, to highlight the artisans' craftsmanship.
2. Enable Secure Transactions: Implement a secure shopping cart system, a smooth payment gateway, and an intuitive checkout process to ensure secure and hassle-free transactions for customers.
3. Support Artisan Creativity: Create a nurturing environment for artisan creativity by offering them tools and profiles to effectively display their

products and manage their orders.

1. Promote Small Businesses: Support small businesses and individual artisans by providing them with a marketplace to reach a broader customer base and grow their businesses.
2. User-Friendly Experience: Develop an intuitive and visually appealing website with user-friendly navigation, robust search and filter options, and efficient shopping features.
3. Data Security and Compliance: Ensure the security and privacy of customer data and compliance with payment regulations, including PCI DSS standards.
4. Marketing and Promotion: Attract artisans and customers to the platform through effective marketing and promotional strategies, including social media, email marketing, and SEO techniques.
5. Continuous Improvement: Gather feedback from users and artisans to continuously improve the platform's functionality and user experience.
6. Legal and Compliance: Ensure that the platform complies with all relevant e-commerce laws and regulations, including terms of service, privacy policy, and return/refund policies.
7. Sustainability: Foster an environment that promotes the sustainability of small artisan businesses, encourages creativity, and supports the growth of the artisanal community.These objectives collectively aim to address the challenges outlined in the problem statement and guide the development and operation of the artisanal e-commerce platform.

**DESIGN THINKING**:

start

User

Admin

Register

Login

nn

login

Search item

Add item category

Online pay

Payment

Wish

list

buy

stop

Review

Logout

Order placed

COD

View item

Report

Add product

Check feedback

Manage order

Manage payment

**Wishlist**

A wishlist is a feature commonly found in e-commerce applications and online shopping platforms. It allows users to create a personalized list of products they are interested in but may not be ready to purchase immediately. Here's a detailed explanation of wishlists and their significance:

**1. User interest:**

* Wishlists are a tool for increasing user engagement on an e-commerce platform. They provide users with a way to interact with the website beyond just browsing and purchasing.

**2. Product Noting:**

* Think of a wishlist as a digital bookmark for products. Users can add items they like, aspire to buy, or are considering for future purchases. This saves them from having to search for those products all over again.

**3. Comparison and Decision-Making:**

* Users often add multiple similar products to their wishlist. This feature enables them to compare items side by side, aiding in the decision-making process. So this will be helpful to the user to get better product and decide which is suitable and satisfy their needs.

**4. Push Notifications:**

* E-commerce platforms can send notifications to users when items on their wishlist go on sale, have limited availability, or are back in stock. This nudges users to complete their purchases.
* Push notifications major role to buy a product if the user left the product int the wishlist for long time then this push notification is helpful to remind them about the product. So the user can buy it.

**5. Abandonment Recovery:**

* For users who abandon their shopping carts, e-commerce platforms can send reminders about the items left behind, potentially motivating users to complete the purchase.

**6. Adding and Removing Items:**

* Users can easily add or remove items from their wishlist, giving them full control over the products they wish to track.

Wishlists are a valuable feature in e-commerce applications, offering users a convenient way to save and track products of interest. They enhance user engagement, provide a personalized shopping experience, aid in decision-making, and contribute to both customer and platform insights. Wishlists not only benefit users but also play a role in boosting sales and improving the overall user experience.

**Product review:**

Product reviews are a fundamental component of e-commerce platforms and online shopping. They are written assessments or evaluations by customers who have purchased and used a specific product. These reviews offer insights, feedback, and opinions about the product's performance, quality, and overall user experience. Here's a detailed explanation of the significance and role of product reviews in e-commerce:

**1. User-Created Content:**

* Product reviews are a form of user-generated content. They provide an authentic and unbiased perspective on a product, making them a valuable source of information for potential buyers.

**2. Transparency and Trust:**

* Reviews contribute to the transparency of the e-commerce platform. Customers can read about the experiences of others before making a purchase decision. This transparency builds trust in the platform and the products it offers. Trust reviews of the other person can make clear the doubt of the products so well.

**3. Informed Decision-Making:**

* Shoppers rely on product reviews to make informed decisions. Reviews can highlight both positive and negative aspects of a product, allowing potential buyers to assess whether it meets their needs and expectations. By the user reviews the buyer can get the idea to whether the product is good to buy or not.

**4. Quality Assurance:**

* Product reviews are a quality assurance mechanism. They serve as a check on the product's performance and accuracy of product descriptions provided by the platform.

**5. Peer Recommendations:**

* Customers tend to trust peer recommendations more than marketing messages. Positive reviews from fellow customers act as endorsements, increasing the likelihood of a purchase.

**6. SEO and Discoverability:**

* Product reviews, when properly structured, can improve the search engine optimization (SEO) of the platform. Reviews can contain keywords that potential customers may use in their searches. Keywords can help the user to get the more wished product and related items to the feed.

**7. Improving Customer Satisfaction:**

* Sellers can respond to reviews, addressing any issues or concerns raised by customers. This responsiveness can lead to improved customer satisfaction and loyalty.

**8. Star Ratings:**

****

* Alongside written reviews, star ratings are a quick way for customers to assess a product's overall quality. Higher ratings can boost a product's visibility and sales. Star rating are pictorial representation so user and the admin can get the review about the product easily.

**9. Filtered Reviews:**

- Many e-commerce platforms allow users to filter reviews based on criteria such as the most recent, most helpful, or highest-rated reviews, helping them find the information they need more easily.

**DEVELOPMENT PHASE:**

### 1)      Plan properly:

Requirements need to clear and a blueprint should be in place. Find out your niche audience and develop that website. What information goes where, what feature goes where, what content is necessary, where it should go, answer all these questions before you start typing that code out on your system.

Decide and define your milestones. What should be the project guideline, what should be the deadline, what should be the [budget](https://www.kadamtech.com/5-easy-steps-create-money-website-design/), what is the estimated traffic or number of hits that you expect, you should analyze way before commence.

### 2)      The scope of the project:

Development only should limit or should it extend beyond that? It is necessary to have an entry and exit criteria before you plunge into the process. This is also crucial that you figure out the best practices to stick to the budget.

It makes no sense splurging on expensive languages and purchase when the open source products work just fine! Create a chart such as Gantt and analyze where is the project headed and roll out control measures to ensure smoother operations.

### 3)      E-commerce Website design:

Colour, logo, design, layout, feature, colour, the prototype should be in place so that you have a fairly similar working model which actually appeals to the customers. The website should aim at establishing you as a brand name.

Therefore, it is indeed essential that you have several beta versions to try and testify how it works better in a commercial space. It is an e-commerce portal and you need customers. What better than potential customers to have a first-hand experience before you use it on a complete commercial scale?

### 4)      Development stage:

Take all individual elements that were finalized during your initial planning phase and implement it in your development stages. Shells, home page, interior pages, the structure of navigation, content structure, gateways, forms, management systems for content, and much more needs to be integrated to form that fully functional system

### 5)      Test and deploy:

Testing is one of the major processes that your website has to go through. Find those bugs and loopholes which might be serious. No glitches or errors are usually expecting if the website programmer tested thoroughly and completely. Display, external links, and various other connectivity need to take care of before it goes live.

**Know your audience**

One advantage of starting an online business in an area you’re familiar with is that you already know your audience, as you're a part of it. But even if you already know your customers, doing a bit of work to understand them better is beneficial for any ecommerce business.

**Plan a content strategy**

Building a website with real content, rather than filler, makes for a better representation of the end product earlier in the process. Working backward and integrating the visuals, text, and other elements later in the process can complicate things.

At the beginning of the development process, need to have a content strategy. This means knowing what content you need to tell your brand's story and communicate what your products do. This may include writing, photos, videos, infographics, and other media you'll need to best serve your customers.

**LANGUAGES USED:**

* HTML
* Python with Flask
* CSS
* SQLite

**LAYOUTS:**

**1. Homepage:**

- The homepage is the first page users see when they visit the e-commerce site. It typically includes a company logo, search bar, navigation menu, featured products or promotions, and maybe some personalized recommendations based on the user's browsing and purchase history.

**2. Navigation Menu:**

- The navigation menu provides links to various product categories, such as clothing, electronics, home goods, etc. These categories are organized hierarchically to help users find the products they are looking for**.**

**3. Product Listings:**

- Product listing pages display a grid of products, often with thumbnail images, product names, prices, and user ratings. Users can browse through these listings, and there may be filtering and sorting options to narrow down choices.

**4. Product Detail Pages:**

- Clicking on a product in the listing takes users to the product detail page, where they can find more information about the product, including detailed descriptions, specifications, reviews, and related products.

**5. Shopping Cart:**

- The shopping cart page shows the items a user has added for purchase. It provides options for modifying quantities, removing items, and applying discount codes. Users can proceed to checkout from this page.

**6. Checkout Process:**

- The checkout process involves multiple steps, each typically represented on a separate page. These steps may include shipping and billing information, payment options, order review, and order confirmation.

**7. User Account:**

- Users can create accounts or log in to existing ones. The user account section may include order history, saved addresses, payment methods, and wish lists.

**8. Search Functionality:**

- A prominent search bar allows users to quickly find specific products by entering keywords or product names.

**9. Filters and Sorting:**

- Product listing pages often include filter options (e.g., by price, brand, size) and sorting options (e.g., by price, popularity) to help users refine their search results.

**10. Responsive Design:**

- E-commerce layouts need to be responsive to work well on various devices, such as desktops, tablets, and smartphones. This ensures that the user experience remains consistent and user-friendly across different screen sizes.

**11. Visual Elements:**

- Effective e-commerce layouts use high-quality images, graphics, and icons to make the application visually appealing. The use of colors, typography, and branding elements is also important for consistency and branding.

**FEATURES:**

1. Product Catalog Management:

* The ability to add, edit, and organize products effectively. This includes features for product descriptions, images, categories, and variants (e.g., size and color options).

2. Online Payment Processing:

* Integration with various payment gateways (e.g., PayPal, Stripe, credit card processors) to facilitate secure and convenient online payments.

3. Shopping Cart:

* A virtual cart that allows users to add and manage items before proceeding to checkout. It should calculate totals, apply discounts, and accommodate both guest and registered users.

4. Checkout Process:

* A streamlined, multi-step checkout process that includes options for shipping and billing addresses, shipping methods, and payment methods. It should also provide order review before finalizing the purchase.

5. User Accounts:

* Registration and login functionality for users, enabling order history, saved addresses, payment details, and wish lists.

6. Security Features:

* SSL encryption to secure data transmission, compliance with payment card industry data security standards (PCI DSS), and protection against fraud.

7. Search and Navigation:

* Effective search functionality, filters, and sorting options to help users find products easily.

8.Responsive Design:

* A design that adapts to various devices, ensuring a consistent and user-friendly experience on desktops, tablets, and smartphones.

9. Reviews and Ratings:

User-generated product reviews and ratings to build trust and help customers make informed purchasing decisions.

10. Content Management:

* Features for creating, managing, and updating content, including product descriptions, blog posts, and landing pages.

11. SEO-Friendly Features:

* Tools and settings to optimize product listings and content for search engines.

**TECHNICAL IMPLEMENTATION:**

**Database:**

**STEP 1**:`import sqlite3`: This line imports the `sqlite3` module in Python, which allows you to interact with SQLite databases.

**STEP 2**: `conn = sqlite3.connect('database.db')`: This line establishes a connection to an SQLite database named 'database.db'. If the database doesn't exist, it will be created. The `conn` variable represents the database connection, which is used to execute SQL commands and interact with the database.

**STEP 3**: `conn.execute('''CREATE TABLE users ... )''`: This code creates a table named 'users' in the database. It defines the structure of the 'users' table with several columns, including:

1. `userId`: An INTEGER column designated as the primary key. It will uniquely identify each user.
2. `password`: A TEXT column for storing user passwords.
3. `email`: A TEXT column for storing user email addresses.
4. `firstName`: A TEXT column for storing the user's first name.
5. `lastName`: A TEXT column for storing the user's last name.
6. `address1` and `address2`: TEXT columns for storing user addresses.
7. `zipcode`: A TEXT column for storing user zip codes.
8. `city`, `state`, and `country`: TEXT columns for storing user city, state, and country.
9. `phone`: A TEXT column for storing user phone numbers.

**STEP 4**: `conn.execute('''CREATE TABLE products ... )''`: This code creates a table named 'products' in the database. The 'products' table is designed to store information about products, and it has the following columns:

1. `productId`: An INTEGER column designated as the primary key, uniquely identifying each product.
2. `name`: A TEXT column for storing the name of the product.
3. `price`: A REAL column for storing the price of the product.
4. `description`: A TEXT column for storing a description of the product.
5. `image`: A TEXT column for storing a reference to an image associated with the product.
6. `stock`: An INTEGER column for storing the quantity of the product in stock.
7. `categoryId`: An INTEGER column that serves as a foreign key, referencing the 'categories' table.

**STEP 5**: `conn.execute('''CREATE TABLE kart ... )''`: This code creates a table named 'kart', which seems to represent a user's shopping cart or a list of products. It has two columns:

1. `userId`: An INTEGER column representing the user associated with the item in the cart. It is a foreign key that references the 'users' table.
2. `productId`: An INTEGER column representing the product in the cart. It is a foreign key that references the 'products' table.

**STEP 6**: `conn.execute('''CREATE TABLE categories ... )''`: This code creates a table named 'categories' for categorizing products. It has the following columns:

1. `categoryId`: An INTEGER column designated as the primary key, uniquely identifying each category.
2. `name`: A TEXT column for storing the name of the category.

**STEP 7**: `conn.close()`: Finally, this line closes the database connection when you're done with it. It's essential to close the connection to release the associated resources and ensure data integrity.

After running this code, you'll have a SQLite database ('database.db') with four tables ('users', 'products', 'kart', and 'categories') created and ready to store data according to the defined schema. You can perform various database operations such as inserting, updating, deleting, and querying data in these tables as needed for your application.

**Queries used in database:**

import sqlite3

#Open database

conn = sqlite3.connect('database.db')

#Create table

conn.execute('''CREATE TABLE users

(userId INTEGER PRIMARY KEY,

password TEXT,

email TEXT,

firstName TEXT,

lastName TEXT,

address1 TEXT,

address2 TEXT,

zipcode TEXT,

city TEXT,

state TEXT,

country TEXT,

phone TEXT

)''')

conn.execute('''CREATE TABLE products

(productId INTEGER PRIMARY KEY,

name TEXT,

price REAL,

description TEXT,

image TEXT,

stock INTEGER,

categoryId INTEGER,

FOREIGN KEY(categoryId) REFERENCES categories(categoryId)

)''')

conn.execute('''CREATE TABLE kart

(userId INTEGER,

productId INTEGER,

FOREIGN KEY(userId) REFERENCES users(userId),

FOREIGN KEY(productId) REFERENCES products(productId)

)''')

conn.execute('''CREATE TABLE categories

(categoryId INTEGER PRIMARY KEY,

name TEXT

)''')

conn.close()

**UPDATION OF DATABASE:**

UPDATE products set price = '999' WHERE productId = 2;

UPDATE products set price = '399' WHERE productId = 3;

UPDATE products set price = '4499' WHERE productId = 4;

UPDATE products set price = '99' WHERE productId = 5;

UPDATE products set price = '9999' WHERE productId = 6;

UPDATE products set price = '599' WHERE productId = 7;

UPDATE products set price = '899' WHERE productId = 8;

UPDATE products set price = '599' WHERE productId = 9;

UPDATE products set price = '499' WHERE productId = 10;

UPDATE products set price = '299' WHERE productId = 11;

SELECT \* FROM products;

UPDATE products set price = '899' WHERE productId = 16;

UPDATE products set name = 'Jewellery' WHERE productId = 16;

UPDATE products set description = 'Fit to Any Dress' WHERE productId = 16;

UPDATE products set image = 'jewel\_2.jpeg' WHERE productId = 16;

UPDATE products set stock = '200' WHERE productId = 16;

UPDATE products set categoryId = 'J29' WHERE productId = 16;

UPDATE categories set name = 'Wearings' WHERE categoryId = 1;

UPDATE categories set name = 'Hangings' WHERE categoryId = 2;

UPDATE categories set name = 'Handmade' WHERE categoryId = 3;

SELECT \*FROM categories;

DELETE FROM categories WHERE categoryId = 4 ;

DELETE FROM categories WHERE categoryId = 5 ;

DELETE FROM categories WHERE categoryId = 6 ;

SELECT \* FROM categories;

UPDATE products set price = '899' WHERE productId = 17;

UPDATE products set name = 'Saree' WHERE productId = 17;

UPDATE products set description = 'Gorgious Looking' WHERE productId = 17;

UPDATE products set image = 'saree.png' WHERE productId = 17;

UPDATE products set stock = '200' WHERE productId = 17;

UPDATE products set categoryId = 'RI23' WHERE productId = 17;

SELECT \* FROM products;

UPDATE products set image = 'jewel\_3.png' WHERE productId = 11;

UPDATE products set image = 'jewel\_2.png' WHERE productId = 16;

UPDATE products set image = 'jewellery.png' WHERE productId = 8;

UPDATE products set price = '499' WHERE productId = 12;

UPDATE products set name = 'Sandal' WHERE productId = 12;

UPDATE products set description = 'Good Quality, Good Comfort' WHERE productId = 12;

UPDATE products set image = 'sandal1.png' WHERE productId = 12;

UPDATE products set stock = '200' WHERE productId = 12;

UPDATE products set categoryId = 'S10' WHERE productId = 12;

SELECT \* FROM products;

UPDATE products set price = '499' WHERE productId = 14;

UPDATE products set name = 'Sandal' WHERE productId = 14;

UPDATE products set description = 'Good Quality, Good Comfort' WHERE productId = 14;

UPDATE products set image = 'sandal2.png' WHERE productId = 14;

UPDATE products set stock = '200' WHERE productId = 14;

UPDATE products set categoryId = 'S10' WHERE productId = 14;

UPDATE products set price = '7999' WHERE productId = 13;

UPDATE products set name = 'Wall Painting' WHERE productId = 13;

UPDATE products set description = 'Amazing and Give a Fabulous Look' WHERE productId = 13;

UPDATE products set image = 'paint1.png' WHERE productId = 13;

UPDATE products set stock = '100' WHERE productId = 13;

UPDATE products set categoryId = 'WP23' WHERE productId = 13;

UPDATE products set price = '7599' WHERE productId = 15;

UPDATE products set name = 'Wall Painting' WHERE productId = 15;

UPDATE products set description = 'Amazing and Give a Fabulous Look' WHERE productId = 15;

UPDATE products set image = 'paint2.png' WHERE productId = 15;

UPDATE products set stock = '100' WHERE productId = 15;

UPDATE products set categoryId = 'WP23' WHERE productId = 15;

**LAYOUTS:**

**HOME.HTML**

<!DOCTYPE HTML>

<html>

<head>

<title>Welcome</title>

<link rel="stylesheet" href={{ url\_for('static', filename='css/home.css') }} />

<link rel="stylesheet" href={{ url\_for('static', filename='css/topStyle.css') }} />

</head>

<body>

<div id="title">

<a href="/">

<img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />

</a>

<form>

<input id="searchBox" type="text" name="searchQuery">

<input id="searchButton" type="submit" value="Search">

</form>

{% if not loggedIn %}

<div id="signInButton">

<a class="link" href="/loginForm">Sign In</a>

</div>

{% else %}

<div class="dropdown">

<button class="dropbtn">Hello, <br>{{firstName}}</button>

<div class="dropdown-content">

<a href="/account/orders">Your orders</a>

<a href="/account/profile">Your profile</a>

<hr>

<a href="/logout">Sign Out</a>

</div>

</div>

{% endif %}

<div id="kart">

<a class="link" href="/cart">

<img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />

CART {{noOfItems}}

</a>

</div>

</div>

<div class="display">

<div class="displayCategory">

<h2>Shop by Category: </h2>

<ul>

{% for row in categoryData %}

<li><a href="/displayCategory?categoryId={{row[0]}}">{{row[1]}}</a></li>

{% endfor %}

</ul>

</div>

<div>

<h2>Items</h2>

{% for data in itemData %}

<table>

<tr id="productName">

{% for row in data %}

<td>

{{row[1]}}

</td>

{% endfor %}

</tr>

<tr id="productImage">

{% for row in data %}

<td>

<a href="/productDescription?productId={{row[0]}}">

<img src={{ url\_for('static', filename='uploads/' + row[4]) }} id="itemImage" />

</a>

</td>

{% endfor %}

</tr>

<tr id="productPrice">

{% for row in data %}

<td>

${{row[2]}}

</td>

{% endfor %}

</tr>

</table>

{% endfor %}

</div>

</div>

</body>

</html>

This code was created using html to get the front end of the application.

**CSSfiles**

**CART.CSS**

#tableItems {

margin-left: 20px;

margin-right: 20px;

margin-top: 20px;

margin-bottom: 20px;

}

#itemImage {

margin-left: 5px;

margin-right: 5px;

margin-top: 5px;

margin-bottom: 5px;

height: 100px;

float: left;

}

#itemName {

margin-left: 5px;

margin-right: 5px;

margin-top: 5px;

margin-bottom: 5px;

height: 100px;

width: 200px;

float: left;

}

#titleName {

width: 200px;

float: left;

}

#titlePrice {

float: left;

}

#itemPrice {

margin-left: 5px;

margin-right: 5px;

margin-top: 5px;

margin-bottom: 5px;

height: 100px;

display: inline-block;

}

#image {

height: 100px;

width: 80px;

}

#seperator {

margin: 0px;

max-width: 400px;

}

#total {

padding-left: 280px;

}

#itemNameTag {

font-weight: bold;

}

#subtotal {

font-weight: bold;

font-size: 20px;

}

These codes are used to design the cart page.

**HOME.css**

#itemImage {

height: 200px;

width: 150px;

}

.display {

margin-top: 20px;

margin-left: 20px;

margin-right: 20px;

margin-bottom: 20px;

}

table {

border-spacing: 20px;

}

#productName {

text-align: center;

font-weight: bold;

}

#productPrice {

text-align: center;

}

.displayCategory ul li {

font-size: 20px;

}

The above codes are used to design the home page of the e-commerce.

**PRODUCTDESCRIPTION.css**

#display {

margin-top: 20px;

margin-left: 20px;

margin-right: 20px;

margin-bottom: 20px;

}

#productImage {

height: 250px;

width: 200px;

margin-left: 20px;

margin-right: 20px;

margin-top: 20px;

margin-bottom: 20px;

display: inline-block;

float: left;

}

#productDescription {

margin-left: 20px;

margin-right: 20px;

margin-top: 20px;

margin-bottom: 20px;

display: inline-block;

font-size: 19px;

}

#descriptionTable td {

width: 150px;

}

#addToCart {

font-size: 20px;

}

**PYTHON with Flask:**

from flask import \*

import sqlite3, hashlib, os

from werkzeug.utils import secure\_filename

app = Flask(\_\_name\_\_)

app.secret\_key = 'random string'

UPLOAD\_FOLDER = 'static/uploads'

ALLOWED\_EXTENSIONS = set(['jpeg', 'jpg', 'png', 'gif'])

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

def getLoginDetails():

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

if 'email' not in session:

loggedIn = False

firstName = ''

noOfItems = 0

else:

loggedIn = True

cur.execute("SELECT userId, firstName FROM users WHERE email = ?", (session['email'], ))

userId, firstName = cur.fetchone()

cur.execute("SELECT count(productId) FROM kart WHERE userId = ?", (userId, ))

noOfItems = cur.fetchone()[0]

conn.close()

return (loggedIn, firstName, noOfItems)

@app.route("/")

def root():

loggedIn, firstName, noOfItems = getLoginDetails()

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute('SELECT productId, name, price, description, image, stock FROM products')

itemData = cur.fetchall()

cur.execute('SELECT categoryId, name FROM categories')

categoryData = cur.fetchall()

itemData = parse(itemData)

return render\_template('home.html', itemData=itemData, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems, categoryData=categoryData)

@app.route("/add")

def admin():

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT categoryId, name FROM categories")

categories = cur.fetchall()

conn.close()

return render\_template('add.html', categories=categories)

@app.route("/addItem", methods=["GET", "POST"])

def addItem():

if request.method == "POST":

name = request.form['name']

price = float(request.form['price'])

description = request.form['description']

stock = int(request.form['stock'])

categoryId = int(request.form['category'])

#Uploading image procedure

image = request.files['image']

if image and allowed\_file(image.filename):

filename = secure\_filename(image.filename)

image.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

imagename = filename

with sqlite3.connect('database.db') as conn:

try:

cur = conn.cursor()

cur.execute('''INSERT INTO products (name, price, description, image, stock, categoryId) VALUES (?, ?, ?, ?, ?, ?)''', (name, price, description, imagename, stock, categoryId))

conn.commit()

msg="added successfully"

except:

msg="error occured"

conn.rollback()

conn.close()

print(msg)

return redirect(url\_for('root'))

@app.route("/remove")

def remove():

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute('SELECT productId, name, price, description, image, stock FROM products')

data = cur.fetchall()

conn.close()

return render\_template('remove.html', data=data)

@app.route("/removeItem")

def removeItem():

productId = request.args.get('productId')

with sqlite3.connect('database.db') as conn:

try:

cur = conn.cursor()

cur.execute('DELETE FROM products WHERE productID = ?', (productId, ))

conn.commit()

msg = "Deleted successsfully"

except:

conn.rollback()

msg = "Error occured"

conn.close()

print(msg)

return redirect(url\_for('root'))

@app.route("/displayCategory")

def displayCategory():

loggedIn, firstName, noOfItems = getLoginDetails()

categoryId = request.args.get("categoryId")

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT products.productId, products.name, products.price, products.image, categories.name FROM products, categories WHERE products.categoryId = categories.categoryId AND categories.categoryId = ?", (categoryId, ))

data = cur.fetchall()

conn.close()

categoryName = data[0][4]

data = parse(data)

return render\_template('displayCategory.html', data=data, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems, categoryName=categoryName)

@app.route("/account/profile")

def profileHome():

if 'email' not in session:

return redirect(url\_for('root'))

loggedIn, firstName, noOfItems = getLoginDetails()

return render\_template("profileHome.html", loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)

@app.route("/account/profile/edit")

def editProfile():

if 'email' not in session:

return redirect(url\_for('root'))

loggedIn, firstName, noOfItems = getLoginDetails()

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT userId, email, firstName, lastName, address1, address2, zipcode, city, state, country, phone FROM users WHERE email = ?", (session['email'], ))

profileData = cur.fetchone()

conn.close()

return render\_template("editProfile.html", profileData=profileData, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)

@app.route("/account/profile/changePassword", methods=["GET", "POST"])

def changePassword():

if 'email' not in session:

return redirect(url\_for('loginForm'))

if request.method == "POST":

oldPassword = request.form['oldpassword']

oldPassword = hashlib.md5(oldPassword.encode()).hexdigest()

newPassword = request.form['newpassword']

newPassword = hashlib.md5(newPassword.encode()).hexdigest()

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT userId, password FROM users WHERE email = ?", (session['email'], ))

userId, password = cur.fetchone()

if (password == oldPassword):

try:

cur.execute("UPDATE users SET password = ? WHERE userId = ?", (newPassword, userId))

conn.commit()

msg="Changed successfully"

except:

conn.rollback()

msg = "Failed"

return render\_template("changePassword.html", msg=msg)

else:

msg = "Wrong password"

conn.close()

return render\_template("changePassword.html", msg=msg)

else:

return render\_template("changePassword.html")

@app.route("/updateProfile", methods=["GET", "POST"])

def updateProfile():

if request.method == 'POST':

email = request.form['email']

firstName = request.form['firstName']

lastName = request.form['lastName']

address1 = request.form['address1']

address2 = request.form['address2']

zipcode = request.form['zipcode']

city = request.form['city']

state = request.form['state']

country = request.form['country']

phone = request.form['phone']

with sqlite3.connect('database.db') as con:

try:

cur = con.cursor()

cur.execute('UPDATE users SET firstName = ?, lastName = ?, address1 = ?, address2 = ?, zipcode = ?, city = ?, state = ?, country = ?, phone = ? WHERE email = ?', (firstName, lastName, address1, address2, zipcode, city, state, country, phone, email))

con.commit()

msg = "Saved Successfully"

except:

con.rollback()

msg = "Error occured"

con.close()

return redirect(url\_for('editProfile'))

@app.route("/loginForm")

def loginForm():

if 'email' in session:

return redirect(url\_for('root'))

else:

return render\_template('login.html', error='')

@app.route("/login", methods = ['POST', 'GET'])

def login():

if request.method == 'POST':

email = request.form['email']

password = request.form['password']

if is\_valid(email, password):

session['email'] = email

return redirect(url\_for('root'))

else:

error = 'Invalid UserId / Password'

return render\_template('login.html', error=error)

@app.route("/productDescription")

def productDescription():

loggedIn, firstName, noOfItems = getLoginDetails()

productId = request.args.get('productId')

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute('SELECT productId, name, price, description, image, stock FROM products WHERE productId = ?', (productId, ))

productData = cur.fetchone()

conn.close()

return render\_template("productDescription.html", data=productData, loggedIn = loggedIn, firstName = firstName, noOfItems = noOfItems)

@app.route("/addToCart")

def addToCart():

if 'email' not in session:

return redirect(url\_for('loginForm'))

else:

productId = int(request.args.get('productId'))

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT userId FROM users WHERE email = ?", (session['email'], ))

userId = cur.fetchone()[0]

try:

cur.execute("INSERT INTO kart (userId, productId) VALUES (?, ?)", (userId, productId))

conn.commit()

msg = "Added successfully"

except:

conn.rollback()

msg = "Error occured"

conn.close()

return redirect(url\_for('root'))

@app.route("/cart")

def cart():

if 'email' not in session:

return redirect(url\_for('loginForm'))

loggedIn, firstName, noOfItems = getLoginDetails()

email = session['email']

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT userId FROM users WHERE email = ?", (email, ))

userId = cur.fetchone()[0]

cur.execute("SELECT products.productId, products.name, products.price, products.image FROM products, kart WHERE products.productId = kart.productId AND kart.userId = ?", (userId, ))

products = cur.fetchall()

totalPrice = 0

for row in products:

totalPrice += row[2]

return render\_template("cart.html", products = products, totalPrice=totalPrice, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)

@app.route("/removeFromCart")

def removeFromCart():

if 'email' not in session:

return redirect(url\_for('loginForm'))

email = session['email']

productId = int(request.args.get('productId'))

with sqlite3.connect('database.db') as conn:

cur = conn.cursor()

cur.execute("SELECT userId FROM users WHERE email = ?", (email, ))

userId = cur.fetchone()[0]

try:

cur.execute("DELETE FROM kart WHERE userId = ? AND productId = ?", (userId, productId))

conn.commit()

msg = "removed successfully"

except:

conn.rollback()

msg = "error occured"

conn.close()

return redirect(url\_for('root'))

@app.route("/logout")

def logout():

session.pop('email', None)

return redirect(url\_for('root'))

def is\_valid(email, password):

con = sqlite3.connect('database.db')

cur = con.cursor()

cur.execute('SELECT email, password FROM users')

data = cur.fetchall()

for row in data:

if row[0] == email and row[1] == hashlib.md5(password.encode()).hexdigest():

return True

return False

@app.route("/register", methods = ['GET', 'POST'])

def register():

if request.method == 'POST':

#Parse form data

password = request.form['password']

email = request.form['email']

firstName = request.form['firstName']

lastName = request.form['lastName']

address1 = request.form['address1']

address2 = request.form['address2']

zipcode = request.form['zipcode']

city = request.form['city']

state = request.form['state']

country = request.form['country']

phone = request.form['phone']

with sqlite3.connect('database.db') as con:

try:

cur = con.cursor()

cur.execute('INSERT INTO users (password, email, firstName, lastName, address1, address2, zipcode, city, state, country, phone) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)', (hashlib.md5(password.encode()).hexdigest(), email, firstName, lastName, address1, address2, zipcode, city, state, country, phone))

con.commit()

msg = "Registered Successfully"

except:

con.rollback()

msg = "Error occured"

con.close()

return render\_template("login.html", error=msg)

@app.route("/registerationForm")

def registrationForm():

return render\_template("register.html")

def allowed\_file(filename):

return '.' in filename and \

filename.rsplit('.', 1)[1] in ALLOWED\_EXTENSIONS

def parse(data):

ans = []

i = 0

while i < len(data):

curr = []

for j in range(7):

if i >= len(data):

break

curr.append(data[i])

i += 1

ans.append(curr)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Register.html**

<html>

<head>

<title>Registration</title>

<script type="text/javascript" src="{{ url\_for('static', filename = 'js/validateForm.js') }}">

</script>

</head>

<body>

<form action="/register" method="POST" onsubmit="return validate()">

<p>Email: <input type="email" name="email"></p>

<P>Password: <input type="password" name="password" id="password" required></p>

<p>Confirm Password: <input type="password" name="cpassword" id="cpassword"></p>

<p>First Name: <input type="text" name="firstName"></p>

<p>Last Name: <input type="text" name="lastName"></p>

<p>Address Line 1: <input type="text" name="address1"></p>

<p>Address Line 2: <input type="text" name="address2"></p>

<p>Zipcode: <input type="text" name="zipcode"></p>

<p>City: <input type="text" name="city"></p>

<p>State: <input type="text" name="state"></p>

<p>Country: <input type="text" name="country"></p>

<p>Phone Number: <input type="text" name="phone"></p>

<p><input type="submit" value="Register"></p>

<a href="/loginForm">Login here</a>

</form>

</body></html>

**LOGIN.html**

<html>

<head>

<title> First flask app </title>

</head>

<body>

<p> {{error}} </p>

<form action="/login" method="POST">

<p>Email: <input type="text" name="email"></p>

<p>Password: <input type="password" name="password"></p>

<p><input type="submit"></p>

<a href="/registerationForm">Register here</a>

</form>

</body>

</html>

**Productdescription.html**

<!DOCTYPE HTML>

<html>

<head>

<title>Product Description</title>

<link rel="stylesheet" href={{url\_for('static', filename='css/productDescription.css')}} />

<link rel="stylesheet" href={{ url\_for('static', filename='css/topStyle.css')}} />

</head>

<body>

<div id="title">

<a href="/">

<img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />

</a>

<form>

<input id="searchBox" type="text" name="searchQuery">

<input id="searchButton" type="submit" value="Search">

</form>

{% if not loggedIn %}

<div id="signInButton">

<a class="link" href="/loginForm">Sign In</a>

</div>

{% else %}

<div class="dropdown">

<button class="dropbtn">Hello, <br>{{firstName}}</button>

<div class="dropdown-content">

<a href="/account/orders">Your orders</a>

<a href="/account/profile">Your profile</a>

<hr>

<a href="/logout">Sign Out</a>

</div>

</div>

{% endif %}

<div id="kart">

<a class="link" href="/cart">

<img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />

CART {{noOfItems}}

</a>

</div>

</div>

<div id="display">

<div id="productName">

<h1>{{data[1]}}</h1>

</div>

<div>

<img src={{url\_for('static', filename='uploads/'+data[4]) }} id="productImage"/>

</div>

<div id="productDescription">

<h2>Details</h2>

<table id="descriptionTable">

<tr>

<td>Name</td>

<td>{{data[1]}}</td>

</tr>

<tr>

<td>Price</td>

<td>${{data[2]}}</td>

</tr>

<tr>

<td>Stock</td>

<td>{{data[5]}}</td>

</tr>

</table>

<h2>Description</h2>

<p>{{data[3]}}</p>

</div>

<div id="addToCart">

<a href="/addToCart?productId={{request.args.get('productId')}}">Add to Cart</a>

</div>

</div>

</body>

</html>

**Changepassword.html**

<html>

<head>

<title>Change Password</title>

<script src={{ url\_for('static', filename='js/changePassword.js') }}></script>

</head>

<body>

<h1>Change password</h1>

<p>{{ msg }}</p>

<form action={{ url\_for('changePassword') }} method="POST" onsubmit="return validate()">

<p>Old Password: <input type="password" name="oldpassword"></p>

<p>New Password: <input type="password" name="newpassword" id="newpassword"></p>

<p>Confirm Password: <input type="password" name="cpassword" id="cpassword"></p>

<input type="submit" value="Save">

</form>

<a href="{{ url\_for('profileHome') }}">Go to Profile</a>

</body>

</html>

**Displaycategory.html**

<!DOCTYPE HTML>

<html>

<head>

<title>Category: {{categoryName}}</title>

<link rel="stylesheet" href={{ url\_for('static', filename='css/home.css') }} />

<link rel="stylesheet" href={{ url\_for('static', filename='css/topStyle.css') }} />

</head>

<body>

<div id="title">

<a href="/">

<img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />

</a>

<form>

<input id="searchBox" type="text" name="searchQuery">

<input id="searchButton" type="submit" value="Search">

</form>

{% if not loggedIn %}

<div id="signInButton">

<a class="link" href="/loginForm">Sign In</a>

</div>

{% else %}

<div class="dropdown">

<button class="dropbtn">Hello, <br>{{firstName}}</button>

<div class="dropdown-content">

<a href="/account/orders">Your orders</a>

<a href="/account/profile">Your profile</a>

<hr>

<a href="/logout">Sign Out</a>

</div>

</div>

{% endif %}

<div id="kart">

<a class="link" href="/cart">

<img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />

CART {{noOfItems}}

</a>

</div>

</div>

<div>

<h2>Showing all products of Category {{categoryName}}:</h2>

{% for itemData in data %}

<table>

<tr id="productName">

{% for row in itemData %}

<td>

{{row[1]}}

</td>

{% endfor %}

</tr>

<tr id="productImage">

{% for row in itemData %}

<td>

<a href="/productDescription?productId={{row[0]}}">

<img src={{ url\_for('static', filename='uploads/' + row[3]) }} id="itemImage" />

</a>

</td>

{% endfor %}

</tr>

<tr id="productPrice">

{% for row in itemData %}

<td>

${{row[2]}}

</td>

{% endfor %}

</tr>

</table>

{% endfor %}

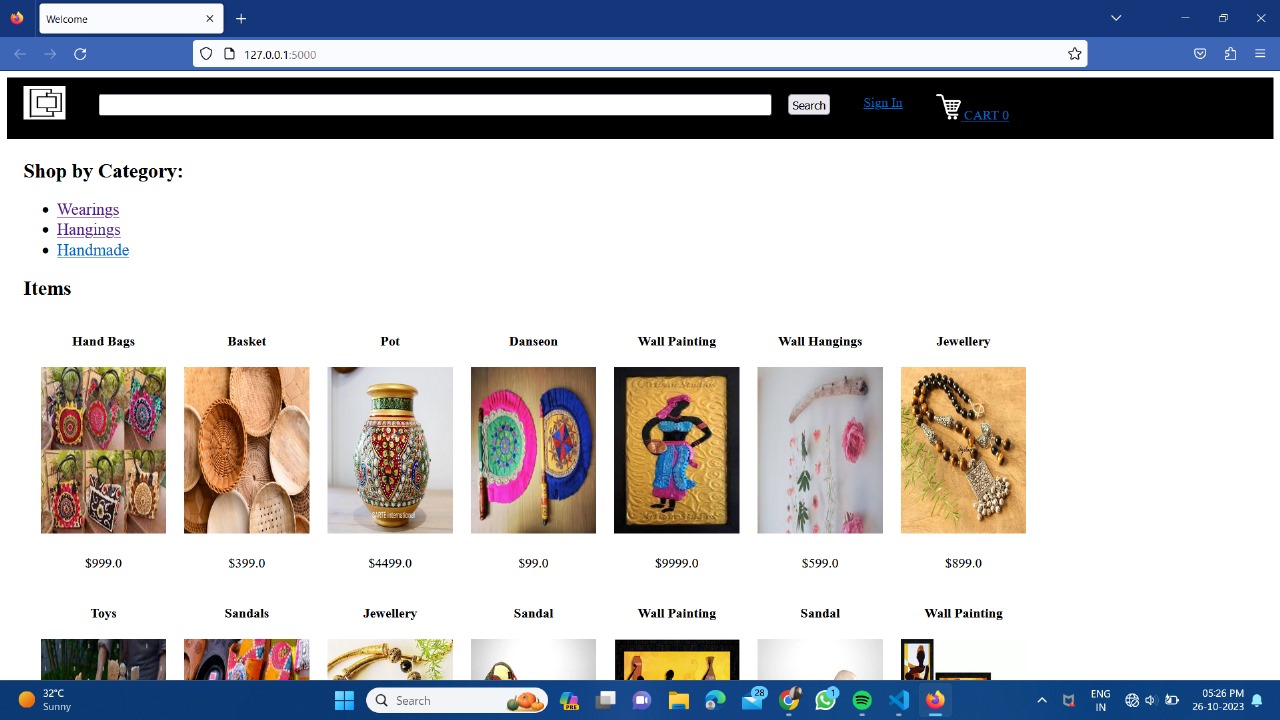
</div>

</body>

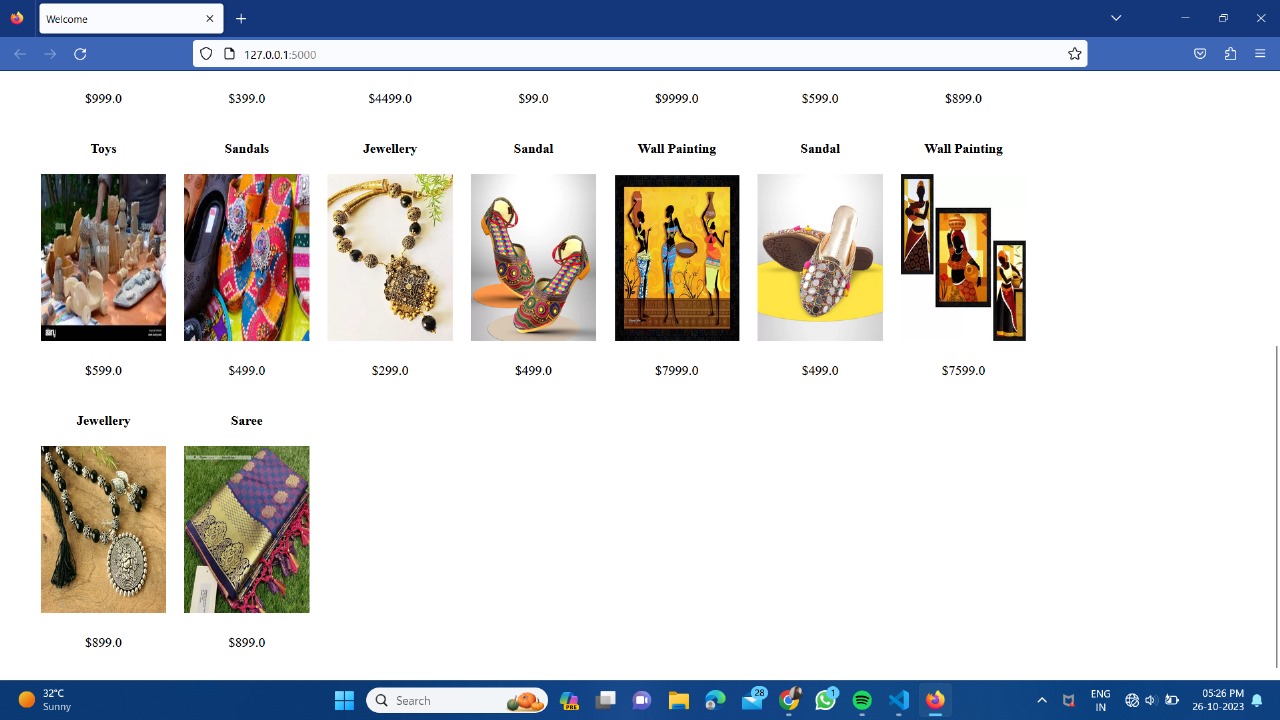
</html>

**Screenshot:**

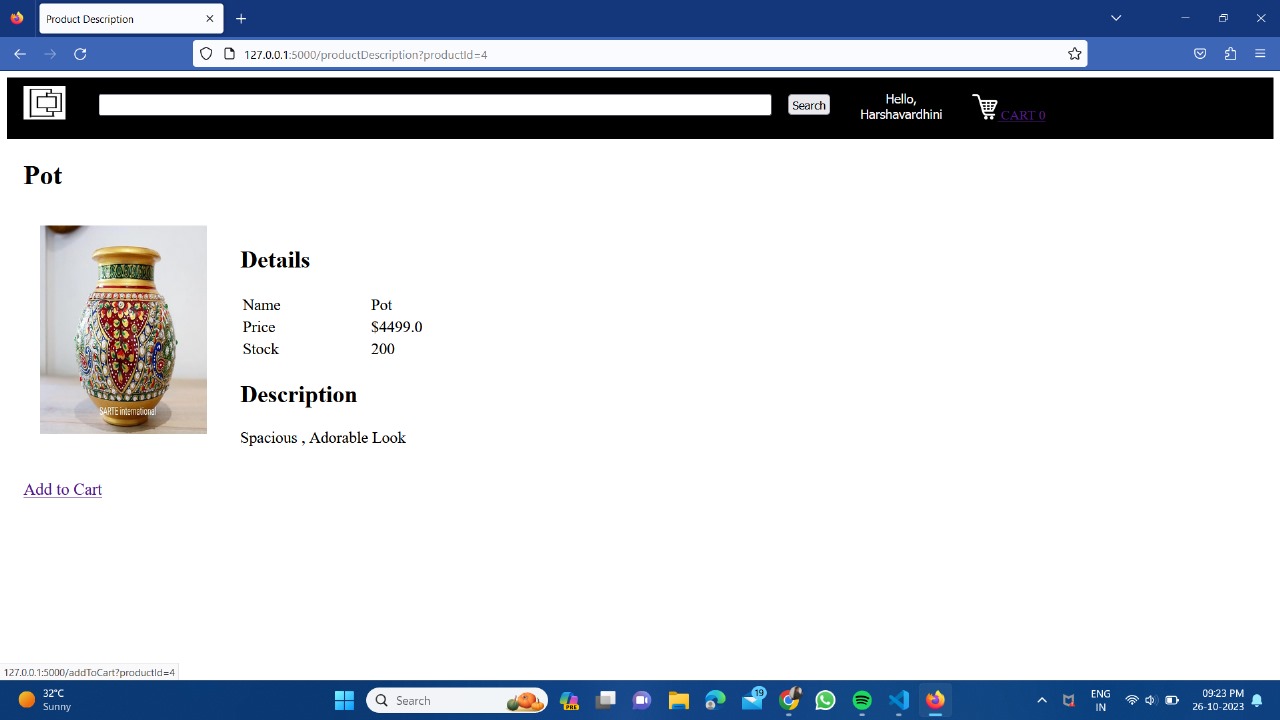
This is the home page of e-commerce application

****

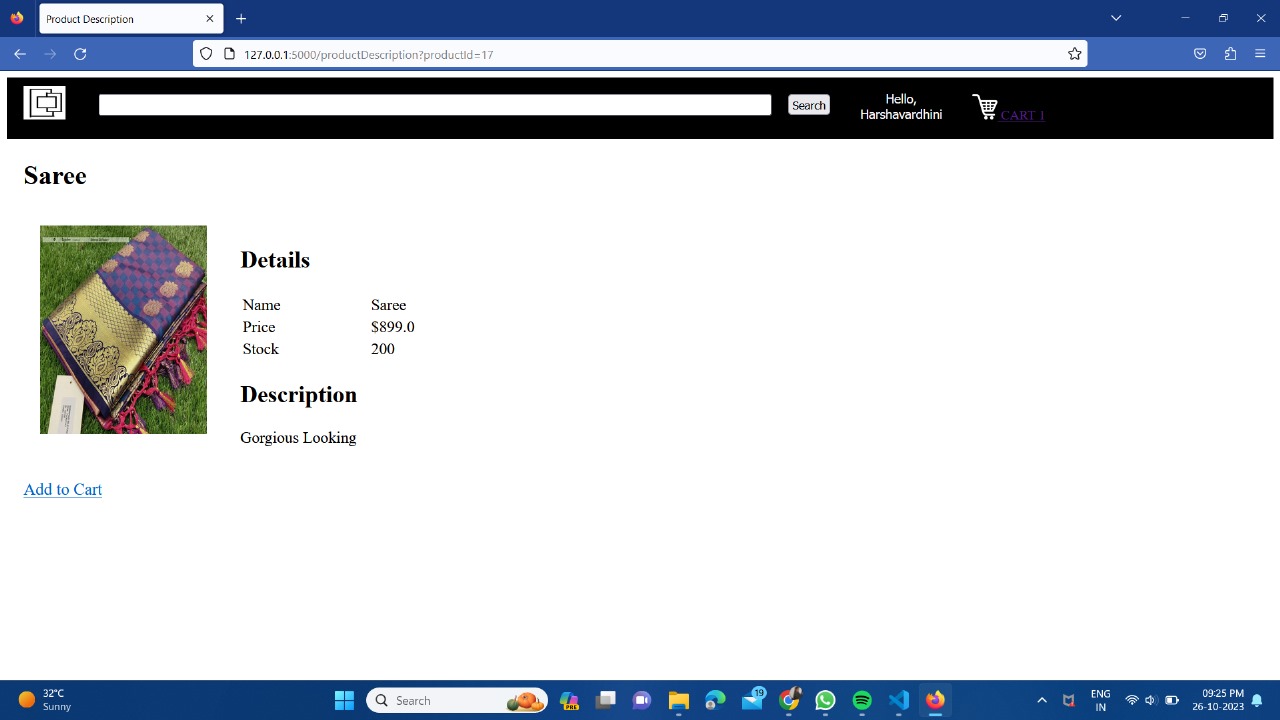
We have implemented the search bar for the easy use . In the search bar the user can search the products and they can also scroll the pages to find the right product.



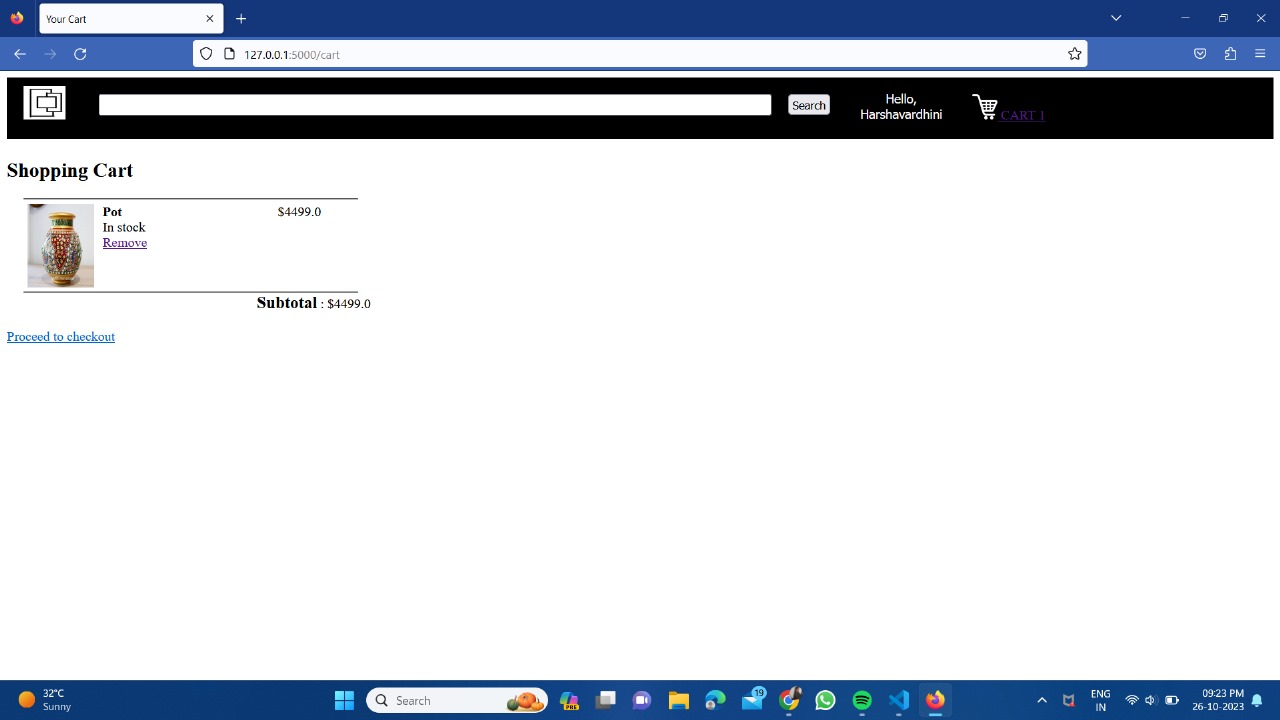
Then the user can select the required product and view the product.Product description and price also gets displayed.



This particular product has its own price,details and description and also an option to add to the cart.

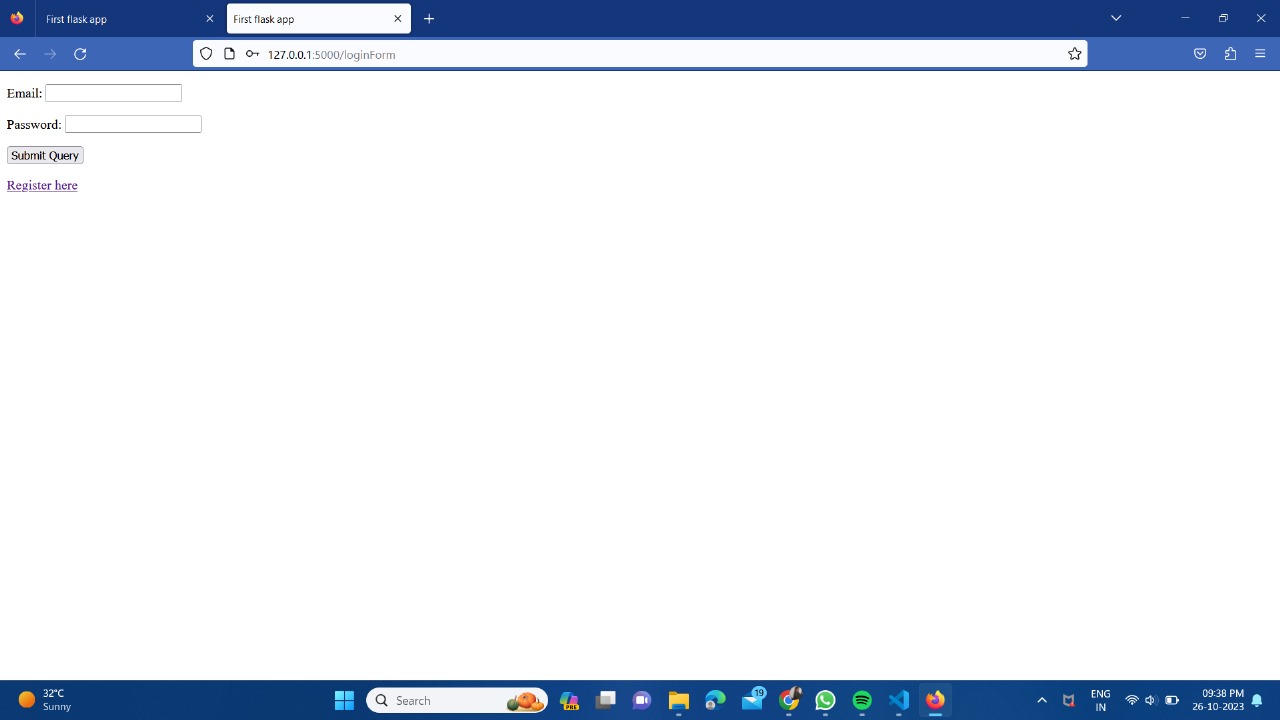


After viewing the product then, the user can add the product to cart

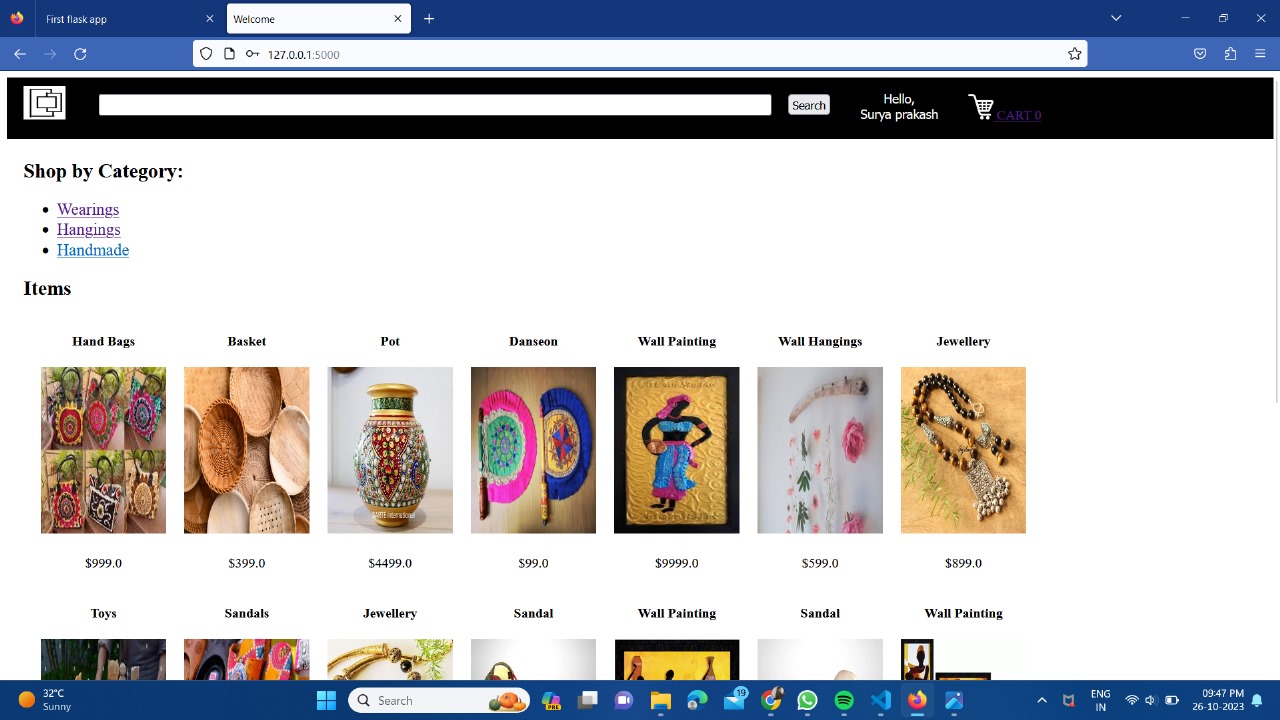


The above one is the screenshot of the product has been added to the cart.It is useful for the customer to finalise the product to buy according to the their likings.

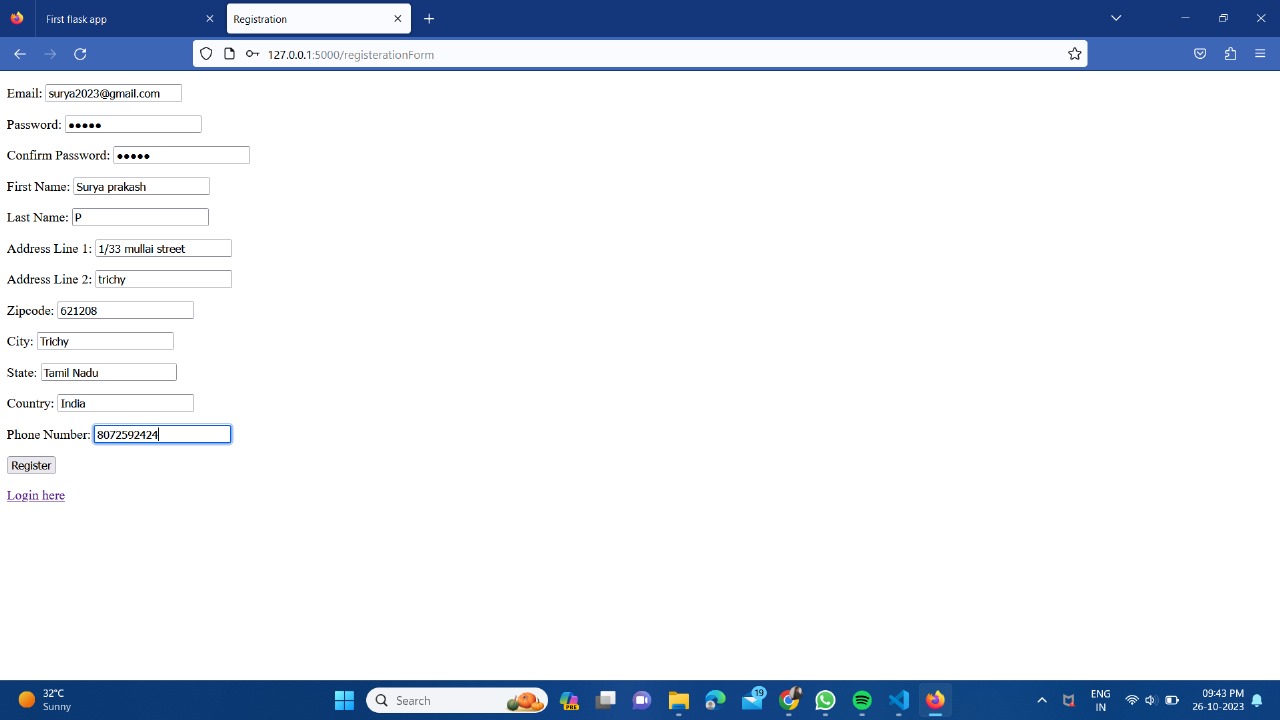
**REGISTER AND LOGIN PAGES:**

If the user already have an account then, the user can login using credentials. 

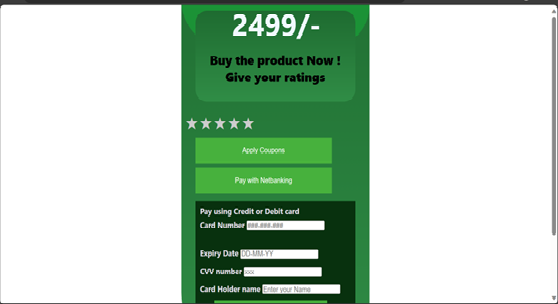
If the user already have an account then the user will login to his account, that will be displayed as “**HELLO username”.**



If the user doesn’t have an account, then the user can register he/she to the application. By this way he can create his own account.



For purchasing the product the user must complete the payment process.



The payment can be done through credit card or debit card and the user can also give review to the product by submitting the stars so that the other user can get the knowledge about the product.

**HOW TO DEPLOY :**

* Create the a new folder to insert the repositories
* Execute it in new terminal
* The code are present at my github repository
* First of all you need to clone the repository to visual studio code using git ,the commands are :

**git clone (add the repository link)**

* After this command executed the repository will be cloned.
* To set the path the command

**cd (folder name)**

* Then the path will be setted
* Now you need to list the files in the folder, their command

**ls**

* If files are not listed then it means you need to follow the instructions once again to check they are correct executed the previous time.
* Now we start to run the server by giving the command

**python main.py**

* Now the host will be shown
* Click on the host above of it we can see the option like open the link
* Now the option is selected and the e commerce page is opened.

**SUMMARY:**

E-commerce applications are digital platforms that facilitate online buying and selling of goods and services. These applications have evolved significantly over time, offering a range of features and capabilities. They enable businesses to display and manage their product catalogs, process payments securely, and provide customers with a convenient shopping experience.

Key features of e-commerce applications include product catalog management, inventory control, payment processing, shopping cart functionality, streamlined checkout processes, user account management, and security measures to protect customer data.

The development of e-commerce has gone through several phases, from its early beginnings in business-to-business transactions to the dot-com boom, mobile commerce, and the era of marketplaces and social commerce. Today, e-commerce is characterized by omnichannel retail strategies, personalization through AI, globalization, and an increasing emphasis on sustainability and ethical commerce.

As technology continues to advance, e-commerce applications are embracing new innovations, such as blockchain and virtual reality, to enhance security and provide immersive shopping experiences. E-commerce has become an integral part of the global economy, offering businesses a platform to reach customers worldwide and consumers a convenient way to shop for products and services.