Phase 3- E-Commerce Application on IBM Cloud Foundry

DEVELOPMENT PHASE – I

**Problem Statement**:

Build a artisanal e-Commerce platform using IBM foundry. Connect the skilled artisans with the global audience. Showcase handmade products from exquisite jewellry 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.

**Development phase – I**

The above problem statement is the one given to us. In the previous phases we had already given the problem definition, requirements,diagrammatic representation for the problem, and the ideas to implement the features like wish list, review, and personalized recommendation. In upcoming we had developed the above components in the e-commerce application.

**LANGUAGES USED:**

* HTML
* Python with Flask
* CSS
* SQLite

In this phase we have developed the following items:

* Database
* Layouts
* Stylesheets

**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()

**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">

<imgsrc={{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]}}">

<imgsrc={{ 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.

**CSS.files**

**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;

}

.displayCategoryul 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;

}

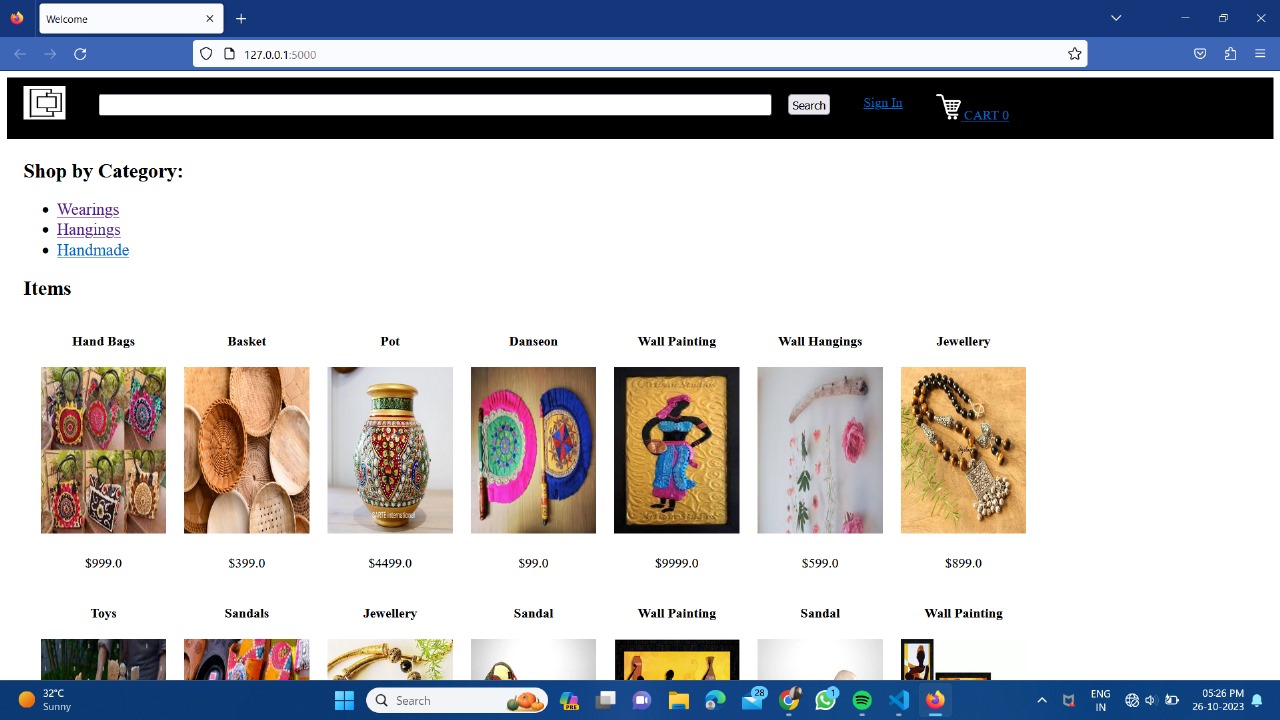
#addToCart {

font-size: 20px;

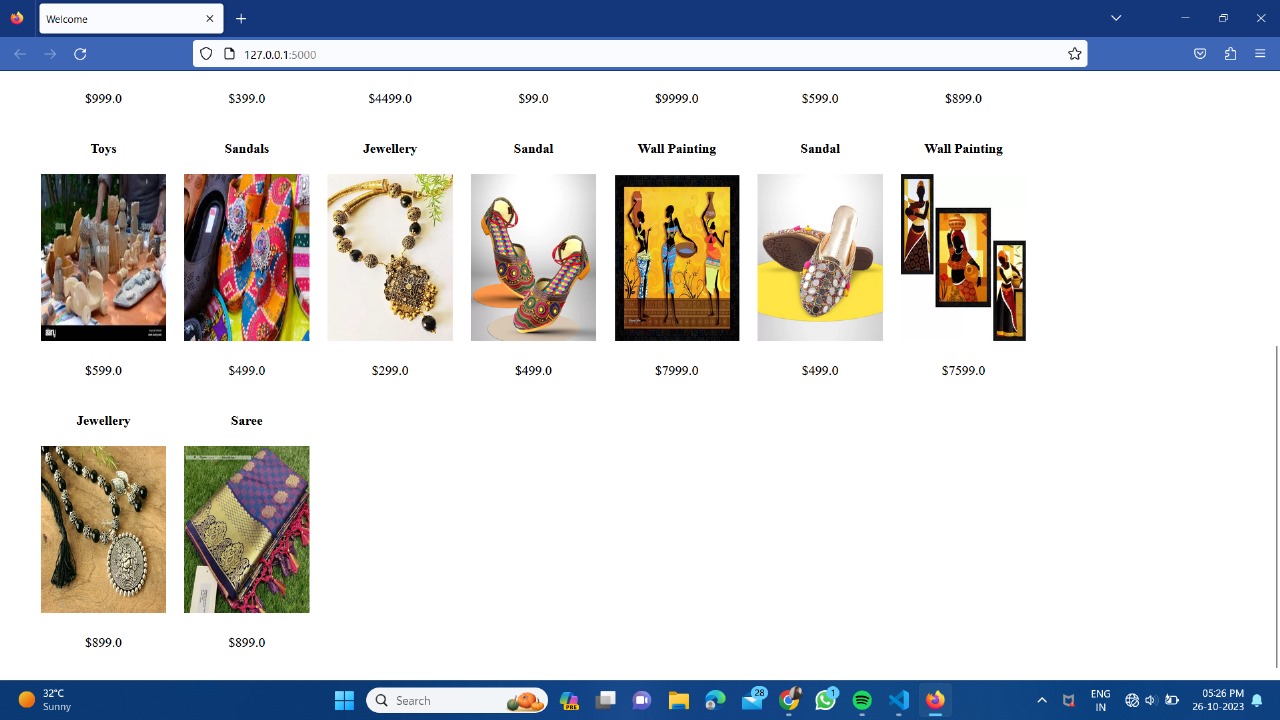
}

The above give files are the css files that had been used to design the e-commerce applications front end.

**SCREENSHOTS:**



After executing the above code we get the output which is shown in the screenshot. This is the home page of our e-commerce application.It is also our idea of platform layout



With adding the database to the web application we can get the user to register our website and add products, remove products, or any updates can also be done.