E-commerce Application on IBM Cloud Foundry

Phase 3: Development Part 1

In this part you will begin building your project.

Begin building the artisanal e-commerce platform on IBM Cloud Foundry.

Design the platform layout and create a database to store product information.

INTRODUCTION:

Phase 3: Development Part 1 - Student Documentation

In this part you will begin building your project. Begin building the artisanal e-commerce platform on IBM Cloud Foundry. Design the platform layout and create a database to store product information. Welcome to Phase 3 of our project, where you will start the process of building our artisanal e-commerce platform. This phase is a crucial step in turning your project idea into a functional platform. Below, we'll outline the key activities and objectives you should focus on during this stage.

Objectives:

1. **Start Building Your Project:** In this part, you will transition from planning and design to actual implementation. You will begin bringing your artisanal e-commerce platform to life.

   Importance: This objective marks the transition from the planning and design stage to the implementation phase. It is essential because it signifies the initiation of actual development work, turning your project concept into a tangible product.

2. **Utilize IBM Cloud Foundry:** To host your platform, you will utilize IBM Cloud Foundry. This is a cloud-based platform-as-a-service (PaaS) that will provide the infrastructure and resources needed for your project.

   Importance: IBM Cloud Foundry is a cloud-based platform-as-a-service (PaaS) that offers various infrastructure and resources. It is crucial for your project because it provides the necessary environment for hosting your e-commerce platform. By using this platform, you can take advantage of scalable and reliable cloud infrastructure without the need for extensive infrastructure management.

3. **Design the Platform Layout:** A well-structured and user-friendly layout is crucial for an e-commerce platform. You will need to design how the platform will look, ensuring it's both aesthetically pleasing and easy to navigate for potential customers.

   Importance: The design of your e-commerce platform's layout is critical for several reasons.

   User Experience (UX): A well-designed layout enhances the user experience by making it easy for customers to navigate your platform, find products, and complete transactions. A positive UX can lead to higher customer satisfaction and increased sales.

Brand Image: Your platform's design is a reflection of your brand. A visually appealing and consistent design can help establish a strong brand image, making your platform more memorable and trustworthy.

Conversion Rates: An effective layout can influence conversion rates, encouraging visitors to become customers. By strategically placing elements like product listings, shopping carts, and call-to-action buttons, you can guide users through the purchasing process.

4. **Create a Database:** An essential component of any e-commerce platform is a database to store product information. You will establish a database that can efficiently and securely store product details, such as product names, descriptions, prices, and images.

   Importance: The database is the backbone of your e-commerce platform, and creating one is crucial for several reasons:

   Data Storage: A database is needed to store essential product information, such as product names, descriptions, prices, and images. Without a database, your platform would have no way to organize, manage, and retrieve this information.

   Data Security: Databases can be designed with security in mind. By creating a database, you can implement measures to protect sensitive customer and product data, helping to ensure the privacy and integrity of information.

   Scalability: A well-designed database can scale as your business grows. It allows you to manage an increasing volume of products and customer data efficiently.

   Query and Search Functionality: Databases provide the capability to search and retrieve product information quickly. This is crucial for users to find the products they want, which can improve their experience and encourage more sales.

Key Activities:

1. **Set Up IBM Cloud Foundry:** Begin by creating an account on IBM Cloud and set up a Cloud Foundry environment. This will be the backbone of your project's infrastructure.
2. **Platform Layout Design:** Work on creating a layout for your platform. Consider the user interface (UI) and user experience (UX) aspects. Ensure that the design aligns with your project's goals and the preferences of your target audience.
3. **Database Creation:** Select an appropriate database system for your project, whether it's a relational database like MySQL or a NoSQL database like MongoDB. Set up the database, define its schema, and establish the necessary tables or collections to store product information.

4. **Data Modeling:** Plan how you will structure the product data within the database. This includes defining fields for product attributes, organizing categories, and establishing relationships between different data elements.

5. **Database Connectivity:** Integrate the database with your e-commerce platform. Develop or configure the necessary connectors or APIs to ensure seamless communication between your platform and the database.

6. **Testing:** It's essential to conduct thorough testing to verify that the database and platform are working as expected. Test various functionalities, such as adding, updating, and retrieving product information.

Documentation:

Throughout this phase, be sure to maintain detailed documentation of your progress. This documentation should include:

- **Design Specifications:** Record your platform layout design, including wireframes and mock-ups.
- **Database Schema:** Document the database schema, detailing the structure of the database and the relationships between tables or collections.
- **Codebase:** Keep track of the code you write, along with comments and explanations for your work.
- **Testing Results:** Document the results of your testing, including any issues you encountered and how you resolved them.

Design Specifications:

Design specifications for an e-commerce application project focused on artisan products should be comprehensive and tailored to the unique needs of such a platform. Here are some key considerations and elements to include in your design specifications:

1. **User Interface (UI) and User Experience (UX):**
    - Intuitive navigation: Ensure that the user interface is easy to navigate, allowing users to browse and search for artisan products efficiently.
    - Responsive design: Make the application responsive to various devices and screen sizes, including mobile phones and tablets.

- Visual aesthetics: Incorporate design elements that reflect the artistic and unique nature of the products being sold.

2. **Product Listings and Display:**

    - Detailed product descriptions: Provide space for artisans to describe their products, including materials, dimensions, and any unique features.
    - High-quality images: Allow artisans to upload high-resolution images of their products to showcase their work effectively.
    - Product categories and tags: Implement a robust categorization system to help users find products easily.

3. **Search and Filter:**

    - Advanced search options: Enable users to search by product type, artisan, price range, or other relevant filters.
    - Search suggestions: Implement auto-suggestions to help users find products more quickly.

4. **Artisan Profiles:**

    - Artisan bios and portfolios: Allow artisans to create detailed profiles with information about their craft, experience, and previous work.
    - Contact information: Include a way for users to contact artisans for inquiries or custom orders.

5. **Shopping Cart and Checkout:**

    - Secure shopping cart: Ensure the cart is easy to use and provides a transparent view of items selected for purchase.
    - Multiple payment options: Support various payment methods, including credit cards, digital wallets, and potentially cryptocurrencies.
    - Shipping options: Integrate with shipping carriers and provide real-time shipping cost estimates.

6. **Reviews and Ratings:**

    - User reviews: Allow customers to leave reviews and ratings for products and artisans.
    - Moderation system: Implement a review moderation system to maintain the quality and authenticity of reviews.

7. **Security:**

    - SSL encryption: Secure user data and payment information with SSL encryption.
    - Payment gateway security: Ensure the payment processing system complies with industry standards for security.

8. **Inventory Management:**

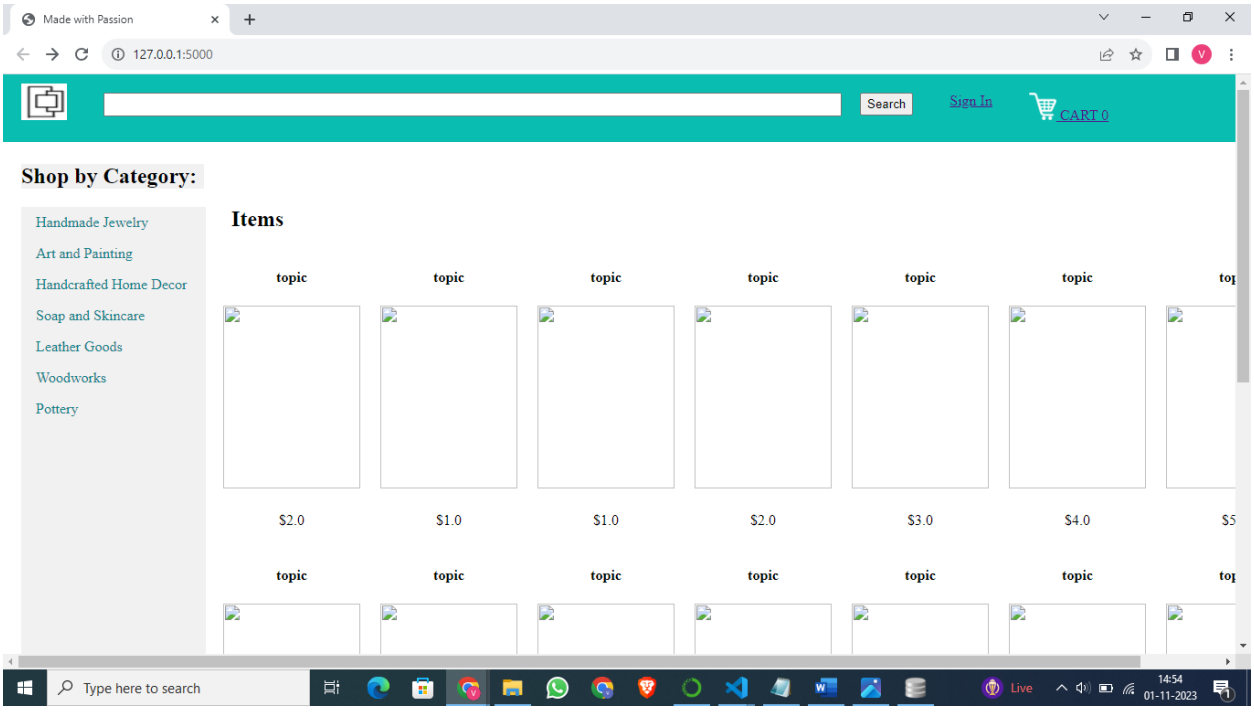
    - Stock tracking: Implement a system that keeps track of product availability and notifies artisans when their products are running low.
    - Product variations: Support different product variations (e.g., size, color) and manage inventory for each variation.

9. **Content Management:**

    - Blog and articles: Include a content management system for artisans to publish articles and stories related to their craft.
    - Event listings: If applicable, provide a calendar for showcasing artisan events, workshops, or exhibitions.

10. **Social Integration:**

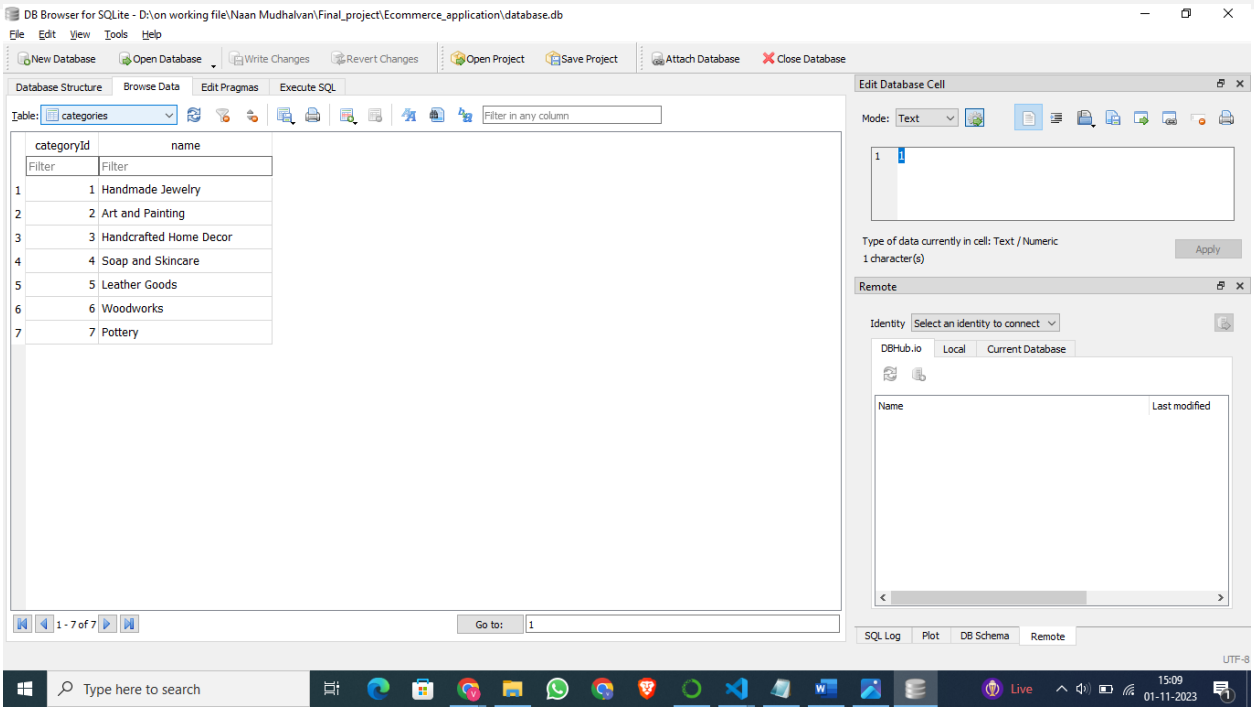
    - Social sharing: Allow users to share products and artisan profiles on social media.
    - Social logins: Enable users to sign in using their social media accounts.

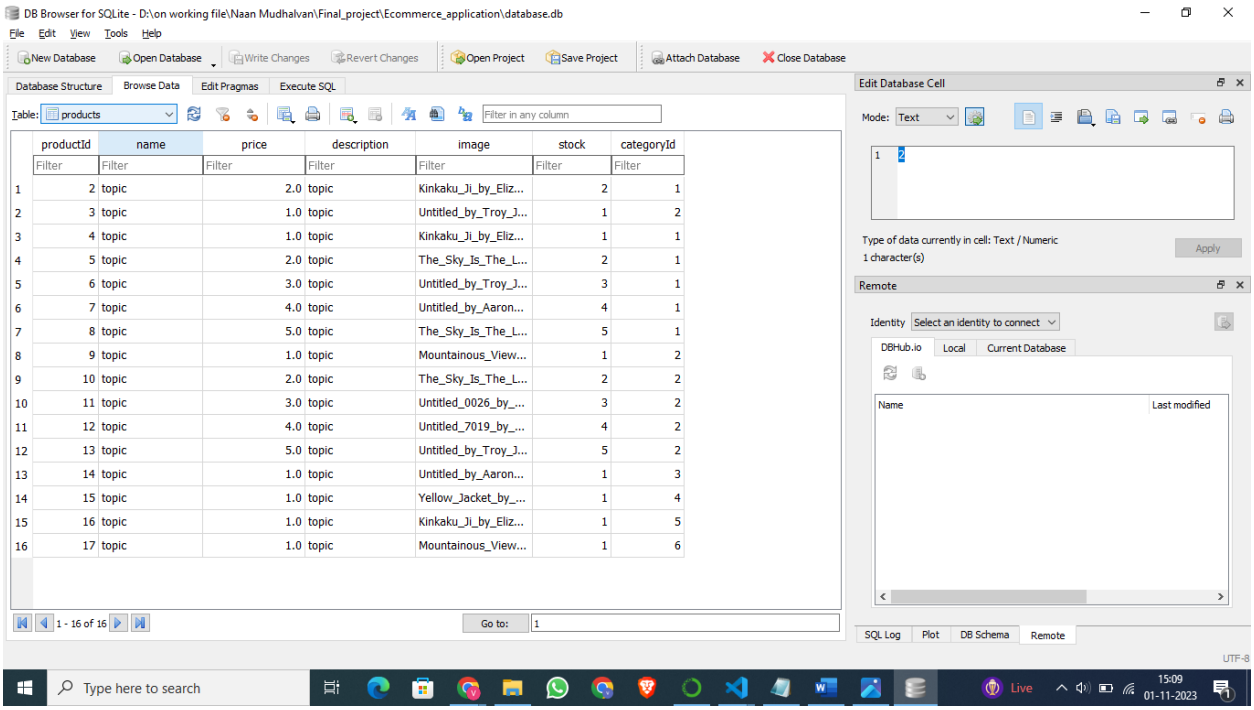


Database Schema:

1 . Database Schema for Product Description :

```
import sqlite3

#Open database
conn = sqlite3.connect('C:/Users/admin/Desktop/database/E_commerce_database.db')

#Create table
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 categories
        (categoryId INTEGER PRIMARY KEY,
        name TEXT
        )''')



conn.close()
```


DB Browser for SQLite - D:\on working file\Naan Mudhalvan\Final_project\Ecommerce_application\database.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database
Database Structure Browse Data Edit Pragmas Execute SQL
Table: categories
Filter in any column
categoryId name
Filter Filter
1 1 Handmade Jewelry
2 2 Art and Painting
3 3 Handcrafted Home Decor
4 4 Soap and Skincare
5 5 Leather Goods
6 6 Woodworks
7 7 Pottery
1 - 7 of 7
Go to: 1
Edit Database Cell
Mode: Text
1 1
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity Select an identity to connect
DBHub.io Local Current Database
Name Last modified
SQL Log Plot DB Schema Remote
UTF-8
Type here to search
Live
15:09
01-11-2023


DB Browser for SQLite - D:\on working file\Naan Mudhalvan\Final_project\Ecommerce_application\database.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database
Database Structure Browse Data Edit Pragmas Execute SQL
Table: products
Filter in any column
productId name price description image stock categoryId
Filter Filter Filter Filter Filter Filter Filter
1 2 topic 2.0 topic Kinkaku_Ji_by_Eliz... 2 1
2 3 topic 1.0 topic Untitled_by_Troy_J... 1 2
3 4 topic 1.0 topic Kinkaku_Ji_by_Eliz... 1 1
4 5 topic 2.0 topic The_Sky_Is_The_L... 2 1
5 6 topic 3.0 topic Untitled_by_Troy_J... 3 1
6 7 topic 4.0 topic Untitled_by_Aaron... 4 1
7 8 topic 5.0 topic The_Sky_Is_The_L... 5 1
8 9 topic 1.0 topic Mountainous_View... 1 2
9 10 topic 2.0 topic The_Sky_Is_The_L... 2 2
10 11 topic 3.0 topic Untitled_0026_by_... 3 2
11 12 topic 4.0 topic Untitled_7019_by_... 4 2
12 13 topic 5.0 topic Untitled_by_Troy_J... 5 2
13 14 topic 1.0 topic Untitled_by_Aaron... 1 3
14 15 topic 1.0 topic Yellow_Jacket_by_... 1 4
15 16 topic 1.0 topic Kinkaku_Ji_by_Eliz... 1 5
16 17 topic 1.0 topic Mountainous_View... 1 6
1 - 16 of 16
Go to: 1
Edit Database Cell
Mode: Text
1 2
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity Select an identity to connect
DBHub.io Local Current Database
Name Last modified
SQL Log Plot DB Schema Remote
UTF-8
Type here to search
Live
15:09
01-11-2023

Codebase:

Main.py:

```
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

@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("/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("/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)

if __name__ == '__main__':
    app.run(debug=True)
```

Home.html:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Made with Passion</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 style="margin-left: 230px;">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>
```

Home.css:

```
#itemImage {
    height: 200px;
    width: 150px;
}

.display {
    margin-top: 20px;
    margin-left: 20px;
    margin-right: 20px;
    margin-bottom: 20px;
}

table {
    margin-left: 200px;
    border-spacing: 20px;
```

```
}

#productName {
    text-align: center;
    font-weight: bold;
}

#productPrice {
    text-align: center;
}




.displayCategory {
    list-style-type: none;
    margin: 0;
    padding: 0;
    width: 200px;
    background-color: #f1f1f1;
  }


body {
    margin: 0;
}

ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    width: 15%;
    background-color: #f1f1f1;
    position: fixed;
    height: 100%;
    overflow: auto;
}

li a {
    display: block;
    color: rgb(19, 131, 139);
    padding: 8px 16px;
    text-decoration: none;
}
```

```css
li a.active {
    background-color: #09bdb1;
    color: white;
}

li a:hover:not(.active) {
    background-color: #09bdb1;
    color: white;
}
```