Shanmuganathan engineering college , code:9124

domain:cloud computing

team members :

- Thrisha k 912421104052
- Srividhya.S 912421104044
- Soundariya.r 912421104042
- Keerthika.s 912421104302

PROJECT:

E-COMMERCE APP

PHASE 5:

GIVEN STATEMENT:

Problem Statement: Build an artisanal e-commerce platform using IBM Cloud Foundry. Connect skilled artisans with a global audience. Showcase handmade products, from exquisite jewelry to artistic home decor. Implement secure shopping carts, smooth payment gateways, and an intuitive checkout process. Nurture creativity and support small businesses through an artisan's dream marketplace!

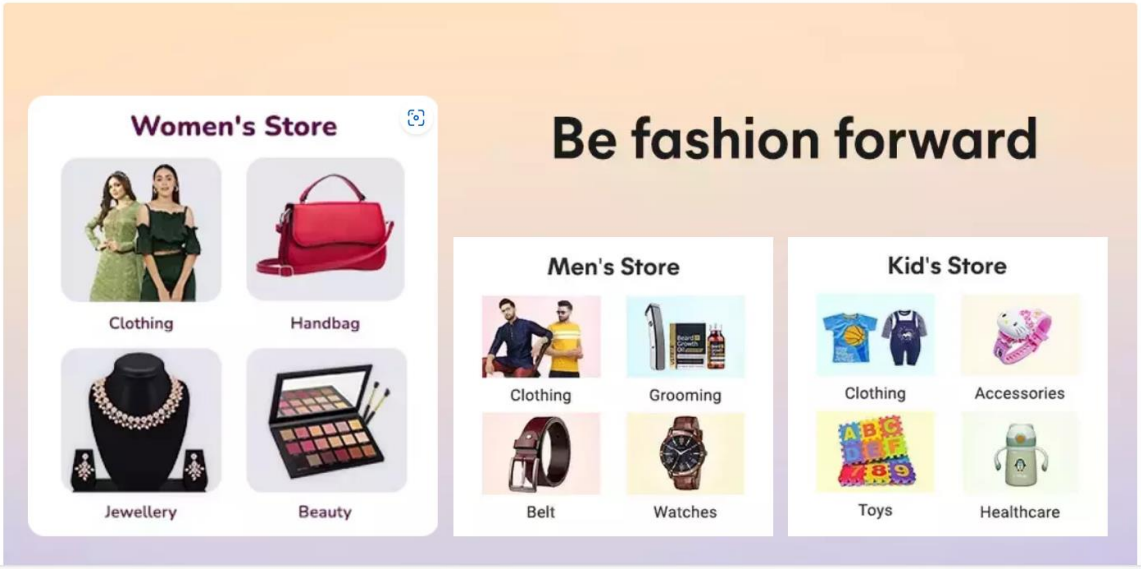
Design thinking

Platform design:

Design the platform layout with sections for product categories, individual product pages , shopping cart , checkout and payment.

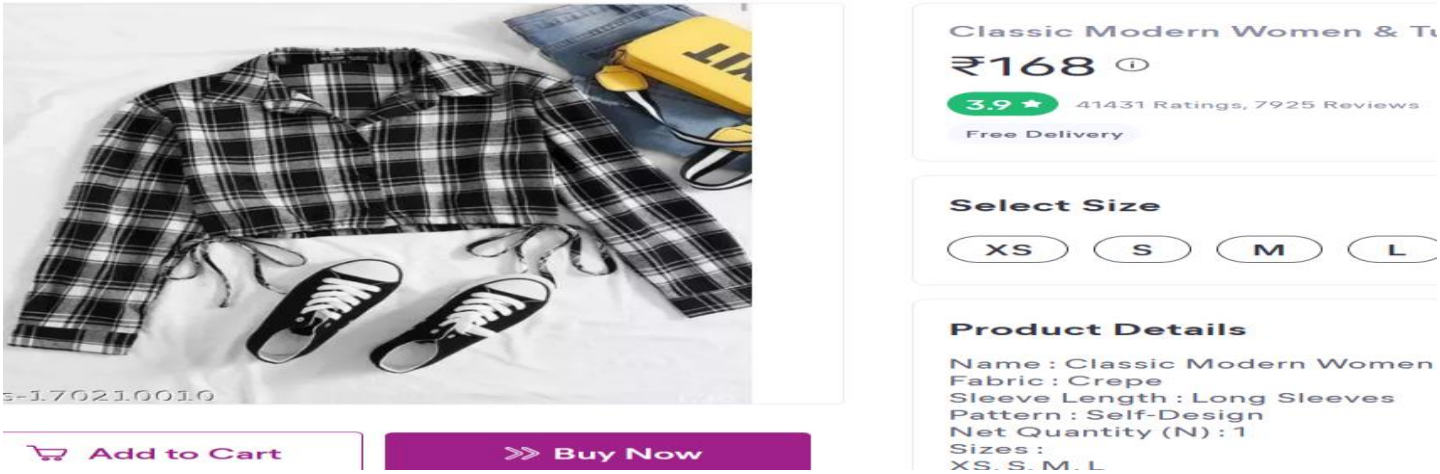
Product Categories:

- Below the header, create a section for product categories. This section should display images and text links to different product categories.
- Each category image should be a clickable link leading to a category page.



Individual product pages :

- Each product should have an image, product name, price, and an "Add to Cart" button



Shopping cart and checkout payment:

- After adding items to the cart, users can proceed to checkout.
- On the checkout page, provide various payment options such as credit/debit card, PayPal, or other payment gateways.

Product showcase:

Create a database to store product information such as images, descriptions, prices and categories.

Creating tables : **Products Table:**

Attributes: ProductID (Primary Key, Auto-incremented),ProductName (String),Description (Text),Price (Decimal),CategoryID (Foreign Key),ImageURL (String)

Categories Table:

Attributes: CategoryID (Primary Key, Auto-incremented),CategoryName (String)

Use Cases From Backend

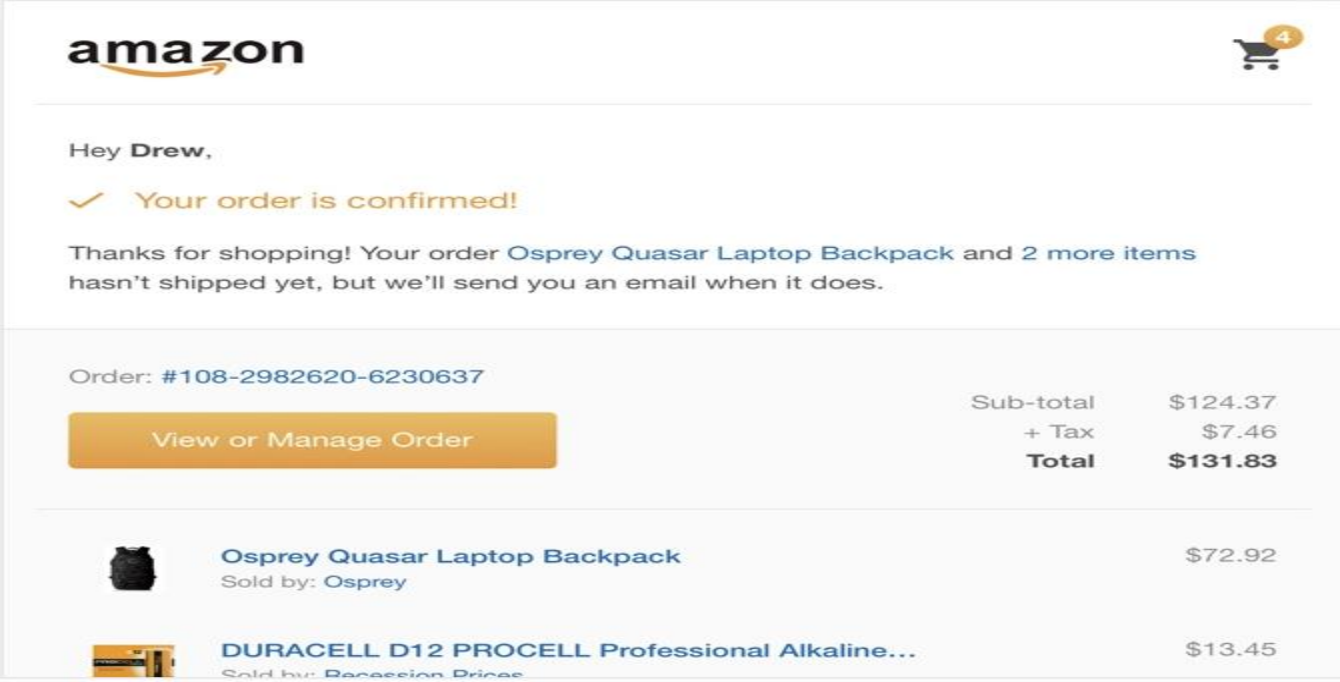
Product Listing: Retrieve a list of products with their names, prices, and images for displaying on the website.

Sql query*:* *SELECT ProductName, Price, ImageURL FROM Products;*

Category Filtering: Allow users to filter products by category by querying the "Products" table based on the CategoryID field.
Sql query: *SELECT ProductName, Price, ImageURL*
FROM Products
WHERE CategoryID = category_id;

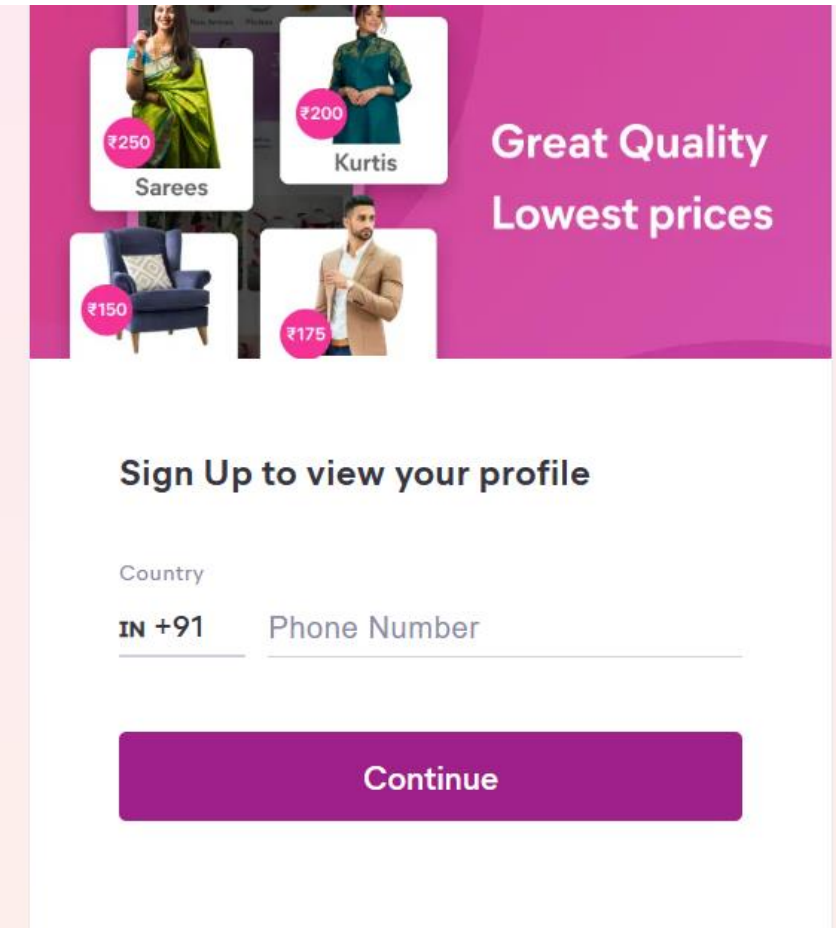
Shopping cart and checkout:
Design and develop the shopping cart functionality and a smooth checkout process.

1. **Shopping Cart Functionality:**Include a shopping cart icon or link in the website's header to give users easy access to their cart.
2. **Smooth Checkout Process:**Create a multi-step checkout process, with each step clearly labeled
3. **Order Confirmation:**Display a confirmation message and order number to users after successfully placing an order.
4. **Error Handling and Validation:**Implement robust error handling to gracefully manage issues such as payment failures, invalid addresses, or out-of-stock items.
5. **Testing:**Thoroughly test the entire checkout process with real and simulated transactions to identify and resolve any issues.



User authentication:

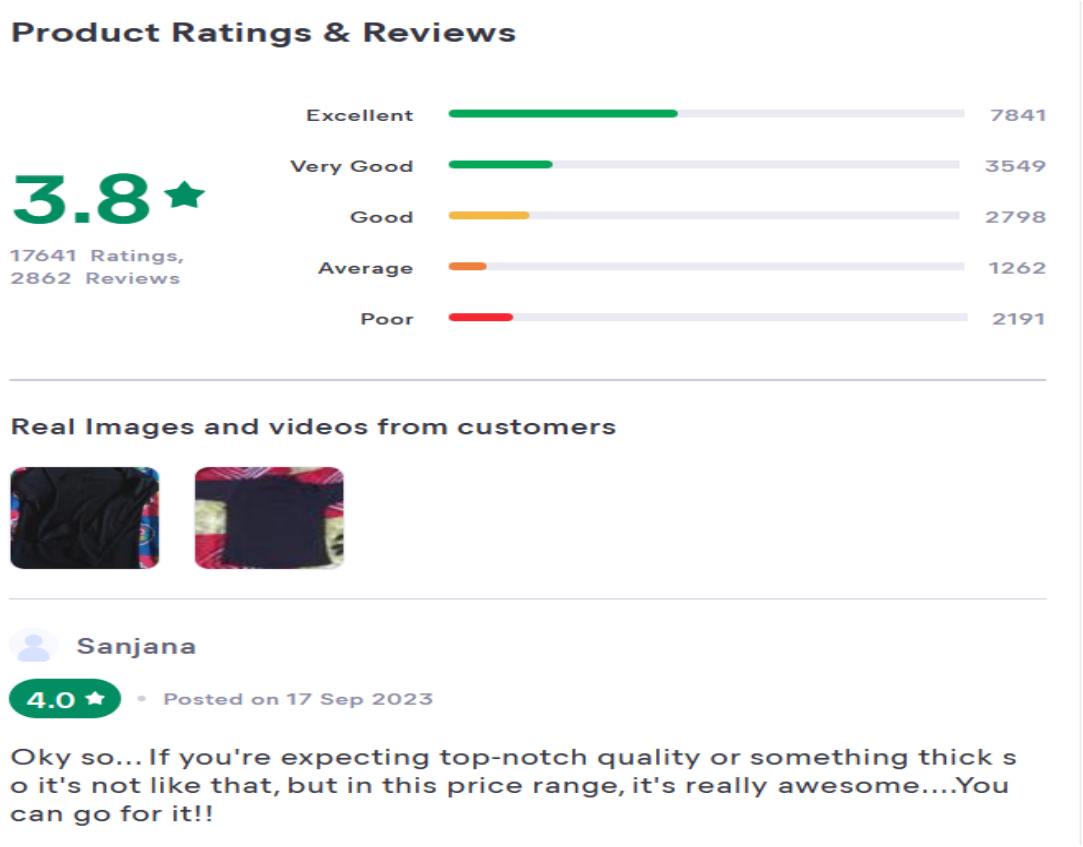
Implement user registration and authentication features to enable artisans and customers to access the platform



User experiance:

Focus on providing an intuitive and visually appealing user experience for both artisan

1. **Reviews and Ratings:**Allow users to leave feedback and ratings for products they've purchased.
2. **Wishlist and Favorites:** - Enable users to create wishlists and save their favorite products for future reference.
3. **Social Sharing:** - Integrate social sharing buttons to encourage users to share their favorite products on social media.
4. **Support and Assistance:** - Offer customer support through chat, email, or phone for any inquiries or issues.



Payment integration:

1. Choose a Payment Gateway:

- ***Research and choose a reputable payment gateway provider that suits your business needs. Popular options include Stripe, PayPal, Square, and Authorize.Net.***

Front-End Integration:

- ***Create a user interface for customers to input payment information.***
- ***Design and implement a payment form that collects credit card details, billing address, and any other required information.***

Back-End Integration:

- ***Implement server-side code to handle payment processing.***
- ***Use the payment gateway's server-side SDK or API to make secure API requests for authorization and capture of funds.***
- ***Verify and validate payment details on the server to prevent fraudulent transactions.***

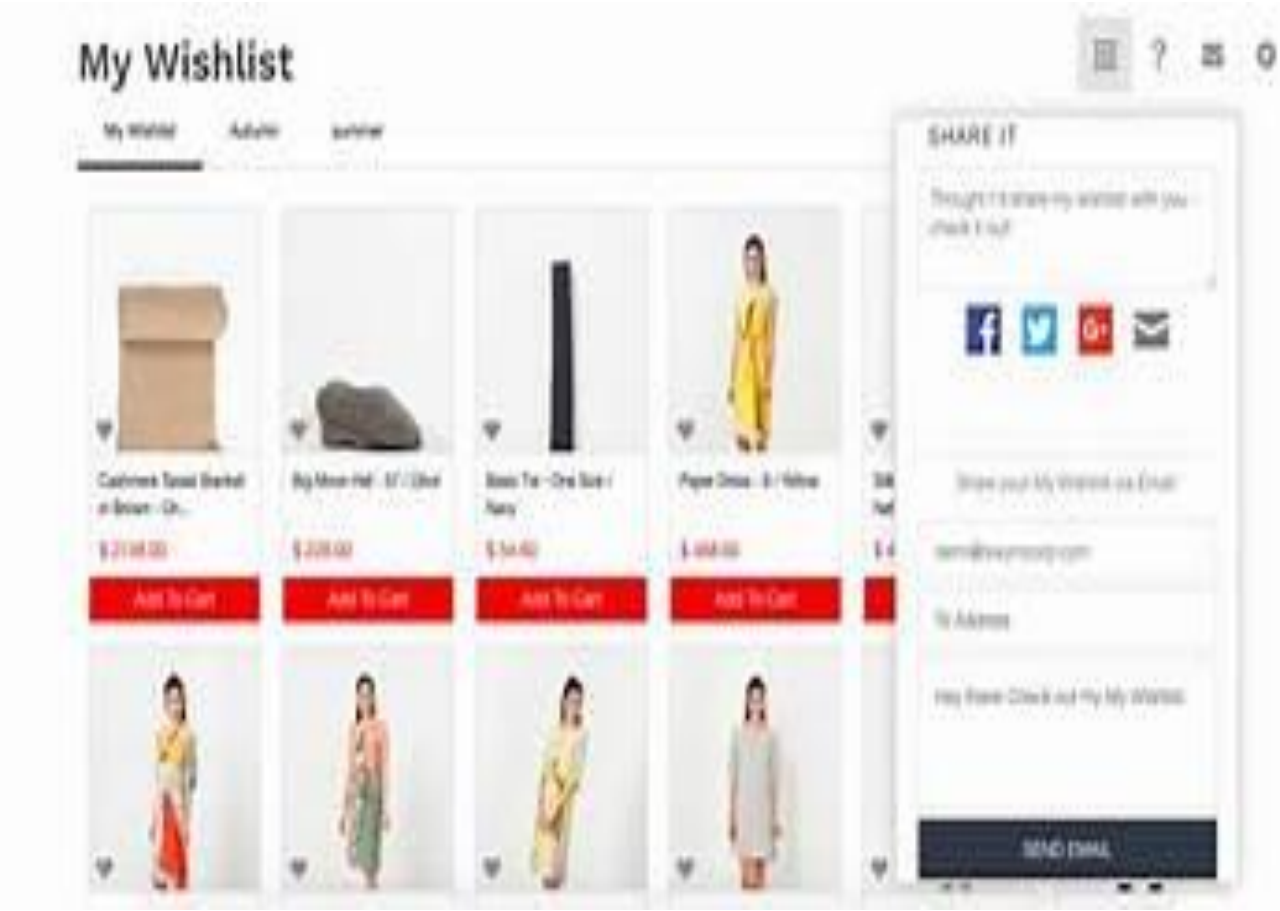
Product Reviews

- Allow users to leave reviews and ratings for products they have purchased.
- Include a comment section for users to provide detailed feedback.
- Display both positive and negative reviews to provide a balanced view.

- Consider implementing a verified purchase badge to enhance credibility.
- Send automated emails to customers encouraging them to review their purchased products.
-

Wishlists

- Enable users to create and manage wishlists of products they are interested in.
- Allow users to share their wishlists with friends and family.
- Implement wishlist notifications to inform users of price drops or product availability.
- Provide the option to make wishlists public or private.
- Integrate social media sharing options for wishlists.



Continuous Improvement

- **Collect and analyze user feedback through surveys and customer support interactions.**
- **Regularly update and optimize the website based on user behavior and preferences.**
- **Stay informed about industry trends and technological advancements to implement new features.**

Personalized Recommendations

- Implement machine learning algorithms to analyze user behavior and preferences.
- Offer personalized product recommendations based on past purchases and browsing history.
- Provide a "Recommended for You" section on the homepage or product pages.
- Allow users to customize their preferences for more accurate recommendations.
- Use data analytics to continuously improve the recommendation engine.

User Engagement:

- **Implement a responsive and user-friendly design for the website.**
- **Send personalized emails, such as abandoned cart reminders or product recommendations.**
- **Create a loyalty program with rewards for repeat customers.**
- **Host special events, promotions, or sales to encourage user engagement.**
- **Implement a live chat feature for real-time customer support.**

Integration with Social Media

- **Allow users to log in or sign up using their social media accounts.**
- **Incorporate social sharing buttons for products and reviews.**
- **Run social media campaigns and promotions.**
- **Use social media analytics to understand user preferences and behavior.**
- **Elements of an advertising campaign, such as commercials, webinars and social media advertisements, can all have improvement from social media posting. If your department wants to further use an advertisement video, posting it to social media can help it circulate even after the adperiod is over.**
- **Run social media campaigns and promotions.**
- **Use social media analytics to understand user preferences and behavior.**

- **Elements of an advertising campaign, such as commercials, webinars and social media advertisements, can all have improvement from social media posting. If your department wants to further use an advertisement video, posting it to social media can help it circulate even after the adperiod is over.**

Mobile Optimization

- **Ensure that the website is optimized for mobile devices to cater to users on various platforms.**
- **Implement a mobile app for a more seamless and personalized shopping experience.**
- **Elements of an advertising campaign, such as commercials, webinars and social media advertisements, can all have improvement from social media posting. If your department wants to further use an advertisement video, posting it to social media can help it circulate even after the adperiod is over.**
- **Run social media campaigns and promotions.**
- **Use social media analytics to understand user preferences and behavior.**
-

Development Part 1

1. Create an IBM Cloud Account:

If you don't have an IBM Cloud account, sign up for one. You can do this by visiting the [IBM Cloud website] (**https://cloud.ibm.com/registration**) and following the registration process.

2. Choose the Appropriate Database Service:

Select the IBM Cloud Database service that best suits your project's needs. As mentioned earlier, you can choose between Db2 or MongoDB, depending on your dataset and requirements.

3. Set Up a Database Instance:

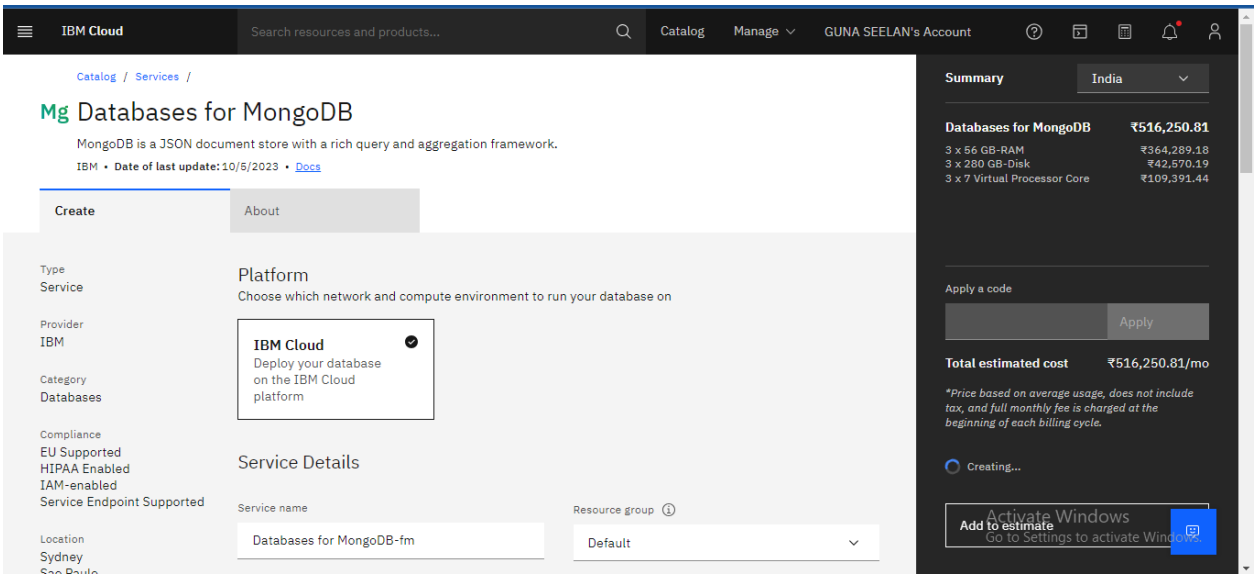
For Db2:

- Log in to your IBM Cloud account.
- From the IBM Cloud dashboard, click on the "Create Resource" button.

- In the catalog, select "Databases" and then "Db2."
- Follow the on-screen instructions to configure your Db2 database instance, including specifying the instance name, region, and other settings.
- Create the instance.

For MongoDB:

- Log in to your IBM Cloud account.
- From the IBM Cloud dashboard, click on the "Create Resource" button.
- In the catalog, select "Databases" and then "MongoDB."
- Follow the on-screen instructions to configure your MongoDB database instance, including specifying the instance name, region, and other settings.
- Create the instance.

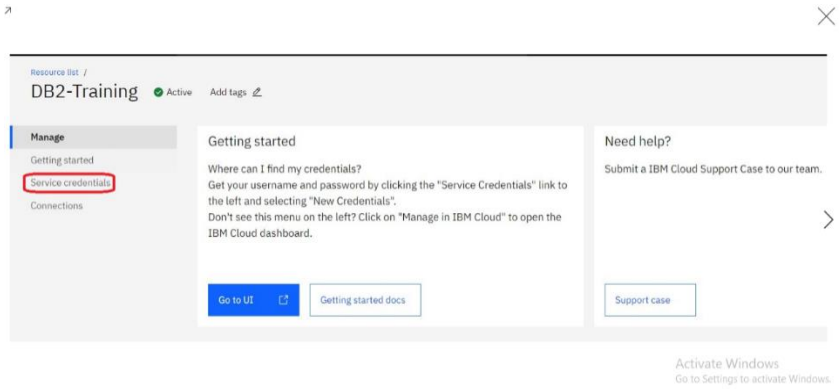


4. Develop e commerce app:

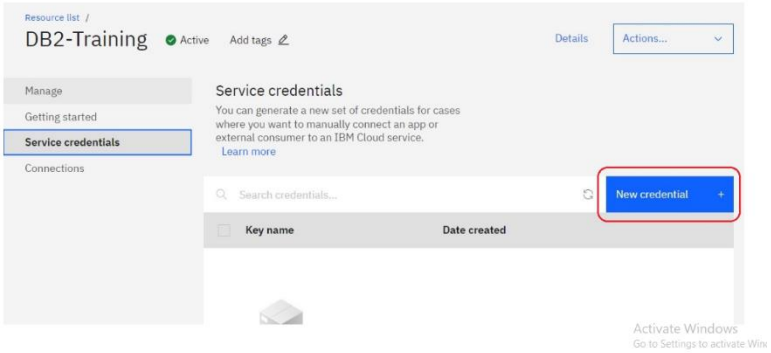
After setting up your database instance, you can start developing queries or scripts to explore and analyze your dataset. The type of queries and scripts you write will depend on the nature of your dataset and your analysis goals. You can use SQL for Db2 or MongoDB's query language for MongoDB.

Creating Service Credentials the IBM DB2 database

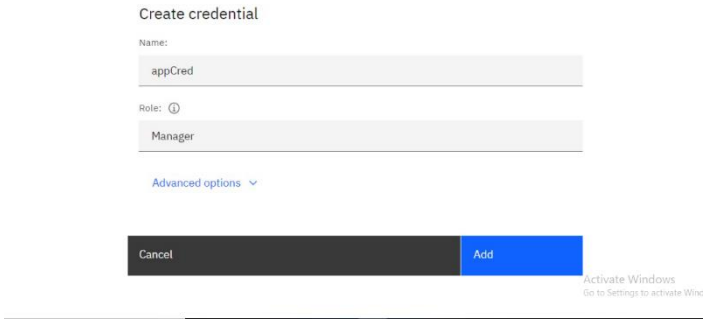
- In the resource list screen of IBM Cloud, click on the DB2 service (displayed under Services and software category) that you created
- From the service page, select the menu option "**Service Credentials**" to create / access the credentials of the db2 database



- Click on **New Credential** button in the Service Credential page to create a new credential

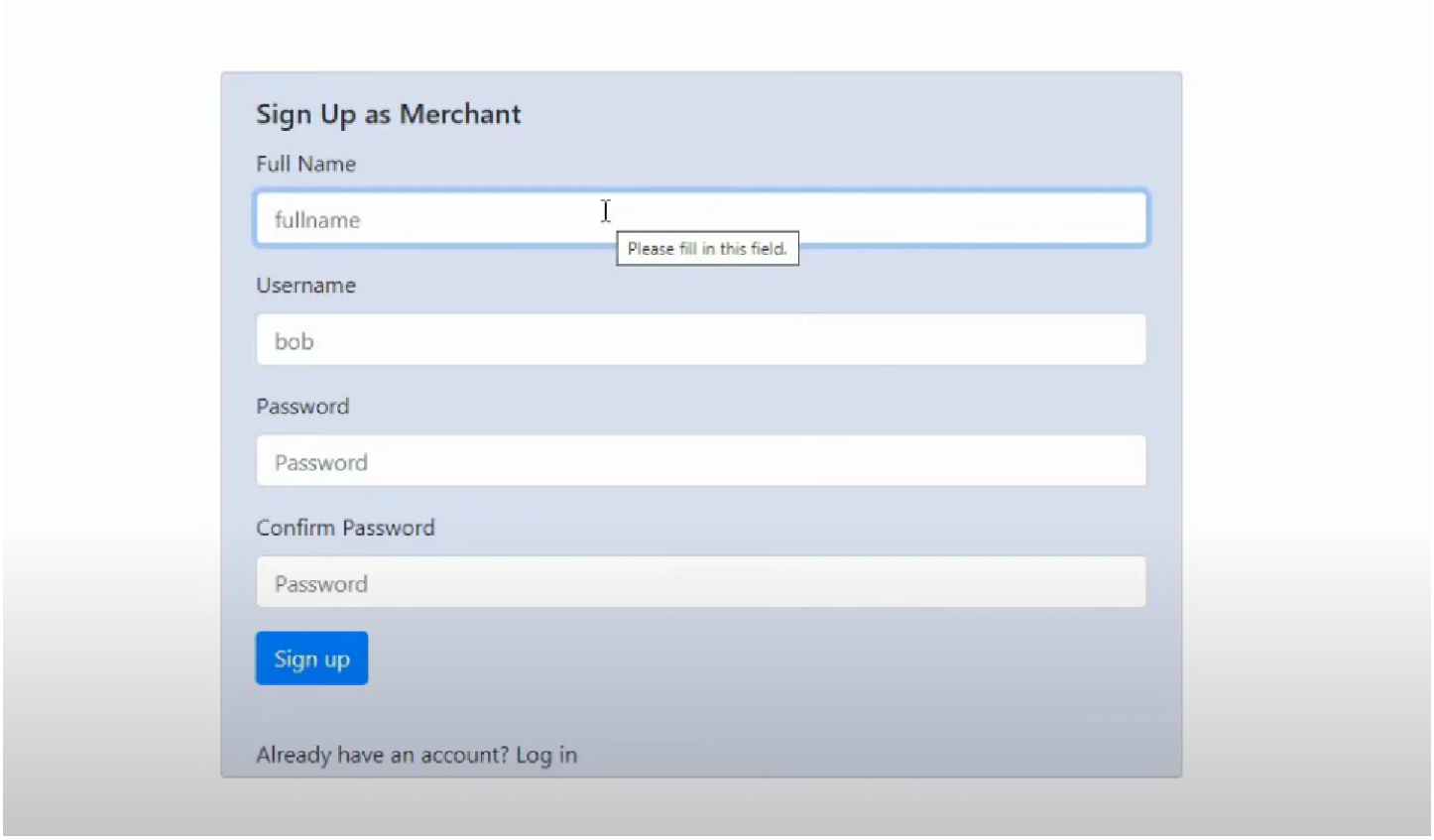


- Provide the any name for service credential (e.g. **appCred**) and click on **Add**



- New credential gets created and is displayed. Expand the newly to created credential to get the all the details that is required for client application to connect to the database. Note down the value for the following properties separately, which we will use it later to configure our application to connect to this database.

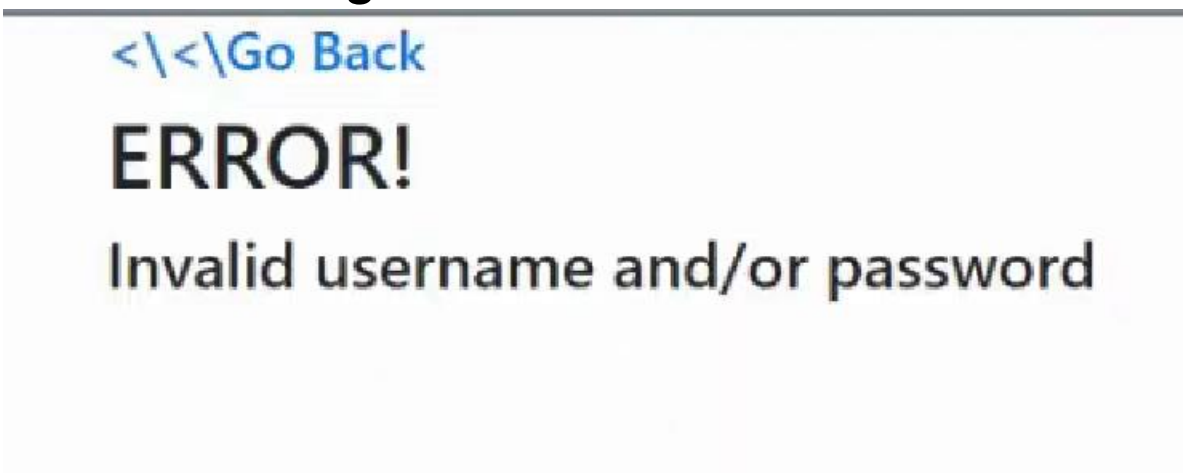
Creating authentication page:



Html code is given below

```
    <div class="container">
        <div class="alert alert-success" role="alert">
            {{ msg }}
        </div>
    </div>
{% endif %}
<div class="card" id="topdiv0">
        <form class="px-4 py-3" method="POST" action="{{ url_for('login')}}">
            <h5 class="card-title">Login as Merchant</h5>
        <div class="form-group">
          <label for="username">Username</label>
          <input type="text" required="required" class="form-control" name="username" placeholder="bob" autofocus>
        </div>
        <div class="form-group">
          <label for="password">Password</label>
          <input type="password" required="required" class="form-control" name="password" placeholder="Password">
        </div>
        <button type="submit" class="btn btn-primary">Sign in</button>
    </form>
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" href="{{ url_for('signup') }}">New around here? Sign up</a>
</div>
```

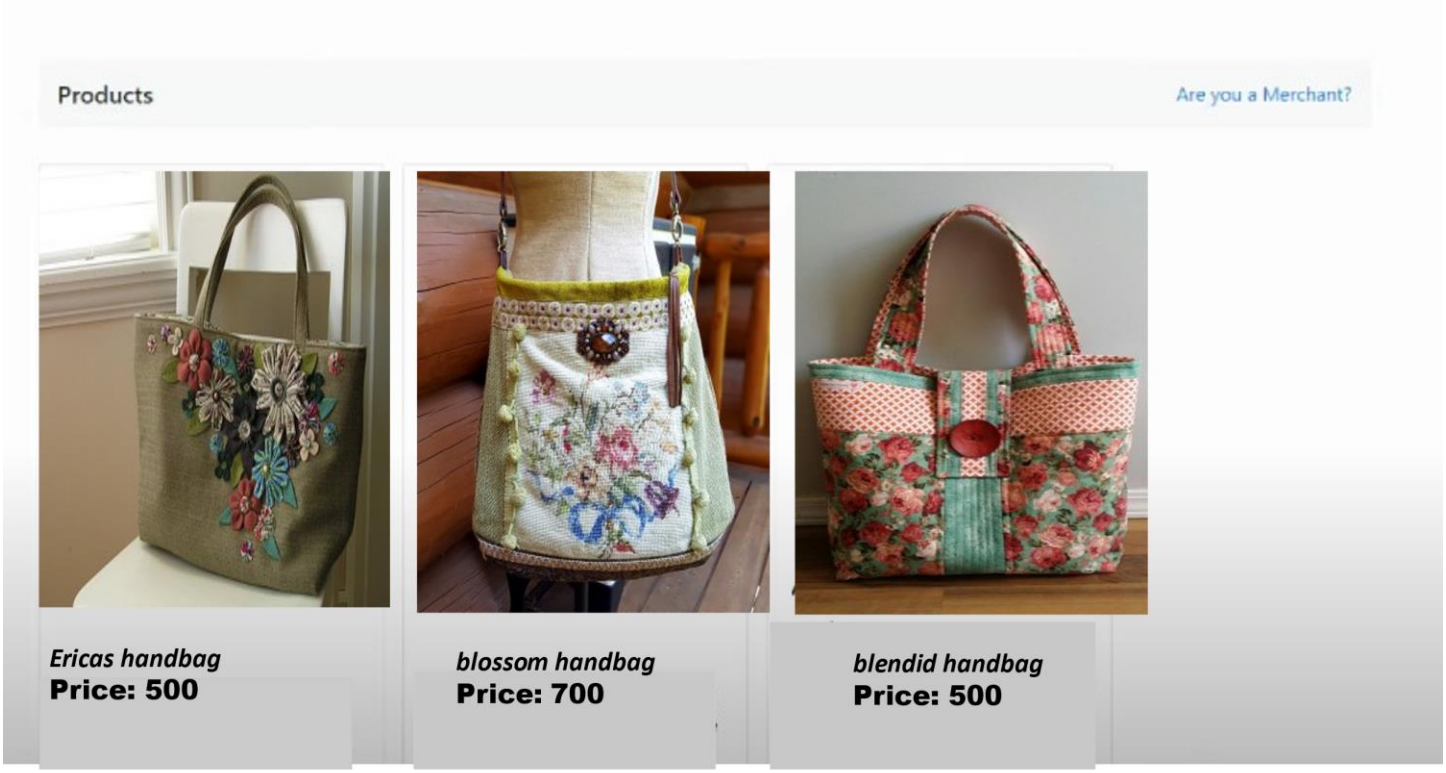
To validate login:



Code:

```
{% extends "layout.html" %}

{% block title %}
    error
{% endblock %}

{% block body %}
    <div class="container">
        <a href="{{ url_for('index') }}"><h4><\<\Go Back</h4></a>
        <h1>ERROR!</h1>
        <h3>{{ message }}</h3>
    </div>
{% endblock %}
```

Creating catalog for products:

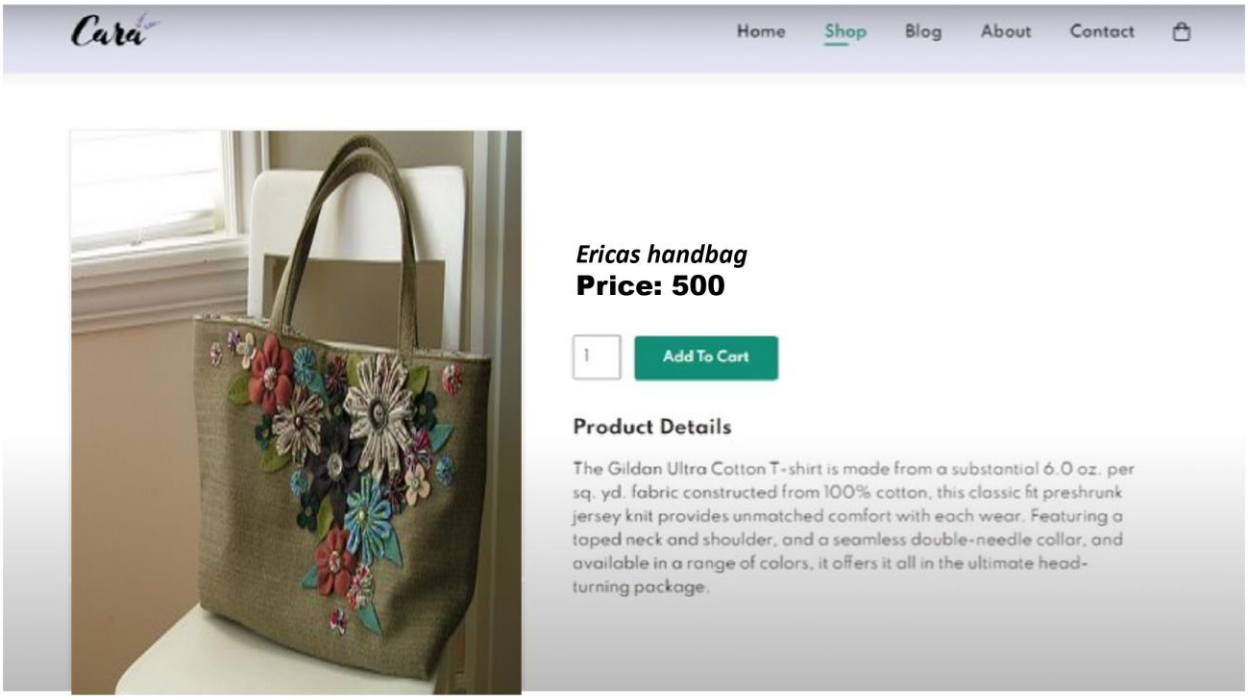


html code for featured products

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1
    <title>Featured Products</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
        }

        .product-container {
            display: grid;
            grid-template-columns: repeat(3, 1fr);
            gap: 20px;
        }

        .product {
            border: 1px solid #ddd;
            border-radius: 8px;
            padding: 15px;
            text-align: center;
        }

        .product img {
            max-width: 100%;
            height: auto;
            border-radius: 6px;
        }

        .product h3 {
            margin-top: 10px;
        }

        .product p {
            font-weight: bold;
            color: #4CAF50;
            margin: 10px 0;
        }
    </style>
</head>
```

```html
<body>
    <h2>Featured Products</h2>

    <div class="product-container">
        <!-- Product 1 -->
        <div class="product">
            <img src="product1.jpg" alt="Product 1">
            <h3>Product 1</h3>
            <p>$19.99</p>
        </div>

        <!-- Product 2 -->
        <div class="product">
            <img src="product2.jpg" alt="Product 2">
            <h3>Product 2</h3>
            <p>$24.99</p>
        </div>

        <!-- Product 3 -->
        <div class="product">
            <img src="product3.jpg" alt="Product 3">
            <h3>Product 3</h3>
            <p>$29.99</p>
        </div>

        <!-- Product 4 -->
        <div class="product">
            <img src="product4.jpg" alt="Product 4">
            <h3>Product 4</h3>
            <p>$39.99</p>
        </div>

        <!-- Product 5 -->
        <div class="product">
            <img src="product5.jpg" alt="Product 5">
            <h3>Product 5</h3>
            <p>$49.99</p>
        </div>

        <!-- Product 6 -->
        <div class="product">
            <img src="product6.jpg" alt="Product 6">
            <h3>Product 6</h3>
```

Creating single product description:



Html code for given

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Artisan Marketplace - Product Details</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            margin: 0;
            padding: 0;
            box-sizing: border-box;
            background-color: #f4f4f4;
        }

        header {
            background-color: #333;
            color: white;
            padding: 1em;
            text-align: center;
        }

        #product-details {
            max-width: 600px;
            margin: 20px auto;
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            padding: 20px;
        }

        #product-details img {
            max-width: 100%;
            border-radius: 8px;
        }

        #add-to-cart-btn {
            background-color: #333;
            color: white;
            padding: 10px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
```

```html
        }

        #add-to-cart-btn:hover {
            background-color: #555;
        }
    </style>
</head>
<body>

    <!-- Header Section -->
    <header>
        <h1>Artisan Marketplace</h1>
    </header>

    <!-- Product Details Section -->
    <div id="product-details">
        <h2>Product Name</h2>
        <img src="product-image.jpg" alt="Product Image">
        <p>Description of the product goes here. This could include details a
        <p>$20.00</p>
        <button id="add-to-cart-btn" onclick="addToCart()">Add to Cart</butto
    </div>

    <script>
        function addToCart() {
            // Add logic to handle adding the product to the shopping cart
            alert('Product added to the cart!');
        }
    </script>

</body>
</html>
```

Connecting database:

As part of your data storing you may need to perform data cleaning and transformation. This can involve database

Remember that I can provide guidance, answer questions, and help with SQL queries or MongoDB queries if you encounter specific issues during your project. Feel free to ask for assistance with any part of your project, and I'll do my best to help you successfully complete it.

```python
import os
import uuid
from flask import Flask, session,render_template,request, Response, redirect, send_from_directory
from werkzeug.utils import secure_filename
from werkzeug.security import check_password_hash, generate_password_hash
from db import db_init, db
from models import  User, Product
from datetime import datetime
from flask_session import Session
from helpers import login_required

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///items.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db_init(app)

# Configure session to use filesystem
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

#static file path
@app.route("/static/<path:path>")
def static_dir(path):
    return send_from_directory("static", path)

#signup as merchant
@app.route("/signup", methods=["GET","POST"])
def signup():
    if request.method=="POST":
        session.clear()
        password = request.form.get("password")
        repassword = request.form.get("repassword")
        if(password!=repassword):
```

```
            return render_template("error.html", message="Passwords do not match!")

        #hash password
        pw_hash = generate_password_hash(password, method='pbkdf2:sha256', salt_length=8)

        fullname = request.form.get("fullname")
        username = request.form.get("username")
        #store in database
        new_user =User(fullname=fullname,username=username,password=pw_hash)
        try:
            db.session.add(new_user)
            db.session.commit()
        except:
            return render_template("error.html", message="Username already exists!")
        return render_template("login.html", msg="Account created!")
    return render_template("signup.html")

#login as merchant
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method=="POST":
        session.clear()
        username = request.form.get("username")
        password = request.form.get("password")
        result = User.query.filter_by(username=username).first()
        print(result)
        # Ensure username exists and password is correct
        if result == None or not check_password_hash(result.password, password):
            return render_template("error.html", message="Invalid username and/or password")
        # Remember which user has logged in
        session["username"] = result.username
        return redirect("/home")
    return render_template("login.html")

#logout
@app.route("/logout")
def logout():
    session.clear()
    return redirect("/login")

#view all products
@app.route("/")
```

```python
#view all products
@app.route("/")
def index():
    rows = Product.query.all()
    return render_template("index.html", rows=rows)

#merchant home page to add new products and edit existing products
@app.route("/home", methods=["GET", "POST"], endpoint='home')
@login_required
def home():
    if request.method == "POST":
        image = request.files['image']
        filename = str(uuid.uuid1())+os.path.splitext(image.filename)[1]
        image.save(os.path.join("static/images", filename))
        category= request.form.get("category")
        name = request.form.get("pro_name")
        description = request.form.get("description")
        price_range = request.form.get("price_range")
        comments = request.form.get("comments")
        new_pro = Product(category=category,name=name,description=description,price_range=price_range,comments=comments, filename=filename,
        db.session.add(new_pro)
        db.session.commit()
        rows = Product.query.filter_by(username=session['username'])
        return render_template("home.html", rows=rows, message="Product added")

    rows = Product.query.filter_by(username=session['username'])
    return render_template("home.html", rows=rows)

#when edit product option is selected this function is loaded
@app.route("/edit/<int:pro_id>", methods=["GET", "POST"], endpoint='edit')
@login_required
def edit(pro_id):
    #select only the editing product from db
    result = Product.query.filter_by(pro_id = pro_id).first()
    if request.method == "POST":
        #throw error when some merchant tries to edit product of other merchant
        if result.username != session['username']:
            return render_template("error.html", message="You are not authorized to edit this product")
        category= request.form.get("category")
        name = request.form.get("pro_name")
        description = request.form.get("description")
        price_range = request.form.get("price_range")
```

```python
        #throw error when some merchant tries to edit product of other merchant
        if result.username != session['username']:
            return render_template("error.html", message="You are not authorized to edit this product")
        category= request.form.get("category")
        name = request.form.get("pro_name")
        description = request.form.get("description")
        price_range = requ
        comments = request
        result.category = category
        result.name = name
        result.description = description
        result.comments = comments
        result.price_range = price_range
        db.session.commit()
        rows = Product.query.filter_by(username=session['username'])
        return render_template("home.html", rows=rows, message="Product edited")
    return render_template("edit.html", result=result)
if __name__=='_main_':
    app.run(debug=True)
```

(variable) category: Any

Create a database to store product information such as images, descriptions, prices and categories.

Creating tables : **Products Table:**

Attributes: ProductID (Primary Key, Auto-incremented),ProductName (String),Description (Text),Price (Decimal),CategoryID (Foreign Key),ImageURL (String)

Categories Table:

Attributes: CategoryID (Primary Key, Auto-incremented),CategoryName (String)

Use Cases From Backend

Product Listing: Retrieve a list of products with their names, prices, and images for displaying on the website.

Sql query*:* *SELECT ProductName, Price, ImageURL FROM Products;*

Category Filtering: Allow users to filter products by category by querying the "Products" table based on the CategoryID field.

Sql query: *SELECT ProductName, Price, ImageURL*

FROM Products

WHERE CategoryID = category_id;

Product Details: Display the detailed product description and price when a user selects a specific product.

Sql query: *SELECT ProductName, Description, Price, ImageURL FROM Products WHERE ProductID = product_id;*

Adding New Products: Insert new products into the "Products" table, specifying the product's name, description, price, category, and image URL.

Sql query:*INSERT INTO Products (ProductName, Description, Price, CategoryID, ImageURL)*

VALUES ('New Product Name', 'Product Description', 19.99, 1, 'image_url.jpg');

Updating Product Information: Modify product details such as price, description, or image URL when necessary.

Sql query:*UPDATE Products*

SET Price = 24.99, Description = 'Updated Description'

WHERE ProductID = product_id;

Deleting Products: Remove products from the database when they are no longer available.

Sql query:*DELETE FROM Products WHERE ProductID = product_id;*

Conclusion:

the first part of the development of ecommerce is done using html,css,javascipt and db2 .sign up ,login ,product list,display one product,adding to cart, connect with database db2.

<u>Creating a E- commerce applications on cloud foundry</u>

Creating an e-commerce application on Cloud Foundry involves several steps:

- **SET UP CLOUD FOUNDRY:**

Ensure you have access to a Cloud Foundry environment. You can use a platform like IBM Cloud, Pivotal Web Services, or SAP Cloud Platform, all of which support Cloud Foundry

- **SELECT A PROGRAMMING LANGUAGE:**

Choose a programming language and framework for your e-commerce application. Popular choices include Java (using Spring Boot), Node.js, or Ruby on Rails.

- **DATABASE**:

Decide on a database for your application. Cloud Foundry

supports various databases like PostgreSQL, MySQL, and MongoDB. You can choose the one that best fits your needs.

- **PUSH TO CLOUD FOUNDRY**:

Use the Cloud Foundry command-line interface (cf CLI) to push your application to the Cloud Foundry platform. This will make your application accessible on the cloud.

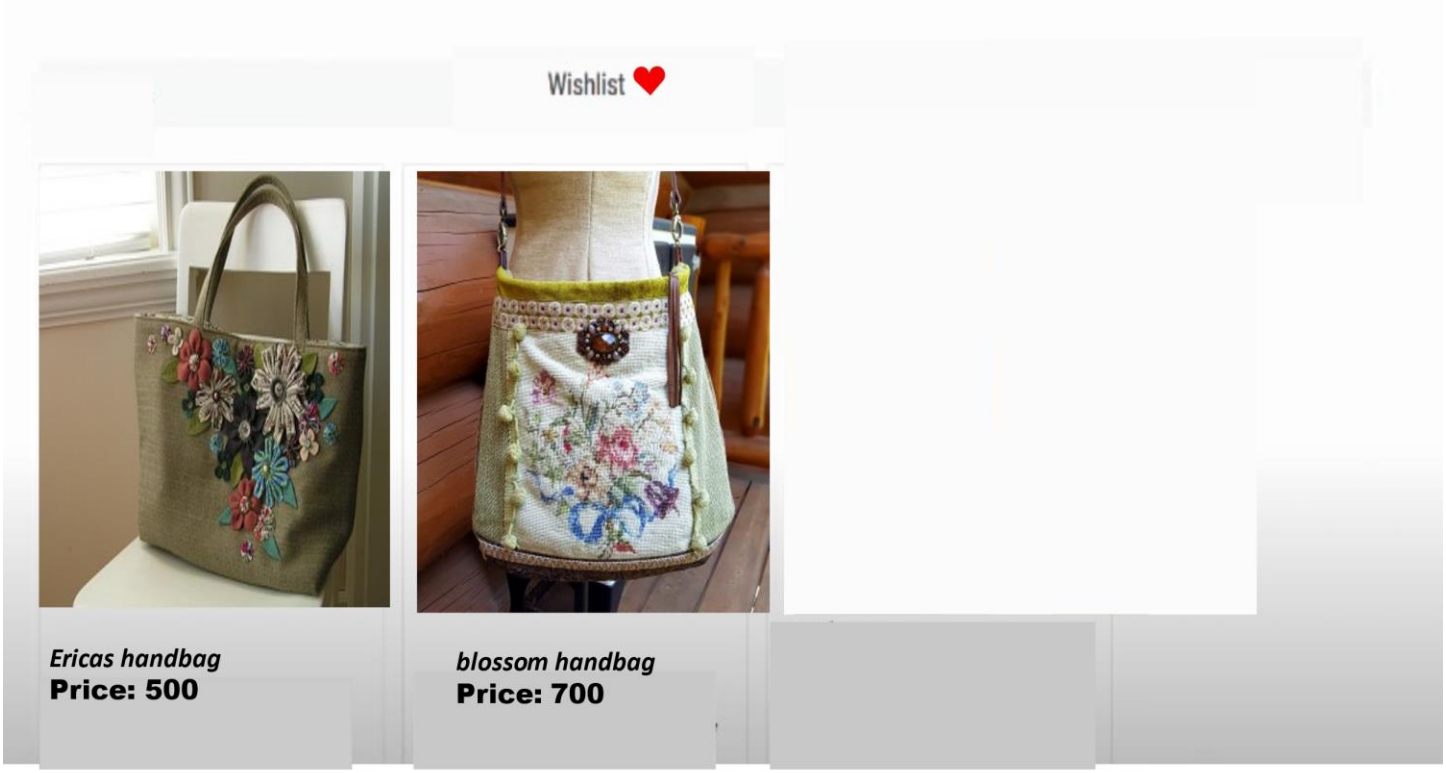
- **SERVICE INTEGRATION:**

Integrate any necessary services like payment gateways, caching, or CDN services. You can use Cloud Foundry's marketplace to add and bind these services to your app.

- **SCALING**:
  Configure auto-scaling and load balancing to ensure your e-commerce application can handle varying levels of traffic.
- **MONITORING AND LOGGING** :
  Set up monitoring and logging to keep an eye on the health and performance of your application. Cloud Foundry often provides tools and integrations for this purpose.

Wishlist products:

To implement a wishlist functionality in an e-commerce website using HTML, you'll typically need to use a combination of HTML, JavaScript, and potentially a back-end language for storing and retrieving wishlist dataTo implement a wishlist functionality in an e-commerce website using HTML, you'll typically need to use a combination of HTML, JavaScript, and potentially a back-end language for storing and retrieving wishlist data



Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Page</title>
  <style>
    .wishlist-button {
      cursor: pointer;
```

```
      color: blue;
      text-decoration: underline;
      margin-left: 10px;
    }
  </style>
</head>
<body>

  <h1>Product Name</h1>
  <p>Description of the product.</p>
  <button class="wishlist-button" onclick="addToWishlist()">Add
to Wishlist</button>

  <script>

    function isLocalStorageSupported() {
      try {
        return 'localStorage' in window && window['localStorage']
!== null;
      } catch (e) {
        return false;
```

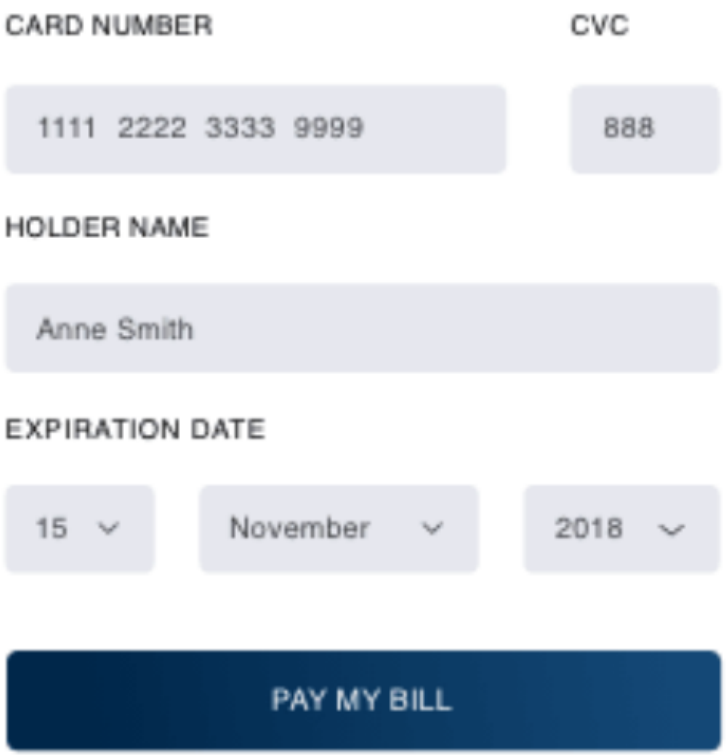
```
    }
  }

  function addToWishlist() {
    var product = {
      id: 1,
      name: "ericas handbag",
      description: "the product is specialised."
    };



    if (isLocalStorageSupported()) {
      // Retrieve the existing wishlist or create a new one
      var wishlist = JSON.parse(localStorage.getItem('wishlist')) ||
[];

      // Check if the product is not already in the wishlist
      if (!wishlist.find(item => item.id === product.id)) {
        // Add the product to the wishlist
        wishlist.push(product);
```

```
        // Save the updated wishlist to localStorage
        localStorage.setItem('wishlist', JSON.stringify(wishlist));

        // Inform the user
        alert('Product added to wishlist!');
      } else {
        // Inform the user that the product is already in the wishlist
        alert('Product is already in the wishlist!');
      }
    } else {

      alert('Wishlist functionality is not available in your browser.');
    }
  }
 </script>
</body>
</html>
```

Payment method:

To working on an e-commerce website and need to handle payments, it's highly recommended to use a secure and

established payment gateway to handle the payment process. Popular payment gateways such as Stripe, PayPal, or others provide secure mechanisms for processing payments without exposing sensitive information to your website.



Code :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

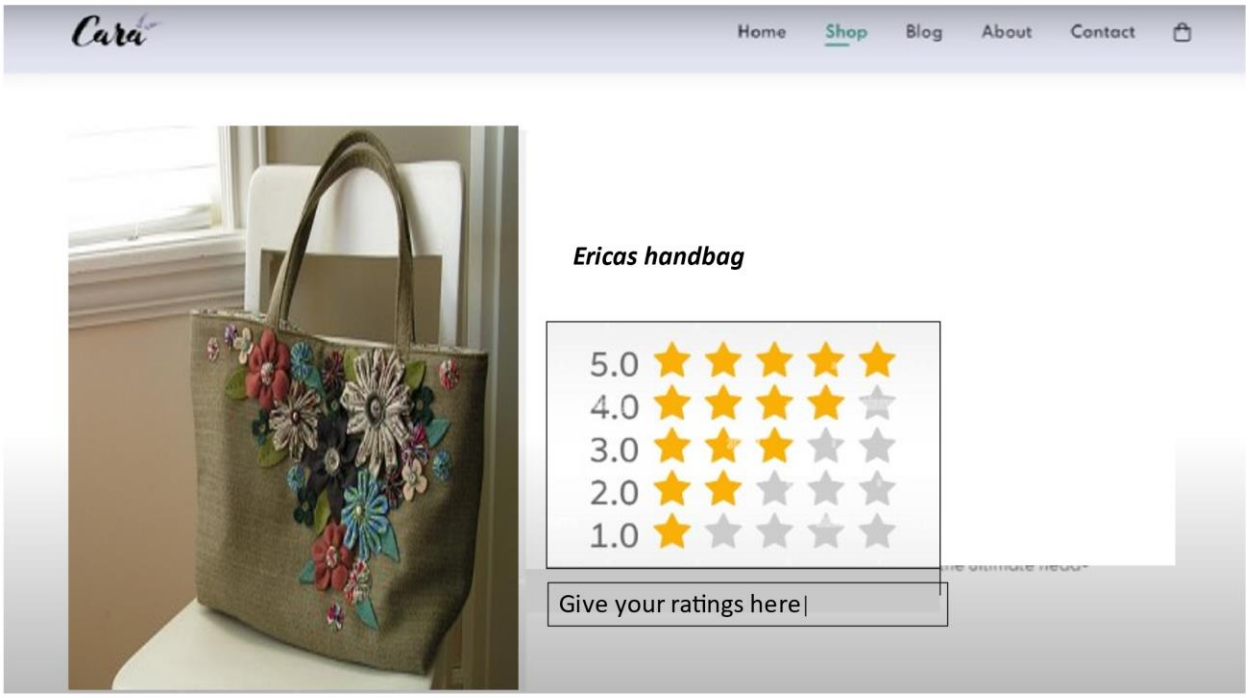
```
  <title>Your E-commerce Website</title>
  <script src="https://js.stripe.com/v3/"></script>
</head>
<body>
  <h1>Checkout</h1>

  <!-- Your product details and form go here -->

  <form id="payment-form">
    <div id="card-element">
      <!-- A Stripe Element will be inserted here. -->
    </div>

    <!-- Used to display form errors. -->
    <div id="card-errors" role="alert"></div>

    <button type="submit">Pay Now</button>
  </form>

  <script>
    var stripe = Stripe('YOUR_PUBLIC_KEY');
```

```
var elements = stripe.elements();

// Create an instance of the card Element.
var card = elements.create('card');

// Add an instance of the card Element into the `card-element`
div.
card.mount('#card-element');

// Handle real-time validation errors from the card Element.
card.addEventListener('change', function(event) {
  var displayError = document.getElementById('card-errors');
  if (event.error) {
    displayError.textContent = event.error.message;
  } else {
    displayError.textContent = '';
  }
});

// Handle form submission.
var form = document.getElementById('payment-form');
```

```
  form.addEventListener('submit', function(event) {
    event.preventDefault();

    stripe.createToken(card).then(function(result) {
      if (result.error) {
        // Inform the user if there was an error.
        var errorElement = document.getElementById('card-errors');
        errorElement.textContent = result.error.message;
      } else {
        // Send the token to your server.
        stripeTokenHandler(result.token);
      }
    });
  });

  function stripeTokenHandler(token) {

  }
 </script>
</body>
</html>
```

Adding ratings to products:

HTML and JavaScript code for adding product reviews on an e-commerce website. In this example, users can provide a review with a rating, and the reviews are displayed on the page.



Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Product Rating</title>
    <style>
      /* Add some basic styling for clarity */
      .rating-container {
        margin-top: 20px;
        font-size: 24px;
      }

      .star {
        cursor: pointer;
        color: gray;
      }

      .star:hover,
      .star.checked {
        color: gold;
      }
    </style>
  </head>
```

```
<body>

  <h1>Product Name</h1>
  <p>Description of the product.</p>

  <h2>Rate This Product</h2>
  <div class="rating-container">
    <span class="star" onclick="setRating(1)">★</span>
    <span class="star" onclick="setRating(2)">★</span>
    <span class="star" onclick="setRating(3)">★</span>
    <span class="star" onclick="setRating(4)">★</span>
    <span class="star" onclick="setRating(5)">★</span>
  </div>

  <p>Your Rating: <span id="selectedRating">0</span>
stars</p>

  <script>
    var selectedRating = 0;

    function setRating(rating) {
```

```
    selectedRating = rating;
    updateRatingDisplay();
  }

  function updateRatingDisplay() {
    var ratingDisplay =
document.getElementById('selectedRating');
    ratingDisplay.textContent = selectedRating;
    highlightStars();
  }

  function highlightStars() {
    var stars = document.querySelectorAll('.star');

    for (var i = 0; i < stars.length; i++) {
      if (i < selectedRating) {
        stars[i].classList.add('checked');
      } else {
        stars[i].classList.remove('checked');
      }
    }
```

```
    }
  </script>
</body>
</html>
```

Conclusion:

thus the part2 of development of ecommerce on cloud foundry is done