**NANMUDHALVAN**

**-** **CAD101 Cloud Application Development - Group 1**

**COLLEGE NAME: JEPPIAAR INSTITUTE OF TECHNOLOGY**

**TITLE: 2106-E-commerece Application on IBM Cloud Foundry**

**TEAM NAME: Proj\_228508\_Team\_2**

**TEAM MEMBERS:**

1. **S.Jagadesh**
2. **S.Ihsan Ahamed**
3. **R.Aravindhan**
4. **S.Guhan**
5. **P.Manoj**

**Phase 5: Project Documentation & Submission**

* In this part you will document your project and prepare it for submission.
* Document the e-commerce platform project and prepare it for submission.

**Project Objectives**:

1. **Establish a Scalable E-commerce Platform**: Creating a robust online shopping platform capable of handling high traffic loads during peak periods.
2. **Ensure Security and Compliance**: Implementing industry-standard security measures, including data encryption, secure payment processing, and compliance with relevant data protection regulations.
3. **Optimize User Experience**: Designing an intuitive and user-friendly interface to enhance the shopping experience, resulting in increased customer satisfaction and retention.
4. **Integrate with IBM Cloud Services**: Leverage IBM Cloud's suite of services for functionalities like AI-powered product recommendations, analytics, and cloud databases to enhance the platform's capabilities.
5. **Implement Multi-Channel Support**: Enable seamless access to the platform across various devices (desktop, mobile, tablet) and browsers to cater to a diverse user base.

**Project Scope**:

Development of a full-featured E-commerce website with user registration, product catalog, shopping cart, order management, and payment processing capabilities.

Integration with IBM Cloud Foundry for hosting, ensuring scalability and high availability.

Implementation of user authentication, authorization, and secure data handling practices.

**Project Deliverables**:

1. Fully functional E-commerce website accessible through web browsers.
2. Secure user authentication and authorization system.
3. Product catalog and search functionality.
4. Shopping cart and checkout process.
5. Integration with IBM Cloud services for enhanced functionality.

**Project Stakeholders**:

* Project Manager
* Development Team
* UX/UI Designers
* Product Owners
* IBM Cloud Consultants
* Quality Assurance Team
* Marketing and Sales Teams

**Project Timeline**:

* Planning and Design Phase: 2 months
* Development and Testing Phase: 4 months
* Deployment and Go-Live: 1 month
* Post-Deployment Support and Optimization: Ongoing

**Budget and Resources**:

* Budget allocation for IBM Cloud services, development tools, and team resources.
* Development team (front-end, back-end, database, UX/UI designers).
* IBM Cloud Consultants for guidance and support.

**Risks and Assumptions**:

* Risks: Potential delays due to unforeseen technical challenges, security vulnerabilities, and external dependencies on third-party services.
* Assumptions: Availability of necessary development tools, stable internet connectivity, and adherence to project timelines.

**Success Criteria**:

* Achieve a minimum of 99.9% platform uptime.
* Process a minimum of 1000 concurrent users during peak traffic.
* Achieve a user satisfaction rating of at least 4.5 out of 5.

**Key Performance Indicators (KPIs)**:

* Conversion Rate (percentage of visitors who make a purchase)
* Average Order Value (AOV)
* Customer Acquisition Cost (CAC)
* Customer Retention Rate
* Website Traffic (number of unique visitors)
* Revenue and Sales Growth

**Project Team**:

* Project Manager
* Front-End Developers (x2)
* Back-End Developers (x2)
* Database Administrator
* UX/UI Designers (x2)
* Quality Assurance Testers (x2)

**Communication Plan**:

* Weekly status meetings with the project team.
* Bi-weekly progress updates to stakeholders.
* Use of project management tools for task tracking and collaboration.

**Change Management Plan**:

* Documented process for handling change requests, including evaluation, approval, and implementation procedures.

**Dependencies and Interdependencies**:

* Integration with IBM Cloud services (e.g., database, AI/ML services).
* Third-party payment gateway integration.
* External partners for product catalog synchronization.

This project overview provides a comprehensive outline of the objectives, scope, resources, and key considerations for deploying an E-commerce platform on IBM Cloud. It serves as a foundation for effective planning and execution.

**Design Thinking Process for an E-commerce Project on IBM Cloud:**

**1. Empathize:**

- Conduct user research through interviews, surveys, and usability tests to deeply understand online shopping behaviors and pain points.

- Gather insights from stakeholders, such as business owners and marketing teams, to grasp their goals and technical requirements for the E-commerce platform.

**2. Define:**

- Formulate a precise problem statement based on the gathered insights, like "Improving customer navigation and purchase efficiency on the current E-commerce platform."

**3. Ideate:**

- Brainstorm creative ideas for enhancing the shopping experience, including features like personalized recommendations and user reviews.

- Explore how IBM Cloud services can be integrated to improve security, scalability, and performance.

**4. Prototype:**

- Create visual representations of the E-commerce platform's user interface using tools like Sketch, Adobe XD, or Figma. These should include features like product listings, search functionality, shopping cart, and the checkout process.

- Develop prototypes to showcase how IBM Cloud services, such as Cloud Foundry and IBM Watson, will be integrated into the platform.

**5. Test:**

- Share prototypes with potential customers for usability and user experience feedback.

- Perform load and performance testing using IBM Cloud tools to ensure the platform can handle high traffic volumes.

**6. Iterate:**

- Refine prototypes based on user feedback and performance testing results, adjusting the user interface and IBM Cloud service configurations as needed.

**7. Implement:**

- Develop the E-commerce platform using chosen technologies and integrate it with IBM Cloud, implementing features like user authentication and secure payment processing.

**8. Evaluate:**

- Continuously monitor the platform's performance using IBM Cloud monitoring tools.

- Gather feedback from users and stakeholders for further improvement opportunities.

**Deployment Phases for E-commerce Platform on IBM Cloud:**

**1. Planning and Discovery:**

- Define business objectives, target audience, and required features.

- Conduct market research to inform platform design.

- Select relevant IBM Cloud services.

**2. Design Phase:**

- Create a clear information architecture, user interface (UI) design, and efficient database schema.

**3. Development:**

- Front-End Development: Implement the UI using technologies like HTML, CSS, and JavaScript.

- Back-End Development: Set up server-side logic and implement key features.

- Database Implementation: Configure and establish relationships between data entities.

**4. Integration with IBM Cloud:**

- Deploy on IBM Cloud Foundry and integrate IBM Cloud services.

**5. Testing:**

- Perform unit testing, integration testing, and user acceptance testing.

- Ensure security and compliance with data protection regulations.

**6. Deployment:**

- Deploy the platform on IBM Cloud Foundry.

**7. Post-Deployment Optimization and Monitoring:**

- Continuously monitor performance and identify areas for optimization.

**8. Launch and Marketing:**

- Plan the official launch and implement marketing and SEO strategies.

**9. Maintenance and Support:**

- Provide ongoing maintenance and customer support.

**Platform Layout**:

Homepage:

**Header**

* Logo and branding
* Navigation menu (Categories, Search, Cart, User Account)
* User authentication and registration options

**Hero Section**

* Promotional banners or featured products
* Call-to-action buttons (e.g., Shop Now)

**Product Categories**

* Grid or list view of product categories
* Featured products from each category

**Featured Products Section**

* Carousel or grid display of highlighted products

**Footer**

* Informational links (About Us, Contact Us, FAQ)
* Social media links
* Newsletter subscription

**Product Listing Page:**

* Breadcrumb Navigation
* Filtering and Sorting Options
* Product Grid
* Product images, names, prices, and quick view options
* Pagination

**Product Detail Page:**

* Product Images
* Product Name, Price, and Description
* Add to Cart Button
* Product Variants (if applicable)
* Customer Reviews and Ratings
* Related Products Section

**Shopping Cart Page:**

* List of Added Products
* Product details, quantity, price, and options to update or remove items
* Cart Summary
* Subtotal, shipping fees, and estimated taxes
* Proceed to Checkout Button

**Checkout Page:**

* Billing and Shipping Information Forms
* Order Summary
* Payment Information
* Order Confirmation

**User Account Dashboard:**

* Order History and Tracking
* Profile Information
* Address Book for Shipping
* Wishlist and Favorites

**Platform Features:**

**User Authentication and Authorization:**

* User registration and login with authentication tokens.
* Forgotten password reset functionality.

**Product Management:**

* Admin interface for adding, editing, and deleting products.
* Categories and tags for organizing products.

**Search and Filtering:**

* Search functionality with filters based on category, price range, and attributes.

**Shopping Cart and Checkout:**

* Persistent shopping cart across sessions.
* Secure checkout process with multiple payment options.

**User Reviews and Ratings:**

* Allow users to rate and review products.
* Display average ratings and reviews on product pages.

**Recommendation Engine:**

* Implement IBM Watson or a custom recommendation system for personalized product suggestions.

**Order Management:**

* Track order status and history for users.
* Admin dashboard for managing orders, processing payments, and handling refunds.

**Security and Compliance:**

* SSL encryption for secure communication.
* Compliance with data protection and privacy regulations.

**Analytics and Reporting:**

* Integration with analytics tools to track user behavior, sales trends, and conversion rates.

**Technical Implementation Details:**

**Front-End:**

* Utilize HTML, CSS, and JavaScript for creating the user interface and adding interactivity.
* Employ dynamic content frameworks like React or Angular.
* Ensure responsiveness to cater to both mobile and desktop users.

**Back-End:**

* Implement server-side scripting with either Node.js or Python.
* Manage data using SQL databases such as PostgreSQL or NoSQL databases like MongoDB.

**Authentication and Authorization:**

* Secure user authentication using JSON Web Tokens (JWT).
* Implement role-based access control (RBAC) for authorization.

**Database and Data Management:**

* Design a normalized database schema to efficiently store data.
* Utilize Object-Relational Mapping (ORM) for database interactions.

**IBM Cloud Integration:**

* Host the platform on IBM Cloud Foundry for scalability and performance.
* Leverage IBM Watson services for AI-powered features, such as product recommendations.

**Security:**

* Follow security best practices to safeguard against common web vulnerabilities like XSS and CSRF.
* Conduct regular security audits and penetration testing.

**Performance Optimization:**

* Enhance loading times through content caching and integration with a Content Delivery Network (CDN).
* Implement load balancing and auto-scaling to handle high traffic loads.

**Monitoring and Analytics:**

* Implement logging and monitoring tools to track performance and handle errors effectively.
* Utilize analytics platforms to analyze user behavior and gain valuable business insights.

**Deployment and Continuous Integration/Continuous Deployment (CI/CD):**

* Employ Git for version control.
* Set up CI/CD pipelines to automate testing and deployment processes.

**Instructions on how to deploy e-commerce platform on IBM Cloud Foundry.**

**Step 1: Set Up IBM Cloud Account**

**1.Sign Up for IBM Cloud:**

* If you don't have an IBM Cloud account, sign up for one at IBM Cloud Sign Up.

**2.Log In to IBM Cloud:**

* Go to IBM Cloud and log in with your credentials.

**Step 2: Prepare Your E-commerce Application**

**1.Codebase:**

* Ensure your E-commerce application is ready for deployment. This should include all the necessary code, dependencies, and configurations.

**2.Cloud Foundry Manifest File:**

* Create a manifest.yml file in the root directory of your application. This file will contain deployment configurations.
* Customize name and memory values based on your application's requirements.

**Step 3: Install IBM Cloud CLI**

**1.Download and Install IBM Cloud CLI:**

* Follow the instructions provided on the IBM Cloud CLI documentation page to install the CLI for your operating system.

**2.Log In to IBM Cloud CLI:**

* Open a terminal/command prompt and run ibmcloud login to log in to your IBM Cloud account.

**Step 4: Deploy Your E-commerce Platform**

**1.Target a Region and Resource Group:**

* Use the following commands to target a region and resource group: ibmcloud target -r <REGION> -g <RESOURCE\_GROUP>
* Replace <REGION> with the desired region (e.g., us-south) and <RESOURCE\_GROUP> with the name of your resource group.

**2.Push Your Application:**

* Navigate to your project directory containing the manifest.yml file and run:ibmcloud cf push
* This command will push your application to IBM Cloud Foundry.

**3.Monitor Deployment:**

* Watch the output for any errors during deployment. If successful, you will receive a URL where your application is hosted.

**Step 5: Access Your E-commerce Platform**

**1.Open the Application:**

* Use the provided URL to access your deployed E-commerce platform.

**2.Test Functionality:**

* Ensure all features and functionalities work as expected in the live environment.

**Step 6: Scale and Manage Your Application**

**1.Scaling Your App:**

* You can scale your application by adjusting the number of instances or memory allocation in your manifest.yml file. Then, run ibmcloud cf push again.

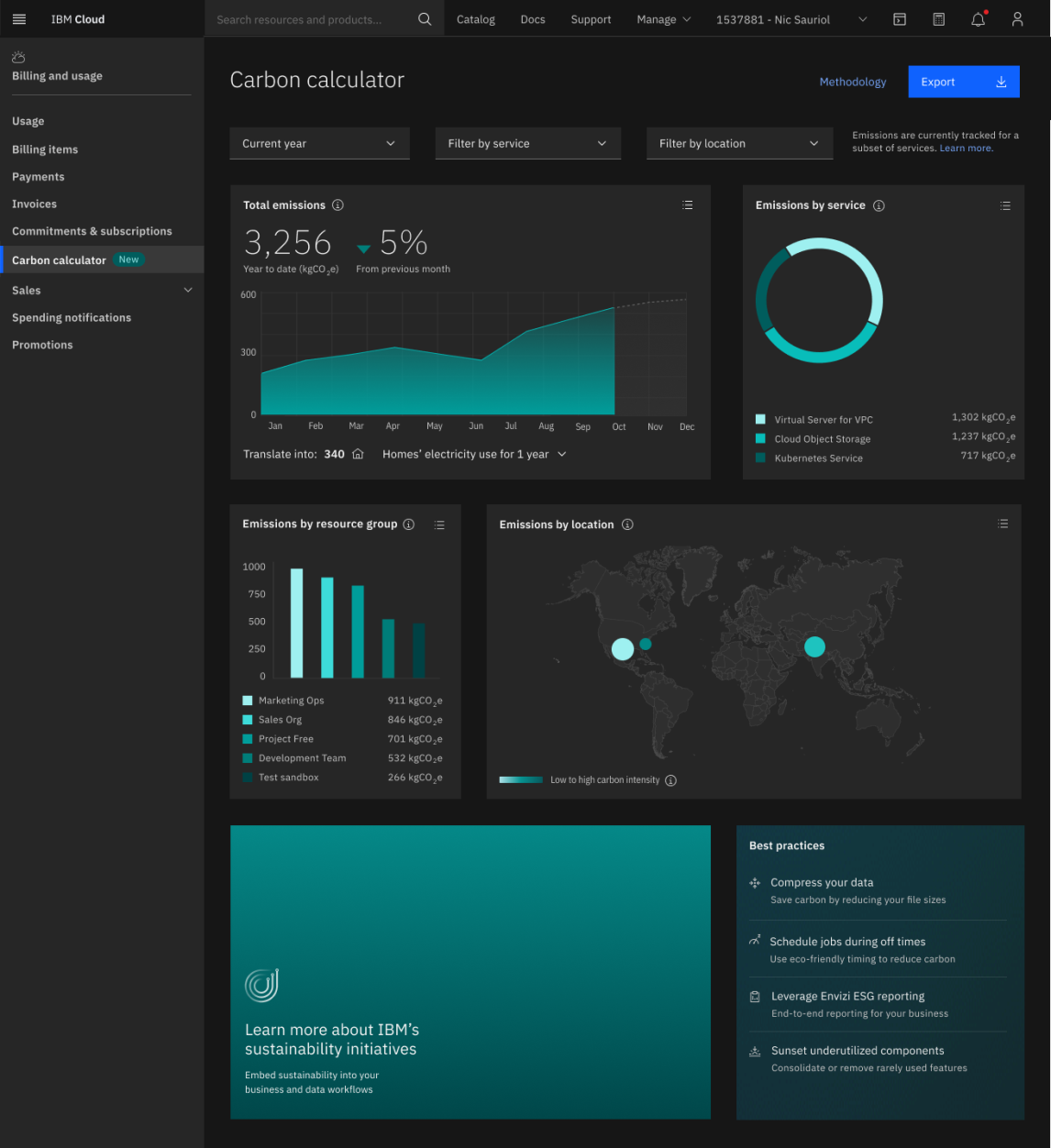
**2.Monitoring and Management:**

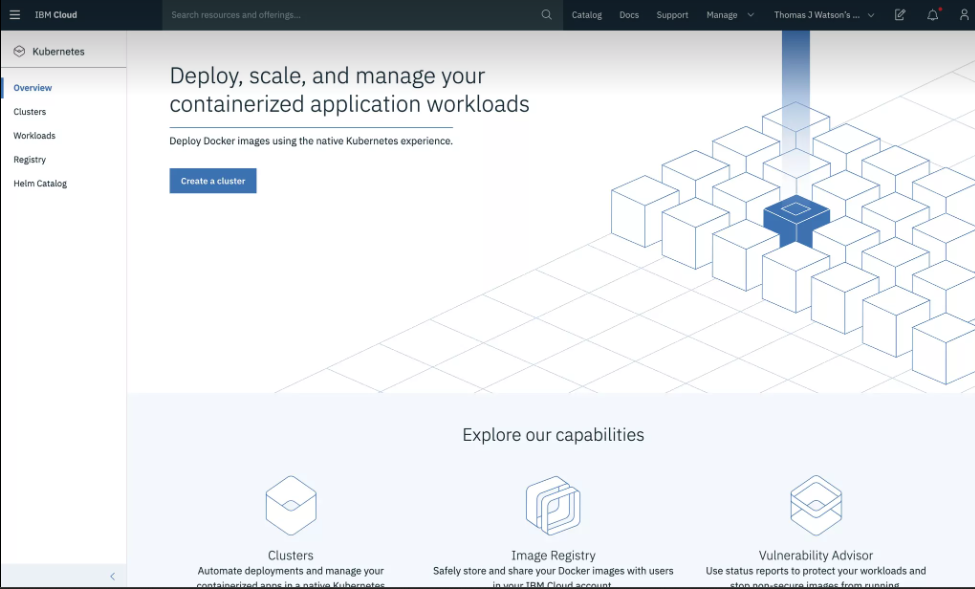
* Utilize IBM Cloud's monitoring and management tools to keep an eye on the performance and health of your application.

**Step 7: Continuous Deployment (Optional)**

**1.Set Up Continuous Deployment:**

* If desired, you can set up CI/CD pipelines for automated testing and deployment. This allows for seamless updates to your E-commerce platform





**Challenges faced by IBM e-commerce**

**Platform Complexity:**

* Understanding the nuances of a cloud platform, including deployment, services, and configurations, can be a formidable task, especially for newcomers to the environment.

**Service Integration:**

* The intricate process of seamlessly integrating various services (e.g., databases, caching, authentication) with the application can be a complex endeavor.

**Scalability and Performance**:

* Crafting the application to effectively handle substantial traffic loads and ensuring efficient scaling, particularly during peak periods, can pose significant challenges.

**Security Concerns:**

* Safeguarding the application against prevalent web vulnerabilities such as SQL injection and cross-site scripting is imperative but often presents complex obstacles.

**Data Management:**

* Effectively managing data, encompassing storage, retrieval, and the handling of user information, products, and orders, is a substantial challenge.

**User Authentication and Authorization:**

* Implementing secure and user-friendly authentication and authorization mechanisms is essential to protect user accounts and sensitive data, but it can be intricate.

**Payment Processing:**

* The integration of a secure and compliant payment gateway while adhering to payment industry standards can be a complex process.

**User Experience (UX) and Interface Design:**

* Crafting an intuitive and visually appealing user interface that delivers a seamless user experience is a significant challenge.

**Cross-Browser Compatibility:**

* Ensuring the application functions flawlessly across diverse web browsers and devices can be challenging due to varying levels of compatibility.

**Regulatory Compliance:**

* Complying with legal and regulatory requirements, particularly those related to data privacy and security (e.g., GDPR, HIPAA), can be intricate and time-consuming.

**Testing and Quality Assurance:**

* Conducting thorough testing for functionality, security, and performance is essential, and establishing automated testing suites can be a daunting task.

**Documentation and Knowledge Transfer:**

* Maintaining comprehensive documentation that covers the application's architecture, deployment process, and codebase is often overlooked but remains pivotal for future development and maintenance.

**Coding**

Creating a complete E-commerce application involves a significant amount of code and would require a detailed project with specific technologies and frameworks. However, I can provide you with a simple example using a Python-based web framework (Flask) and a SQLite database. This is a very basic starting point and should be further developed according to your specific requirements.

Dataset link: https://archive.ics.uci.edu/dataset/352/online+retail

**Create a basic Flask app:**

from flask import Flask, render\_template

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

@app.route('/')

def index():

return render\_template('index.html')

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

app.run(debug=True)

**Create a basic HTML template (templates/index.html):**

<!DOCTYPE html>

<html>

<head>

<title>Simple E-commerce App</title>

</head>

<body>

<h1>Welcome to our E-commerce Store</h1>

<ul>

<li>Product 1 - $10</li>

<li>Product 2 - $20</li>

<li>Product 3 - $30</li>

</ul>

</body>

</html>

**Set up your database (SQLite in this case):**

import sqlite3

def create\_database():

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

c = conn.cursor()

c.execute('''CREATE TABLE products

(id INTEGER PRIMARY KEY, name TEXT, price REAL)''')

c.execute("INSERT INTO products (name, price) VALUES (?, ?)", ('Product 1', 10))

c.execute("INSERT INTO products (name, price) VALUES (?, ?)", ('Product 2', 20))

c.execute("INSERT INTO products (name, price) VALUES (?, ?)", ('Product 3', 30))

conn.commit()

conn.close()

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

create\_database()

**Run your Flask application:**

python app.py

**User Authentication:**

# Install Flask-Login

pip install Flask-Login

from flask import Flask, render\_template, redirect, url\_for, request

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

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

app.secret\_key = 'your\_secret\_key'

# Initialize Flask-Login

login\_manager = LoginManager()

login\_manager.init\_app(app)

# Simple user model

class User(UserMixin):

def \_\_init\_\_(self, id):

self.id = id

# Example user data

users = {'user1': {'password': 'password1'}}

@login\_manager.user\_loader

def load\_user(user\_id):

return User(user\_id)

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if users.get(username) and users[username]['password'] == password:

user = User(username)

login\_user(user)

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

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

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

@app.route('/dashboard')

@login\_required

def dashboard():

return f'Welcome, {current\_user.id}! This is your dashboard.'

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

app.run(debug=True)

**Shopping Cart Functionality:**

# Assuming you have a product database

products = {

1: {'name': 'Product 1', 'price': 10},

2: {'name': 'Product 2', 'price': 20},

3: {'name': 'Product 3', 'price': 30}

}

# Inside your Flask routes

@app.route('/add\_to\_cart/<int:product\_id>')

@login\_required

def add\_to\_cart(product\_id):

if 'cart' not in session:

session['cart'] = []

session['cart'].append(product\_id)

return redirect(url\_for('view\_cart'))

@app.route('/view\_cart')

@login\_required

def view\_cart():

cart\_contents = [products[product\_id] for product\_id in session.get('cart', [])]

total\_price = sum(item['price'] for item in cart\_contents)

return render\_template('cart.html', cart\_contents=cart\_contents, total\_price=total\_price)

**Payment Processing:**

# Assuming you have installed the Stripe library

pip install stripe

# Inside your Flask routes

import stripe

stripe.api\_key = 'your\_stripe\_secret\_key'

@app.route('/checkout', methods=['POST'])

@login\_required

def checkout():

cart\_contents = [products[product\_id] for product\_id in session.get('cart', [])]

total\_price = sum(item['price'] for item in cart\_contents)

# Create a payment intent with Stripe

intent = stripe.PaymentIntent.create(

amount=int(total\_price \* 100),

currency='usd',

payment\_method=request.form['payment\_method']

)

return render\_template('checkout.html', client\_secret=intent.client\_secret)

**Loading the Dataset**

import pandas as pd

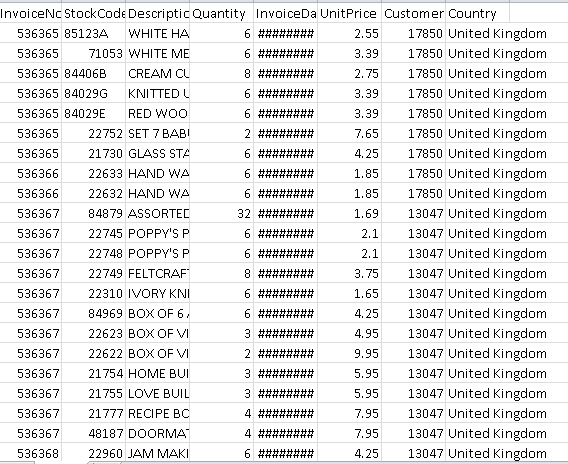
# Load the dataset

df = pd.read\_csv('ecommerce\_data.csv')

# Print the first few rows to verify the data has been loaded correctly

print(df.head())

**DATASET**

****

**Preprocessing**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

# Assuming 'df' is your DataFrame

# Step 1: Handling Missing Values

df.fillna(df.mean(), inplace=True)

# Step 2: Handling Categorical Data (One-Hot Encoding)

df = pd.get\_dummies(df, columns=['categorical\_column'])

# Step 3: Feature Scaling

scaler = StandardScaler()

df[['numerical\_column1', 'numerical\_column2']] = scaler.fit\_transform(df[['numerical\_column1', 'numerical\_column2']])

# Step 8: Splitting Data

X = df.drop('target\_variable', axis=1)

y = df['target\_variable']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**USER AUTHENTICATION**

pip install Flask Flask-Login

**IMPLEMENTATON OF USER AUTHENTICATION**

from flask import Flask, render\_template, redirect, url\_for, request

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

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

app.secret\_key = 'your\_secret\_key' # Change this to a strong, random secret key

login\_manager = LoginManager()

login\_manager.init\_app(app)

# Mock user data for demonstration

users = {'user1': {'password': 'password1'}, 'user2': {'password': 'password2'}}

class User(UserMixin):

def \_\_init\_\_(self, id):

self.id = id

@login\_manager.user\_loader

def load\_user(user\_id):

return User(user\_id)

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if users.get(username) and users[username]['password'] == password:

user = User(username)

login\_user(user)

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

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

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

@app.route('/dashboard')

@login\_required

def dashboard():

return f'Welcome, {current\_user.id}! This is your dashboard.'

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

app.run(debug=True)

**SHOPPING CART**

**Setup flask application**

from flask import Flask, render\_template, request, session

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

app.secret\_key = 'your\_secret\_key' # Change this to a strong, random secret key

# Sample products (for demonstration)

products = [

{'id': 1, 'name': 'Product 1', 'price': 10},

{'id': 2, 'name': 'Product 2', 'price': 20},

{'id': 3, 'name': 'Product 3', 'price': 30}

]

@app.route('/')

def index():

return render\_template('index.html', products=products)

@app.route('/add\_to\_cart', methods=['POST'])

def add\_to\_cart():

product\_id = int(request.form['product\_id'])

product = next((p for p in products if p['id'] == product\_id), None)

if product:

cart = session.get('cart', [])

cart.append(product)

session['cart'] = cart

return redirect('/')

@app.route('/cart')

def view\_cart():

cart = session.get('cart', [])

total\_price = sum(item['price'] for item in cart)

return render\_template('cart.html', cart=cart, total\_price=total\_price)

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

app.run(debug=True)

**create html template**

**1.index.html**

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

</head>

<body>

<h1>Products</h1>

<ul>

{% for product in products %}

<li>

{{ product.name }} - ${{ product.price }}

<form action="/add\_to\_cart" method="post">

<input type="hidden" name="product\_id" value="{{ product.id }}">

<input type="submit" value="Add to Cart">

</form>

</li>

{% endfor %}

</ul>

</body>

</html>

**2.cart.html**

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

</head>

<body>

<h1>Shopping Cart</h1>

<ul>

{% for item in cart %}

<li>{{ item.name }} - ${{ item.price }}</li>

{% endfor %}

</ul>

<p>Total Price: ${{ total\_price }}</p>

</body>

</html>

**CHECKOUT FUNCTIONALITIES**

**update Flask app**

from flask import Flask, render\_template, request, session, redirect, url\_for

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

app.secret\_key = 'your\_secret\_key'

# Mock products for demonstration

products = [

{'id': 1, 'name': 'Product 1', 'price': 10},

{'id': 2, 'name': 'Product 2', 'price': 20},

{'id': 3, 'name': 'Product 3', 'price': 30}

]

@app.route('/')

def index():

return render\_template('index.html', products=products)

@app.route('/add\_to\_cart', methods=['POST'])

def add\_to\_cart():

product\_id = int(request.form['product\_id'])

product = next((p for p in products if p['id'] == product\_id), None)

if product:

cart = session.get('cart', [])

cart.append(product)

session['cart'] = cart

return redirect('/')

@app.route('/cart')

def view\_cart():

cart = session.get('cart', [])

total\_price = sum(item['price'] for item in cart)

return render\_template('cart.html', cart=cart, total\_price=total\_price)

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

def checkout():

if request.method == 'POST':

# Process payment (mocked)

cart = session.get('cart', [])

total\_price = sum(item['price'] for item in cart)

session.pop('cart', None) # Clear the cart after checkout

return render\_template('confirmation.html', total\_price=total\_price)

return render\_template('checkout.html')

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

app.run(debug=True)

**create html templates**

**(i)checkout.html**

<!DOCTYPE html>

<html>

<head>

<title>Checkout</title>

</head>

<body>

<h1>Checkout</h1>

<form action="/checkout" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br><br>

<label for="address">Address:</label>

<input type="text" id="address" name="address" required><br><br>

<input type="submit" value="Submit Order">

</form>

</body>

</html>

**(ii)confirmation.html**

<!DOCTYPE html>

<html>

<head>

<title>Order Confirmation</title>

</head>

<body>

<h1>Order Confirmation</h1>

<p>Your order has been placed successfully!</p>

<p>Total Price: ${{ total\_price }}</p>

</body>

</html>

**USER REGISTRATION(BACK END)**

**Update Flask app**

from flask import Flask, render\_template, request, redirect, url\_for, flash

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

app.secret\_key = 'your\_secret\_key'

# Mock user data (for demonstration)

users = {}

@app.route('/')

def index():

return render\_template('index.html')

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

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if username in users:

flash('Username already exists. Please choose a different username.', 'error')

else:

users[username] = {'password': password}

flash('Registration successful! Please log in.', 'success')

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

return render\_template('register.html')

@app.route('/login')

def login():

return render\_template('login.html')

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

app.run(debug=True)

**create html template**

**(i)register.html**

<!DOCTYPE html>

<html>

<head>

<title>Register</title>

</head>

<body>

<h1>Register</h1>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul>

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

<form action="/register" method="post">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br><br>

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

</form>

</body>

</html>

**(iii)login.html**

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

</head>

<body>

<h1>Login</h1>

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

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br><br>

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

</form>

</body>

</html>

**AUTHENTICATIN(BACKEND)**

**Update Flask app**

from flask import Flask, render\_template, request, redirect, url\_for, flash

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

app.secret\_key = 'your\_secret\_key'

# Mock user data (for demonstration)

users = {'user1': {'password': 'password1'}, 'user2': {'password': 'password2'}}

@app.route('/')

def index():

return render\_template('index.html')

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

def login():

username = request.form['username']

password = request.form['password']

if users.get(username) and users[username]['password'] == password:

flash('Login successful!', 'success')

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

else:

flash('Invalid username or password. Please try again.', 'error')

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

@app.route('/dashboard')

def dashboard():

return render\_template('dashboard.html')

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

app.run(debug=True)

**create html template**

**(i)index.html**

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

</head>

<body>

<h1>Login</h1>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul>

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

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

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br><br>

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

</form>

</body>

</html>

**(ii)dashboard.html**

<!DOCTYPE html>

<html>

<head>

<title>Dashboard</title>

</head>

<body>

<h1>Welcome to the Dashboard</h1>

<p>This is a protected page.</p>

</body>

</html>

**SHOPPING CART FUNCTIONALITIES,TOTAL,SMOOTH CHECKOUT PROCESS**

**Update Flask app**

from flask import Flask, render\_template, request, redirect, url\_for, flash

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

app.secret\_key = 'your\_secret\_key'

# Mock products for demonstration

products = [

{'id': 1, 'name': 'Product 1', 'price': 10},

{'id': 2, 'name': 'Product 2', 'price': 20},

{'id': 3, 'name': 'Product 3', 'price': 30}

]

@app.route('/')

def index():

return render\_template('index.html', products=products)

@app.route('/add\_to\_cart', methods=['POST'])

def add\_to\_cart():

product\_id = int(request.form['product\_id'])

product = next((p for p in products if p['id'] == product\_id), None)

if product:

cart = session.get('cart', [])

cart.append(product)

session['cart'] = cart

return redirect('/')

@app.route('/cart')

def view\_cart():

cart = session.get('cart', [])

total\_price = sum(item['price'] for item in cart)

return render\_template('cart.html', cart=cart, total\_price=total\_price)

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

def checkout():

if request.method == 'POST':

# Process payment (mocked)

cart = session.get('cart', [])

total\_price = sum(item['price'] for item in cart)

session.pop('cart', None) # Clear the cart after checkout

flash('Order placed successfully! Total: $' + str(total\_price), 'success')

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

return render\_template('checkout.html')

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

app.run(debug=True)

**create html template**

**(i)index.html**

<!-- ... -->

{% for product in products %}

<li>

{{ product.name }} - ${{ product.price }}

<form action="/add\_to\_cart" method="post">

<input type="hidden" name="product\_id" value="{{ product.id }}">

<input type="submit" value="Add to Cart">

</form>

</li>

{% endfor %}

<!-- ... -->

**(ii)cart.html**

<!-- ... -->

<h2>Shopping Cart</h2>

<ul>

{% for item in cart %}

<li>{{ item.name }} - ${{ item.price }}</li>

{% endfor %}

</ul>

<p>Total Price: ${{ total\_price }}</p>

<form action="/checkout" method="post">

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

</form>

<!-- ... -->

**(iii)checkout.html**

<!-- ... -->

<h2>Checkout</h2>

<form action="/checkout" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br><br>

<label for="address">Address:</label>

<input type="text" id="address" name="address" required><br><br>

<input type="submit" value="Submit Order">

</form>

<!-- ... -->