E-Commerce Web Application

Motive : Selling Products

Software Requirements :

Front-end :

HTML, CSS, JavaScript

Back-end :

MySQL

Execution Server :

IBM Cloud

Hardware Requirements :

- Hard Disk(minimum 128GB)
- RAM 2GB
- HD Display

Descriptions

Planning and Research:

Start by defining your business goals, conducting market research, analyzing competitors, identifying your target audience, selecting the technology stack, and establishing a budget and timeline.

Front-End Development:

Focus on the user interface design, responsive layout, navigation, product listings, shopping cart, user registration/login, and ensuring an excellent user experience (UX).

Back-End Development:

Set up the server, design the database, implement user authentication, manage products and orders, integrate payment gateways, and prioritize security.

Database:

Create and manage user data, product data, order data, cart data, reviews, ratings, and inventory information.

Payment and Shipping:

Integrate payment gateways, define shipping methods and providers, validate addresses, and implement order tracking.

Testing:

Perform unit testing, integration testing, user acceptance testing, performance testing, and security testing to ensure the application's reliability.

Deployment:

Set up hosting, deployment environments, CI/CD pipelines, load balancing, and plan for scalability.

Marketing and SEO:

Optimize the website for search engines (SEO), integrate social media, plan email marketing campaigns, create a content strategy, and set up analytics.

Maintenance and Support:

Handle bug fixes, implement updates and enhancements, monitor server health, perform backups and recovery, and provide customer support.

Documentation and Training:

Create comprehensive documentation for developers, users, and administrators, along with training materials.

User-Centric Design:

Prioritize user experience by understanding the needs and preferences of your target audience.

Conduct user research to identify pain points in the current e-commerce design.

Personalization:

Use data analytics to personalize the user journey, providing tailored product recommendations and content.

Implement dynamic pricing strategies based on user behavior and preferences.

Mobile Optimization:

Ensure that the e-commerce platform is optimized for mobile devices, as a significant portion of users shop using smartphones.

Consider progressive web apps (PWAs) for a seamless mobile experience.

Social Commerce Integration:

Leverage social media platforms for e-commerce activities, allowing users to make purchases directly through social channels.

Integrate social elements into the e-commerce platform, such as user reviews and social sharing features.

Sustainability and Ethical Practices:

Incorporate sustainable and eco-friendly practices into the e-commerce design.

Communicate transparently about the sourcing and manufacturing processes of products.

Agile Development:

Adopt agile development methodologies to quickly iterate and respond to changing market trends and user feedback.

Implement A/B testing to optimize various elements of the e-commerce platform.

Cybersecurity Measures:

Prioritize the security of user data and financial transactions.

Stay updated on the latest cybersecurity trends and implement robust security measures to build trust with users

Multiple payment methods

Credit/Debit Cards: The most common form of online payment. Visa, MasterCard, American Express, and others are widely accepted.

Digital Wallets: Services like PayPal, Apple Pay, Google Pay, and Samsung Pay allow users to store card information securely and make payments using their mobile devices.

Bank Transfers: Direct transfers from a bank account to the seller's account.

Cryptocurrencies: Some businesses accept cryptocurrencies like Bitcoin, Ethereum, or others as a form of payment.

Cash on Delivery (COD): Payment is made when the product is delivered, commonly used in e-commerce.

Installment Plans: Allows customers to pay in fixed installments over time.

E-checks: Electronic checks that are processed online.

Prepaid Cards: Cards that are loaded with a specific amount of money in advance.

Mobile Payments: Payments made using mobile apps, often linked to a bank account or credit card.

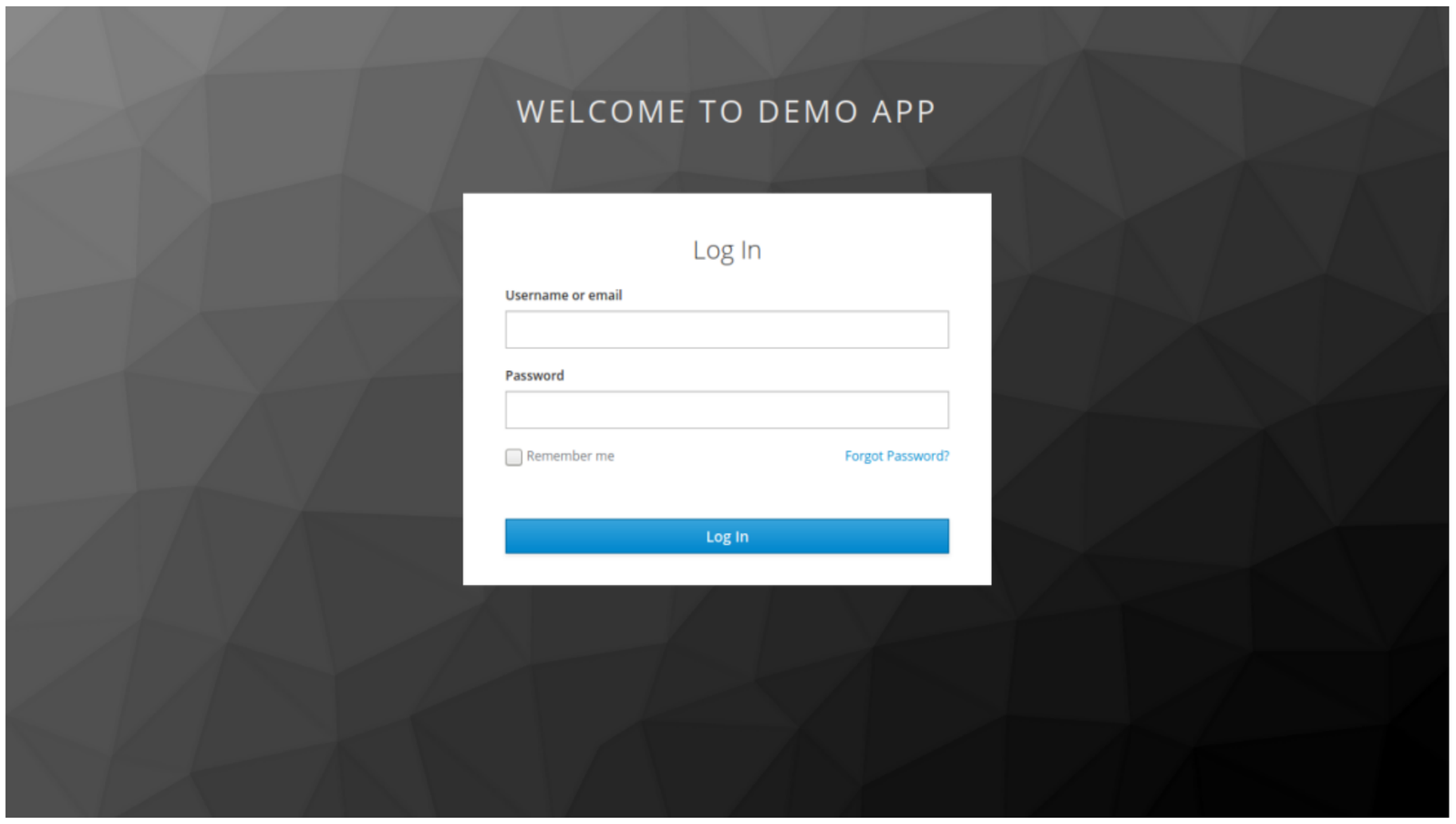
Contactless Payments: Using technologies like NFC to make secure, quick payments without physical contact.

Collaboration with Startups:

Consider partnerships with startups and tech innovators to bring fresh ideas and technologies to your e-commerce platform.

Explore collaborations with fintech companies for innovative payment solutions.

Page 1 design:



DATABASE TO STORE PRODUCT INFORMATION

```
from flask import Flask, jsonify

import ibm_db

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)

app = Flask(__name)

# Define your database connection parameters
```

```
db_credentials = {
 'db2_driver': '{IBM DB2 ODBC DRIVER}',
 'db2_database': 'YOUR_DB_NAME',
 'db2_hostname': 'YOUR_DB_HOST',
 'db2_port': '50000',
 'db2_protocol': 'TCPIP',
 'db2_uid': 'YOUR_DB_USERNAME',
 'db2_pwd': 'YOUR_DB_PASSWORD'
}
# Create a connection string
conn_str = (
 "DRIVER={};"
 "DATABASE={};"
 "HOSTNAME={};"
 "PORT={};"
 "PROTOCOL={};"
 "UID={};"
 "PWD={};"
).format(
 db_credentials['db2_driver'],
 db_credentials['db2_database'],
 db_credentials['db2_hostname'],
 db_credentials['db2_port'],
 db_credentials['db2_protocol'],
 db_credentials['db2_uid'],
 db_credentials['db2_pwd']
```

)

Route to retrieve data from the database

@app.route('/get_data')

def get_data():

conn = ibm_db.connect(conn_str, '', '')

stmt = ibm_db.exec_immediate(conn, "SELECT * FROM your_table_name")

data = ibm_db.fetch_both(stmt)

result = []

while data:

result.append(data)

data = ibm_db.fetch_both(stmt)

ibm_db.close(conn)

return jsonify(result)

if __name__ == '__main__':

app.run(debug=True)

CODE EXPLANATION :

This code is a Python script that uses the Flask web framework to create a web application with two

routes:

The main route '/' renders an HTML template called 'index.html'.

Another route '/get_data' retrieves data from an IBM Db2 database and returns it as JSON.

HTML CODE:

</html><!DOCTYPE html>

<html>

<head> <title>Men's Fashion</title>

<meta charset="utf-8"></head>

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
 <body><div></br><div>
 <link rel="stylesheet" href="Styles.css">
 <nav>
 <label class="dot">Men's Fashion</label>
 <ul><li><a href="#Home" window="new">Home</a></li>
 <li><a href="#Contact">Contact </a></li>
 <li><a href="#Cart">Cart</a></li>
 <li><a href="#Payment">Payment</a></li>
 </ul>
 </nav>
<a class="vet" href="default.asp">
<img src="boys.jpg" alt="HTML tutorial" style="width:175px;height:150px;">
<p>Round neck T-shirt<br>499.Rs</p></a>
<a class="vet" href="default.asp">
 <img src="boys2.jpg" alt="HTML tutorial" style="width:175px;height:150px;">
<p>Round neck T-shirt<br>499.Rs</p></a>
<a class="vet" href="default.asp">
 <img src="boys3.jpg"alt="HTML tutorial" style="width:175px;height:150px;">
<p>Round neck T-shirt<br>499.Rs</p></a>
<a class="vet" href="default.asp">
 <img src="boys4.jpg" alt="HTML tutorial" style="width:175px;height:150px;">
<p>Round neck T-shirt<br>499.Rs</p></a>
<a class="vet" href="default.asp">
 <img src="boys5.jpg" alt="HTML tutorial" style="width:175px;height:150px;">
<p>Round neck T-shirt<br>499.Rs</p></a>
```

```
<a class="vet" href="default.asp">

 <img src="boys6.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

 <img src="boys7.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

 <img src="boys8.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

 <img src="boys9.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys10.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys11.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys12.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys13.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys14.jpg" alt="HTML tutorial" style="width:175px;height:150px;">
```

```
<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys15.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys16.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys17.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys18.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys19.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

<a class="vet" href="default.asp">

<img src="boys20.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Round neck T-shirt<br>499.Rs</p></a>

</body>

</html>
```

CSS CODE:

```
*{

 margin: 0px;

 padding:0px;

 text-decoration: none;
```

```
 list-style: none;
 box-sizing: border-box;
}
body{
 font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}
nav{
 background: #0082e6;
 height: 80px;
 width: auto;
}
label.dot{
 color: white;
 font-size: 25px;
 line-height: 80px;
 padding: 0 100px;
 font-weight: bold;
}
nav ul{
 float:right;
 margin-right: 20px;
}
nav ul li{
 display: inline-block;
 line-height: 80px;
 margin:0 5px;
```

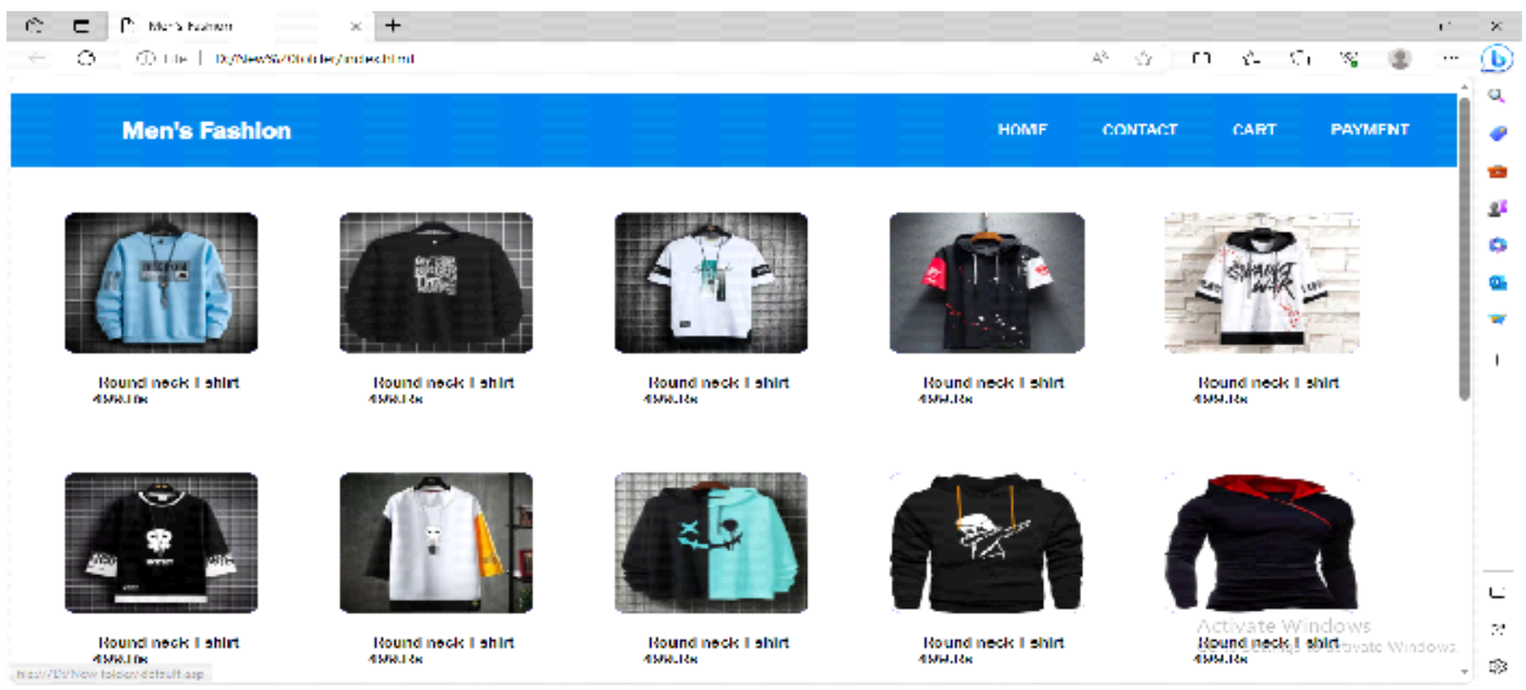
```
}
nav ul li a{
 color:white;
 font-size: 17px;
 padding: 8px 18px ;
 text-transform: uppercase;
 border-radius: 4px;
}
.vet{
margin-left:48px;
display:flex;
flex-wrap:wrap;
float:left;
width:200px;
height:280px;
text-decoration:none;
text-align:left;
color:black;
padding-top: 50px;
}
p{
border: 1px ;
box-sizing: border-box;
width:350px;
padding-inline:25px;
text-indent:5px;
```

```
}
img{
background-color:blue;
border-radius:10px;
}
*{
 margin: 0px;
 padding:0px;
 text-decoration: none;
 list-style: none;
 box-sizing: border-box;
}
body{
 font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}
nav{
 background: #0082e6;
 height: 80px;
 width: auto;
}
label.dot{
color: white;
font-size: 25px;
line-height: 80px;
padding: 0 100px;
font-weight: bold;
```

```
}
nav ul{
 float:right;
 margin-right: 20px;
}
nav ul li{
 display: inline-block;
 line-height: 80px;
 margin:0 5px;
}
nav ul li a{
 color:white;
 font-size: 17px;
 padding: 8px 18px ;
 text-transform: uppercase;
 border-radius: 4px;
}
.vet{
margin-left:48px;
display:flex;
flex-wrap:wrap;
float:left;
width:200px;
height:280px;
text-decoration:none;
text-align:left;
```

```
color:black;
padding-top: 50px;
}
p{
 border: 1px ;
 box-sizing: border-box;
 width:350px;
 padding-inline:25px;
 text-indent:5px;
}
img{
background-color:blue;
border-radius:10px;
}
```

E-COMMERCE APP OUTLOOK:



1. Setup and Configuration:

Set up your development environment by creating a new directory for your project and initializing a virtual environment.

Install Flask and any necessary extensions, e.g., Flask-WTF for forms.

Configure your Flask application and define your secret key and database

connection.

2. User Registration and Authentication:

Create data models for users, products, and orders. You can use SQLAlchemy as an ORM.

Implement user registration and login forms using Flask-WTF.

Create routes and views for user registration, login, and logout.

Implement secure password hashing (e.g., bcrypt) for user passwords.

Use Flask-Login to manage user sessions.

3. Product Listings:

Set up a product database (in this example, you can use Python dictionaries).

Create routes to display product listings.

4. Shopping Cart:

Implement a shopping cart mechanism using session storage.

Allow users to add/remove items from the cart.

Calculate and display the total cost of items in the cart.

5. Checkout Process:

Create a checkout form using Flask-WTF.

Implement a route and view for the checkout process.

Verify user authentication before proceeding with the checkout.

For a simplified example, simulate the payment process.

6. HTML Templates:

Create HTML templates for user registration, login, product listings, shopping cart, and checkout pages using Jinja2 templating.

7. Styling and Frontend:

Apply CSS styles to your HTML templates for a better user interface.

Optionally, use JavaScript to enhance the client-side interactivity (e.g., updating the shopping cart without page reload).

8. Testing:

Thoroughly test your e-commerce platform's functionality.

Consider using testing frameworks like pytest for automated testing.

9. Deployment:

Deploy your e-commerce platform on a web server or cloud hosting service (e.g., Heroku, AWS, DigitalOcean).

10. Security and Data Storage:

Enhance security with secure password hashing (e.g., bcrypt).

Implement security measures for payments, such as SSL encryption and integrating with real payment gateways.

Replace the in-memory data store with a real database (e.g., PostgreSQL, MySQL) for production use.

11. Scaling and Additional Features:

As your e-commerce platform grows, consider adding features like user profiles, order history, product reviews, and product search.

Please note that this is a simplified guide, and building a fully functional e-commerce platform involves a lot of work. It's essential to stay updated on best practices for security and user data protection. Additionally, if your e-commerce platform is going to handle real payments, it's advisable to engage with professionals for payment processing and security audits.

HTML code:login

```
 <!DOCTYPE html>
<html>
```

```
<head>
 <title>Login</title>
</head>
<body>
 <h1>Please Log In</h1>
 <form method="POST" action="/login">
 <label for="username">Username:</label>
 <input type="text" name="username" id="username" required>
 <br>
 <label for="password">Password:</label>
 <input type="password" name="password" id="password" required>
 <br>
 <input type="submit" value="Log In">
 </form>
</body>
</html>
```

HTML code:Register

```
<!DOCTYPE html>
<html>
<head>
 <title>Register</title>
</head>
<body>
 <h1>Register for an Account</h1>
 <form method="POST" action="/register">
```

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

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

<br>

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

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

 <br>

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

 </form>

</body>

</html>
```

HTML code:Cart

```
<!DOCTYPE html>

<html>

<head>

 <title>Shopping Cart</title>

</head>

<body>

 <h1>Shopping Cart</h1>

 <ul>

 {% for item in cart %}

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

 {% endfor %}

 </ul>

 <a href="checkout.html">Checkout</a>

</body>
```

```
</html>

JAVASCRIPT:

const express = require('express');

const session = require('express-session');

const bodyParser = require('body-parser');

const app = express();

const port = 3000;

// Use sessions for tracking user login state

app.use(session({

 secret: 'your-secret-key',

 resave: true,

 saveUninitialized: false,

}));

app.use(bodyParser.urlencoded({ extended: true }));

// In-memory user and product data (for demonstration)

const users = [];

const products = [

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

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

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

];

// Middleware to check if the user is logged in

const requireLogin = (req, res, next) => {

 if (!req.session.user) {

 res.redirect('/login');
```

```
 } else {
 next();
 }
};
// Home page
app.get('/', (req, res) => {
 res.send('Welcome to the e-commerce platform');
});
// User registration
app.get('/register', (req, res) => {
 res.send('Register for an account');
});
app.post('/register', (req, res) => {
 const { username, password } = req.body;
 users.push({ username, password });
 res.redirect('/login');
});
// User login
app.get('/login', (req, res) => {
 res.send('Please log in');
});
app.post('/login', (req, res) => {
 const { username, password } = req.body;
 const user = users.find((u) => u.username === username && u.password ===
password);
```

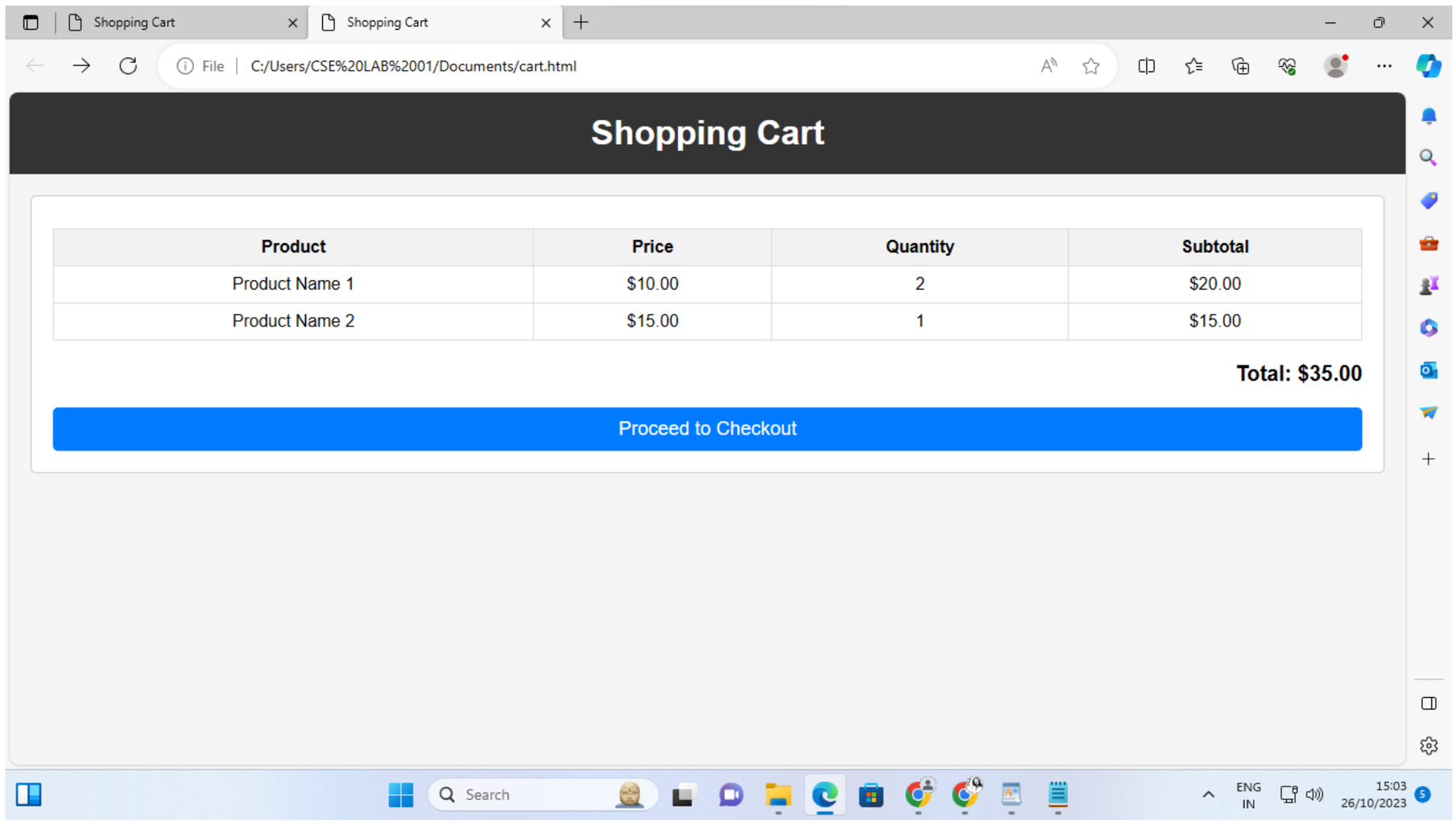
```
if (user) {
req.session.user = user;
res.redirect('/dashboard');
} else {
res.send('Login failed. Check your username and password.');
 }
});
// User dashboard
app.get('/dashboard', requireLogin, (req, res) => {
 res.send(`Welcome, ${req.session.user.username}! This is your dashboard.`);
});
// Product listings
app.get('/products', (req, res) => {
 res.send('Product listings:\n' + products.map(p => `${p.name} -
$${p.price}`).join('\n'));
});
// Shopping cart (simplified)
const cart = [];
app.get('/cart', requireLogin, (req, res) => {
 res.send('Shopping Cart:\n' + cart.map(item => `${item.product.name} -
$${item.product.price}`).join('\n'));
});
app.post('/cart/add/:productId', requireLogin, (req, res) => {
 const productId = parseInt(req.params.productId, 10);
 const product = products.find(p => p.id === productId);
```

```
if (product) {
cart.push({ product });
res.redirect('/cart');
} else {
res.send('Product not found.');
}});
// Checkout (simplified)
app.get('/checkout', requireLogin, (req, res) => {
 const total = cart.reduce((acc, item) => acc + item.product.price, 0);
 res.send(`Checkout - Total: $${total}`);
});
app.listen(port, () => {
 console.log(`Server is running on port ${port}`);});});
```

DATABASE:

```
CREATE DATABASE e_commerce;
CREATE USER e_commerce_user WITH PASSWORD 'your_password';
GRANT CONNECT ON DATABASE TO USER e_commerce_user;
GRANT CREATEIN ON SCHEMA PUBLIC TO USER e_commerce_user;
GRANT IMPLICIT_SCHEMA ON DATABASE TO USER e_commerce_user
```

OUTLOOK:


Shopping Cart
Shopping Cart
File | C:/Users/CSE%20LAB%2001/Documents/cart.html
Shopping Cart
Product
Price
Quantity
Subtotal
Product Name 1
$10.00
2
$20.00
Product Name 2
$15.00
1
$15.00
Total: $35.00
Proceed to Checkout
Search
ENG
IN
15:03
26/10/2023