**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.

**Cyber Security Measures:**

Prioritize the security of user data and financial transactions.

Stay updated on the latest cyber security 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.

**Crypto currencies:** Some businesses accept crypto currencies like Bit coin , 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.

**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:**

<!DOCTYPE html>

<head><title>Chocalate </title>

<link rel="icon" type="icon" href="chocolate.png"></head>

<link rel="stylesheet" href="name.css"></style>

<html>

<body >

<nav>

<label class="dot">Ancro Chocolates</label>

<ul>

<li><a href="#Home">Home</a></li>

<li><a href="#All categories">All categories</a></li>

<li><a href="Orders.html">My Orders</a></li>

<li><a href="My Cart.html">My Cart</a></li>

</ul>

</nav>

<div class="box">

<a class="vet" href="chocs1.html">

<img src="chocs1.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Mahalacto<br>Chocolate Box<br><b>₹220</b></p></a>

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

<img src="chocs2.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Choki Choki <br>Chocolate Box<b>₹150</b></p></a>

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

<img src="chocs3.jpg"alt="HTML tutorial" style="width:175px;height:150px;">

<p><br><b>Pulips ice stick <br>₹50</b></p></a>

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

<img src="chocs4.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p> Chocoloony Milk Chocolate Basket<br><b>₹499</b></p></a>

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

<img src="chocs5.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Cigarette candy & lipstick candy<br><b>₹200</b></p></a>

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

<img src="chocs6.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Sweet and Tasty Lolipops<br><b>₹40</b></p></a>

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

<img src="chocs7.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p style="padding: 10px 0px;">Chocoballs Packet<br><b>₹250</b></p></a>

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

<img src="chocs9.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Love Milk Chocolate Box<br><b>₹150 </b></p></a>

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

<img src="chocs10.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Delight assorted<br><b>₹780</b></p></a>

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

<img src="chocs11.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Chocolate Candy Bar Mockup<br><b>₹499</b></p></a>

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

<img src="chocs12.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Assorrted Chocolates<br>Chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs13.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Hershey's kisses<br>Chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs14.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Delight assorted<br>Chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs15.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Brookside Dark Chocolate<br>Box<br><b>₹499</b></p></a>

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

<img src="chocs16.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>kitkat Milk & Dark<br>Chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs17.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Delight assorted<br>chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs18.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Delight assorted<br>Chocolate Box<br><b>₹499</b></p></a>

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

<img src="chocs19.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Dairy Milk Bookay <br><b>₹499</b></p></a>

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

<img src="chocs20.jpg" alt="HTML tutorial" style="width:175px;height:150px;">

<p>Delight assorted<br>Chocolate Box<br><b>₹499</b></p></a>

</div>

</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: #995200;

height: 80px;

width: auto;

}

label.dot{

color: white;

font-size: 25px;

line-height: 80px;

padding: 0 200px;

font-weight: bold;

font-family: cursive;

}

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;

font-family: cursive;

}

.vet{

margin-left: 48px;

display:flex;

flex-wrap:wrap;

float:left;

width:200px;

height:280px;

text-decoration:none;

text-align:left;

color:black;

padding: 50px 0px;

padding-top: 50px;

}

img {

border: 2px solid gray;

border-radius:10px;

}

.box{

background:white;

border: 1px solid #555;

align-items: center;

justify-content: center;

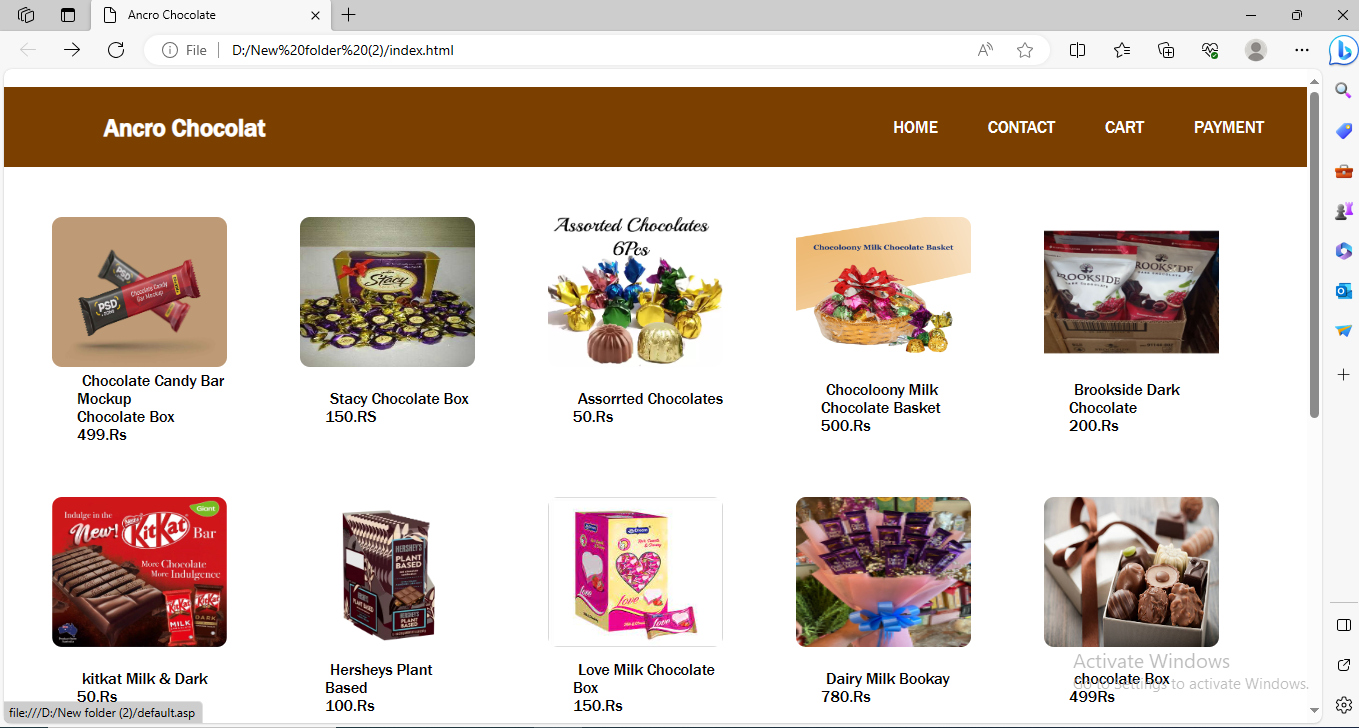
color:white;

}

.me{

text-decoration: none;

}

**HOME PAGE OUTLOOK:** 

E-COMMERCE APPLICATION

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.

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

6. 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).

7. Testing: Thoroughly test your e-commerce platform's functionality.

Consider using testing frameworks like pytest for automated testing.

8. Deployment:

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

9. 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.

10. 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>

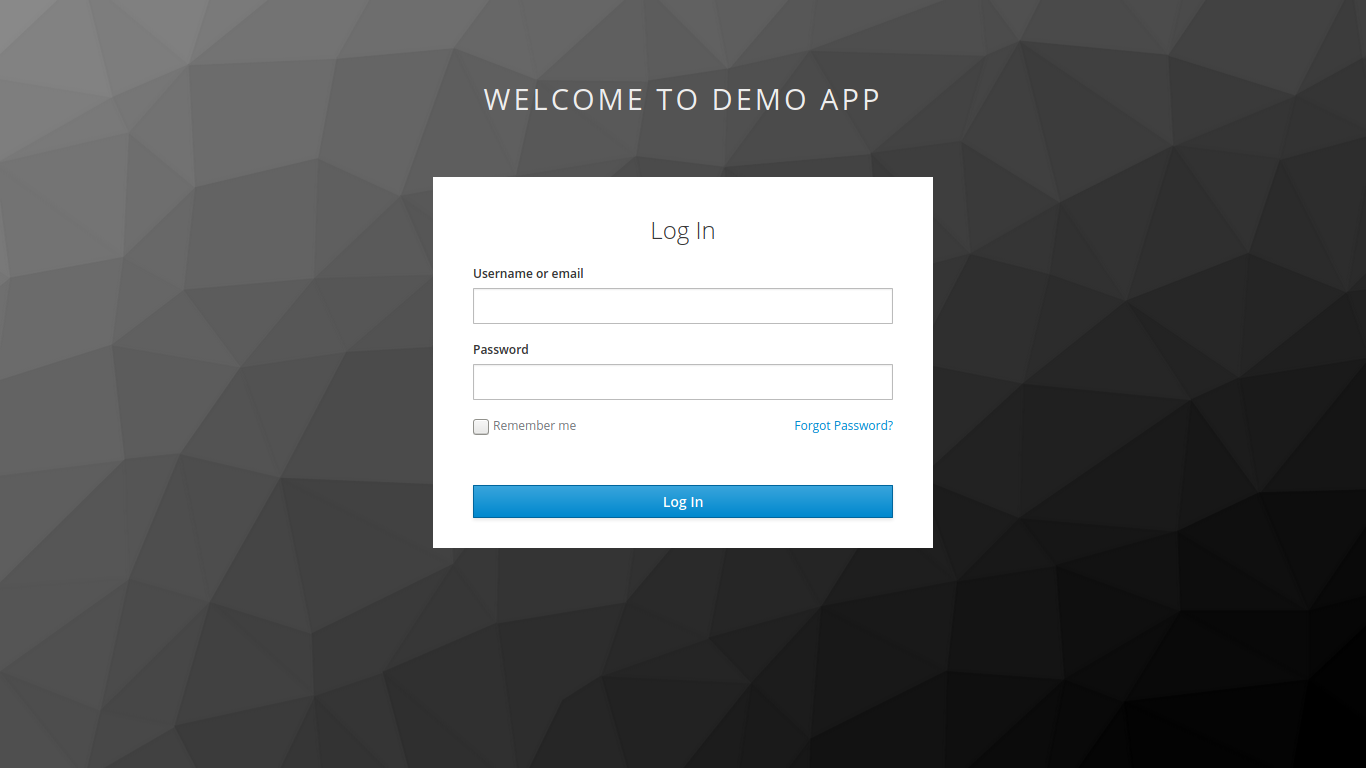
<h1>Please Log In</h1>

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

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

<input type="text" name="username" id="username" required><label for="password">Password:</label><input type="password” name="password" id="password" required> </form></body></html>

**LOGIN PAGE OUTLOOK:**



**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 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}`);

}); 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) => {

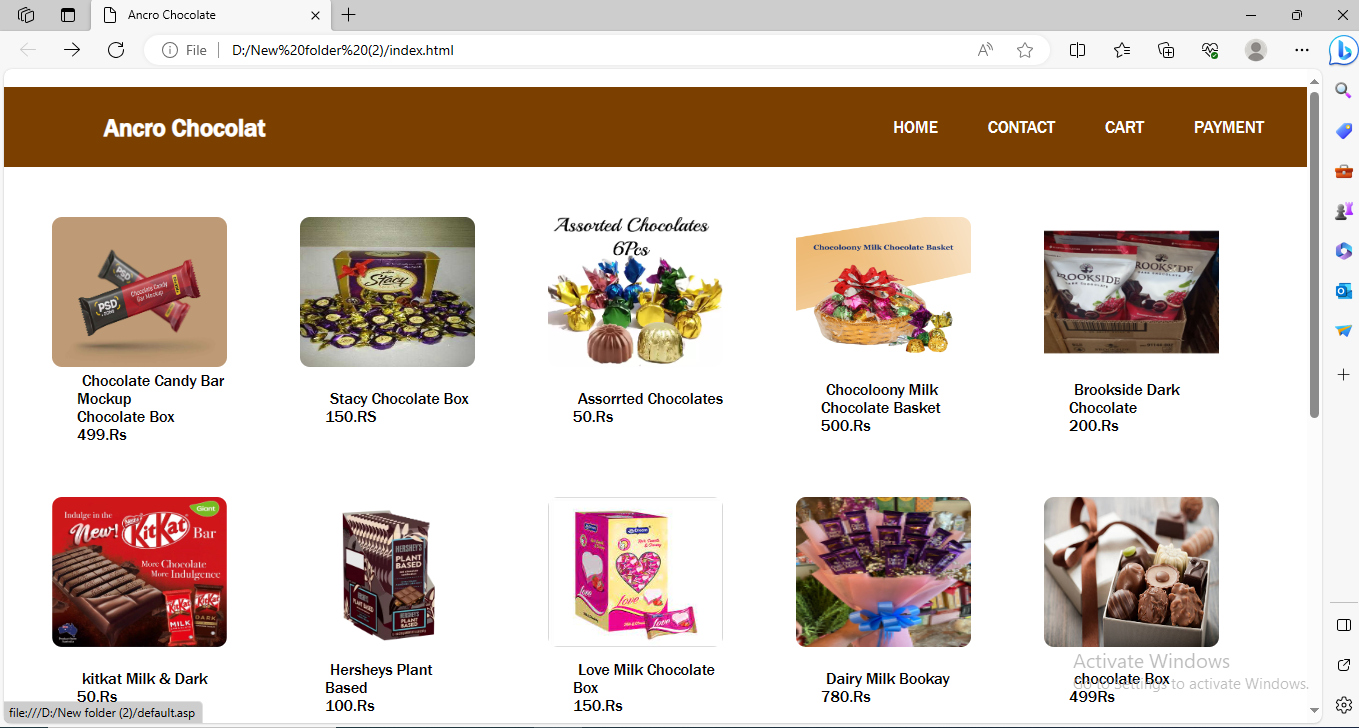
const productId = parseInt(req.params.productId, 10);

const product = products.find(p => p.id === productId);

if (product) {

cart.push({ product });

**OUTLOOK:**



**CART AND CHECKOUT:**

****