Project 2: E-commerce Application on IBM Cloud Foundry

File Name: **CAD_Phase3: e-commerce platform by implementing user authentication, shopping cart, and checkout functionality.**
Year: **3rd year**

Submitted by team,
1.Sulfeekar Ali - 721921104057
2.Albin George Kurian - 721921104012
3.Nihil - 721921104043
4.Ahamed shanih - 721921104008
5.Rahan - 721921104045
6.Vishnu - 721921104962
7.Akhil - 721921104011

#User Authentication and shopping cart functionality

User authentication is using nodejs, expressjs and SQL

```
const express = require('express');
const { Pool } = require('pg');
const app = express();
const port = process.env.PORT || 3000;
// PostgreSQL configuration
const pool = new Pool({
user: 'your_username',
host: 'your_host',
database: 'your_database',
password: 'your_password',
port: 5432,
});
// Database schema creation function
const createTables = async () => {
const createProductsTable = `CREATE TABLE IF NOT EXISTS products (
product_id SERIAL PRIMARY KEY,
product_name VARCHAR(255) NOT NULL,
description TEXT,
price DECIMAL,
image_url TEXT,
```

```
category_id INT
);`;
const createCategoriesTable = `CREATE TABLE IF NOT EXISTS categories (
category_id SERIAL PRIMARY KEY,
category_name VARCHAR(255) NOT NULL
);`;
const createUsersTable = `CREATE TABLE IF NOT EXISTS users (
user_id SERIAL PRIMARY KEY,
username VARCHAR(255) NOT NULL,
password VARCHAR(255) NOT NULL,
email VARCHAR(255) NOT NULL
);`;
const createOrdersTable = `CREATE TABLE IF NOT EXISTS orders (
order_id SERIAL PRIMARY KEY,
user_id INT,
product_id INT,
quantity INT,
total_price DECIMAL,
order_date DATE,
FOREIGN KEY (user_id) REFERENCES users(user_id),
FOREIGN KEY (product_id) REFERENCES products(product_id)
);`;
try {
await pool.query(createProductsTable);
await pool.query(createCategoriesTable);
await pool.query(createUsersTable);
await pool.query(createOrdersTable);
} catch (error) {
console.error('Error creating tables', error);
}
};
app.use(express.json());
// User registration endpoint
app.post('/register', async (req, res) => {
try {
const { username, password, email } = req.body;
const insertUserQuery = 'INSERT INTO users (username, password, email)
VALUES ($1, $2,
$3)';
await pool.query(insertUserQuery, [username, password, email]);
res.status(201).send('User registered successfully');
} catch (error) {
console.error('Error registering user', error);
res.status(500).send('Internal Server Error');
}
```

```
// User login endpoint
app.post('/login', async (req, res) => {
try {
const { username, password } = req.body;
const userQuery = 'SELECT * FROM users WHERE username = $1 AND password =
$2';
const { rows } = await pool.query(userQuery, [username, password]);
if (rows.length === 1) {
res.status(200).send('Login successful');
} else {
res.status(401).send('Invalid credentials');
}
} catch (error) {
console.error('Error during login', error);
res.status(500).send('Internal Server Error');
}
});
// Add to cart endpoint
app.post('/cart/add', async (req, res) => {
try {
const { userId, productId, quantity } = req.body;
// Implement shopping cart functionality here
// You need to manage user carts and quantities
res.status(200).send('Product added to cart successfully');
} catch (error) {
console.error('Error adding to cart', error);
res.status(500).send('Internal Server Error');
}
});
// Remove from cart endpoint
app.post('/cart/remove', async (req, res) => {
try {
const { userId, productId } = req.body;
// Implement shopping cart functionality here
// Remove products from the user's cart
res.status(200).send('Product removed from cart successfully');
} catch (error) {
console.error('Error removing from cart', error);
res.status(500).send('Internal Server Error');
}
});
```

This code is a Node.js application using the Express.js framework to create a web server. It connects to a PostgreSQL database for data storage. The code defines routes for user registration, login, and adding items to a cart. It uses SQL for database operations, handles errors, and communicates using standard HTTP status codes. The application is designed for building a RESTful API and includes async/await for asynchronous operations and environmental variables for configuration.

This code defines and implements API endpoints for user authentication, shopping cart management, and database interactions in a Node.js application using the Express.js framework. It allows users to log in and manage their shopping cart by adding and removing products. The code includes error handling and responds with appropriate HTTP status codes and messages based on the outcome of these operations.

#checkout functionality

```
// Checkout endpoint
app.post('/checkout', async (req, res) => {
try {
const { userId, products, totalPrice } = req.body;
// Implement the checkout process, including payment handling
// Create an order entry and update product quantities
res.status(200).send('Checkout successful');
} catch (error) {
console.error('Error during checkout', error);
res.status(500).send('Internal Server Error');
}
});
// Endpoint to fetch all products
app.get('/products', async (req, res) => {
try {
const { rows } = await pool.query('SELECT * FROM products');
res.json(rows);
} catch (error) {
console.error('Error executing query', error);
res.status(500).send('Internal Server Error');
}
});
app.listen(port, async () => {
console.log(`Server is running on port ${port}`);
await createTables();
});
```

Conclusion:

In conclusion, this project aims to address the challenges faced by e-commerce businesses by developing a scalable, reliable, and secure application on the IBM Cloud Foundry platform. By leveraging the capabilities of Cloud Foundry, the proposed solution seeks to provide businesses with a robust foundation for their online operations, enabling them to focus on delivering exceptional customer experiences in the dynamic digital marketplace. Through this project, we anticipate contributing to the growth and success of businesses in the e-commerce sector.