PROJECT PROPOSAL: E-COMMERCE WEBSITE

JYOTHISWAROOP SREENIVASULUREDDY (02147898)

Introduction

This project will develop an e-commerce platform that will cater to the needs of individuals visiting the site as they can view items, be provided with detailed descriptions, the ability to choose a quantity they require, and proceed to online payment. The website will be developed based on modern technologies and protocols so that it will be fast, secure, and effectively work in popular browsers. In terms of security, we shall strive to use HTTPS for secure transactions, and the main storage will be MongoDB; the key operations will be done through REST API.

What?

The project results provide an end-to-end e-commerce system aimed at developing a website that will make shopping on the Internet as comfortable as it can be. It will allow users to:

1. Search through a catalog of products arranged in type. Learn about products, their specifications, price, and whether they in stock or out out of stock.
2. Choose product amounts and include the product in their shopping cart.
3. The system will then ask the user to enter delivery details together with the order and take the user to the payment option, which will be secure.
4. For safe and secure payment processing use third-party payment gateways such as Stripe, PayPal, and the like.
5. However, users can optionally log in with an account that allows for the order history, wishlists, and personal details of the user.

Why?

E-business is one of the thriving industries and getting a safe and effective website is very crucial. The need to meet the current consumer demands requires a well-structured website with capacities for payment integration, encryption, and sufficient data storage. This project will:

1. Assist the businesses to get to as many consumers as possible not bounded by region.
2. To guarantee the customers' convenience as well as their safety while purchasing products.
3. Offer auditing tools for monitoring the user's activities and transactions for insights and fraud prevention in businesses.

4. Provide one with a flexible architecture to integrate with more features in the future and make it expandable with the growth of the business.

Supported Features

1. **UI**: They are fully responsive and optimized for mobile devices and built with HTML5 and JavaScript. It will remain functioning in the same manner in all the major browsers available, including Chrome, Firefox, and MS Edge and it will function well in all screen sizes.
2. **HTTPS with Localtunnel**: Localtunnel will be used to launch 'https' URL to enable interaction with the website in the safest way, whereby popular details like payment information will not be exposed to attackers.
3. **Backend**: Powered by Node. js or Java as, depending on the final project design, all logic and APIs will be implemented by this part.
4. **Database**: MongoDB will be used to store any information that is associated with products, users, orders, and transactions. Another great feature of MongoDB is its document-oriented storage, as well as the company's ability to scale properly and handle product details and orders.
5. **Authentication**: CB: (Optional) Secure user registration and Login can be done, we can use JSON Web Tokens (JWTs) or Sessions to enhance security. This lets users tailor settings or parameters; see the history of orders; and securely store or update individual data.
6. **Payment Gateway:** Support for integration with Stripe or PayPal for payment processing, so the users would not have to worry about their transactions being unsafe while the developers would not have to bother with optimizing the payment transactions for a better user experience during the checkout.
7. **Auditing and Logging:** It will be possible to audit everything taking place on the platform since user logins, product purchases, and admin activities will be recorded. It will be possible to divide the log into several levels (info, debug, error) to analyze the system at different depths.
8. **REST API:** It includes all the major features like product management, order processing, and payment gateways will be through REST API endpoints. This we will make it easy to interconnect with other systems or mobile applications in the future when the designs are fully implemented.

The Not Supported Features

1. **Multilingual Support:** This version will be initially developed in English only. Future versions could support more languages than the current version of the work is capable of supporting.

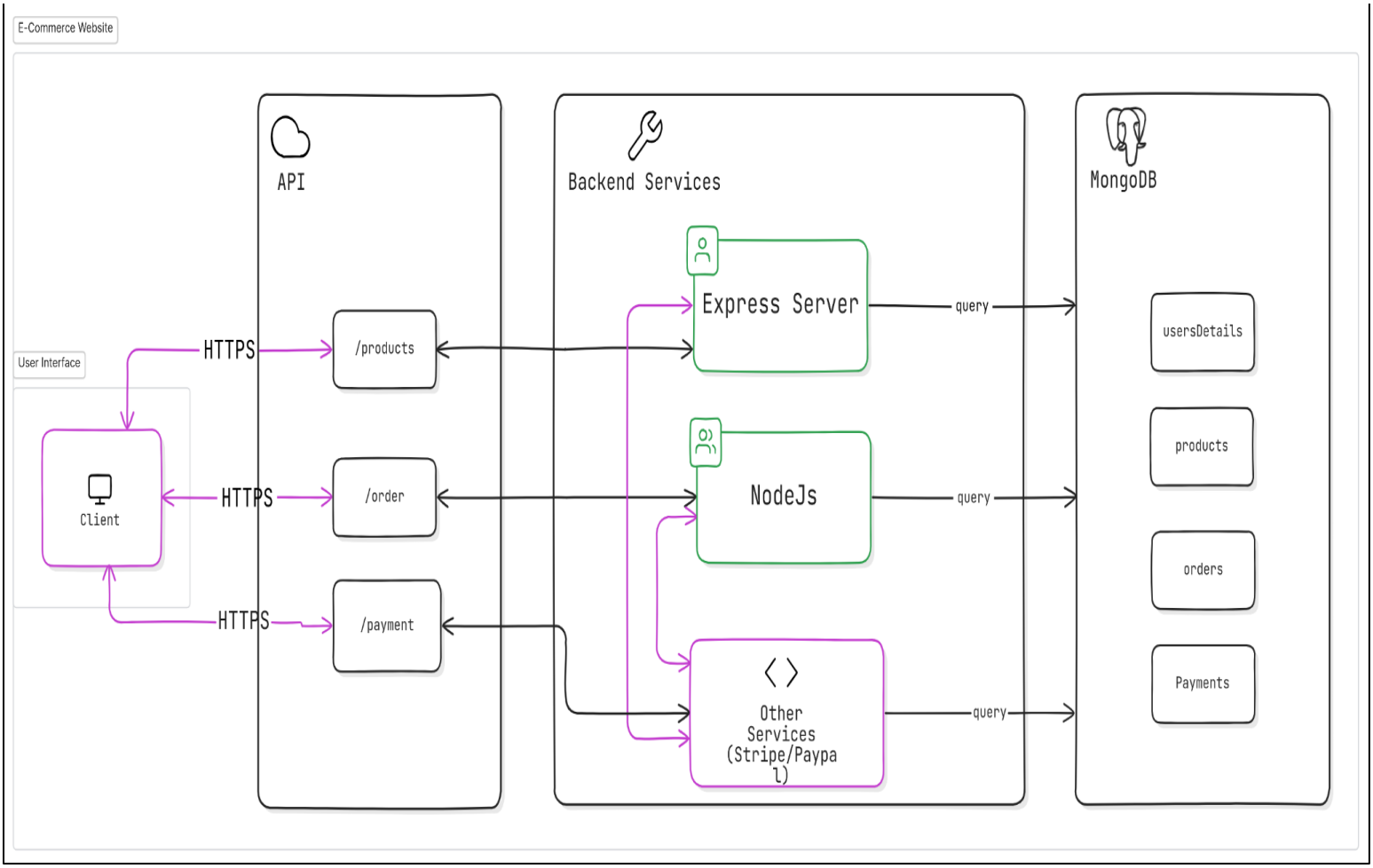
2. **Live Chat Support:** As implemented in this version, customer support will not be interactive through live chat. Instead, users will have to use contact forms or emails for support.

Future Planned Features

1. **Mobile Application:** The native Android and iOS apps' development is still under active consideration, which is aimed at using APIs of the web platform.
2. **Real-time Notifications:** In the future, utilizing WebSockets or server-sent events, users will be able to see real-time updates in the order status and cart, as well as promotions.
3. **Personalized Recommendations:** Artificial intelligence and Random models will be integrated to map the users' activity and make product recommendations based on their activity.
4. **Customer Reviews:** We are implementing an option that allows users to give their feedback about products they bought in the form of a rating, which will assist other customers in the future.

How to achieve the Website

E-Commerce Website Architecture



Components and Modules

1. **Frontend/UI Module**: HTML, CSS, and JavaScript have been used in the development. This module will take care of the entire client-side interactivity such as the presentation of products, capturing and processing user inputs (for instance, addition of the products to the cart and making an order), and rendering activities.
2. **Backend Module:** This backend will be develop in node.js (with Express) or Java (using Spring Boot), work as the middle layer between the client and the database. It shall also coordinate business logic such as ordering and authentication of the users.
3. **Database Module:** The last element in the architecture will be MongoDB which will serve as the database layer and will contain information about products, users, and orders. MongoDB has no rigid structure; the extension for the features to be added in the future will not be a problem.
4. **Authentication Module:** This module will carry out the user's authentication and sessions, with the JWT for Node, coming in handy. js or Spring Security for Java. We can define it as a server-side programming language for web applications or web API. For passwords, Bcrypt will be used in hashing the passwords.

5. **Payment Gateway Module:** Connecting with either Stripe or PayPal SDK for processing payments securely to the users safely and issuing receipts. Payment information will be stored as a record in the database and tied to the right orders.

Language to be Used for Each Module

1. **Frontend/UI**: HTML5, CSS3, JavaScript
2. **Backend**: Node.js (Express framework)
3. **Database**: MongoDB (NoSQL)
4. **Payment Gateway**: JavaScript SDKs provided by Stripe or PayPal

3rd Party/Open-source Modules

1. **Express.js**: For building web servers and API routes (Node.js)
2. **Mongoose**: For MongoDB interaction and schema modeling (Node.js)
3. **Winston**: Logging for Node.js applications to ensure system activities are traceable
4. **Bcrypt**: Password encryption for secure storage
5. **Stripe SDK**: For payment gateway integration
6. **Localtunnel**: For enabling HTTPS in development and testing environments

Table of Licenses

Module	**Licenses**
Express.js	MIT
Mongoose	MIT
Winston	MIT
Bcrypt	MIT
Stripe SDK	Proprietary
MongoDB	SSPL (Server Side Public License)

3rd Party Services / APIs

Stripe (Paid): For handling secure payments, and supporting multiple currencies and payment methods.
MongoDB Atlas (Free for small applications): Cloud-hosted database service for handling user and order data, with built-in security and backup options.
Localtunnel (Free): Enables HTTPS for development environments.

REST APIs Endpoint

1. GET METHOD: /products

This endpoint retrieves the product information that are available at the store.

Request Parameters

I. category (string): Filter products by category.
II. limit (int): Limit the number of products returned.
III. sortBy (string): Sort products based on price, popularity, or rating.

Response: Returns a JSON array of products

```
[
  {
    "id": "p123",
    "name": "Product Name",
    "description": "Detailed product description",
    "price": 19.99,
    "stock": 100,
    "category": "Category Name",
    "imageUrl": "http://example.com/product-image.jpg"
  },
  ...
]
```

2. POST METHOD: /order

Sends a new order containing the selected products by the user along with the relevant details of the user. This endpoint adds a new order and computes the total price from the items in the cart.

Request Payload

```
{
  "userId": "u456",
  "products": [
    {
      "productId": "p123",
      "quantity": 2
    },
    {
      "productId": "p789",
      "quantity": 1
    }
  ],
  "shippingAddress": {
    "street": "123 Main St",
    "city": "Nairobi",
    "zipCode": "00100",
    "country": "Kenya"
  },
  "paymentMethod": "stripe"
}
```

Response: Confirms that the order has been placed and provides a unique order ID and total amount.

```
{
  "orderId": "o789",
  "status": "order_placed",
  "totalAmount": 59.97,
  "expectedDelivery": "2024-09-25"
}
```

3. POST METHOD: /register

This is used to register a new user. The client will provide the user details such as email, password, and other necessary information.

Request Payload:

```
{
  "name": "John Doe",
  "email": "john.doe@example.com",
  "password": "supersecretpassword",
  "phoneNumber": "+254700123456"
}
```

Response: Provides a user ID and authentication token (JWT) upon successful registration.

```
{
  "userId": "u456",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

4. GET METHOD: /users/orders

This endpoint is used to fetch the order history of the users who are logged in.

Response: Returns a JSON array of orders associated with the authenticated user.

```
[
  {
    "orderId": "o789",
    "products": [
      { "productId": "p123", "name": "Product 1", "quantity": 2 },
      { "productId": "p456", "name": "Product 2", "quantity": 1 }
    ],
    "totalAmount": 59.97,
    "orderDate": "2024-09-19",
    "status": "delivered"
  },
  ...
]
```

5. POST METHOD: /payments

This endpoint is used to process the payment for a specific order. This endpoint will handle interactions with the payment getaways.

Request Payload:

1. orderId: The unique ID of the order being paid for.
2. paymentMethod: The selected payment method.
3. paymentToken: A token representing the payment details (received from the payment provider after the client submits card details).

```
{
  "orderId": "o789",
  "paymentMethod": "stripe",
  "paymentToken": "tok_1HMB1J2eZvKYlo2Cz7eOwr4T"
}
```

Response: Returns the payment confirmation, including a transaction ID from the payment gateway.

1. paymentStatus: Indicates whether the payment was successful (success or failed).
2. transactionId: The ID generated by the payment provider.
3. amountPaid: The total amount paid for the order.

4. currency: The currency used in the transaction.

```
{
  "paymentStatus": "success",
  "transactionId": "txn_1HMB2C2eZvKYlo2T53k4P1Al",
  "amountPaid": 59.97,
  "currency": "USD",
  "orderId": "o789"
}
```

Building Steps / Scripts

For Node.js:

1. Install dependencies

   npm install

2. Build the Project;

   npm run build

3. Start the application

   npm start

Install steps / Scripts

1. Start by making sure you do have nodeJs and it's package manager installed using the commands **node -v** and **npm -v**

2. In the terminal of the IDE move to the directory you want to have you project in using **cd.** Make the folder to hold the project using **mkdir ecommerce.** Change the directory to that of the project and do a **npm init -y**

3. After that make a **server.js** file inside the project

4. The next step is to have to install the dependencies we are going to use by doing a **npm install express mongoose cors body-parser dotenv stripe winston**

5. After successfully setting up the dependencies for the backend, we are going to create the frontend project folder using **npx- create-react-app client.**

6. Next it's to change the directory to that of client using **cd client**. After this we are going to install all the dependencies

7. Till here our project is has been fully set up. The next step is crating an account with MongoDB. After visiting the official website and creating the account, in our **directory** we are going to make a **database.js** file which will be used to intiate a connection.

8. Setup a .env file in the root directory to hold the MongoDB connection URL.

9. Test the project to see if it will run with local tunnel using the command **lt --port 3000**

Git Information:

Link:

https://github.com/jyothiswaroop-sreenivasulureddy/comp5130/tree/Week3

References

MongoDB documentation. (n.d.). MongoDB Documentation. https://docs.mongodb.com/

Express - Node.js web application framework. (n.d.). https://expressjs.com/

Stripe API reference. (n.d.). https://stripe.com/docs/api

Index | Node.js v22.9.0 Documentation. (n.d.). https://nodejs.org/en/docs/