Coffee Collaboration Platform

# Introduction

Welcome to the Coffee Collaboration Platform! This platform connects coffee enthusiasts and roasters, allowing users to discover unique coffee blends, share reviews, manage orders, and more. This platform is built to provide a seamless user experience, with features like product search, filters, wish lists, reviews, and order management, along with a secure checkout system using Stripe.

# Table of Contents

* - Features
* - Installation
* - Usage
* - Home Page
* - Product Page
* - Cart
* - Orders
* - Profile
* - Wishlist
* - Admin Dashboard
* - Protected Routes
* - Redis Caching
* - Bull Queue for Asynchronous Emails
* - Technologies Used

# Features

## Frontend

1. Home Page: Coffee blend recommendations and a newsletter signup section.

2. Product Page: Filter products by different categories (e.g., brands, price). Accessible through the navbar search field and brands page.

3. Product Detailed Page: Includes buttons for Add to Cart, Buy Now (with quantity controls), Add to Favourites, and a Review Module where users can manage their reviews.

4. Cart: Manage the quantity of items, clear the cart, and proceed to checkout using Stripe for payment.

5. Orders Page: View placed orders and their statuses.

6. Profile: Update preferences, shipping, and delivery addresses.

7. Wishlist: Manage favorite products.

8. Admin Dashboard: Admins can manage products, brands, orders, and logs.

## Backend

Node.js and Express handle user authentication, product management, and order processing. Integration with Stripe for payments. Redis caching is implemented, and Bull Queue is used for asynchronous email sending.

# Installation

## Prerequisites

* - Node.js
* - npm or yarn
* - MongoDB instance
* - Redis installed locally or on a server
* - Stripe account

## Steps

1. Clone the repository:

```bash  
git clone https://github.com/JayaRamSamavedam/Coffee.git  
```

2. Navigate to the project folder:

```bash  
cd Coffee  
```

3. Install dependencies for both frontend and backend:

```bash  
npm install  
cd client && npm install  
```

4. Set up environment variables in a `.env` file in the root directory:

```env  
STRIPE\_PUBLIC\_KEY=your\_stripe\_public\_key  
STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key  
MONGODB\_URI=your\_mongo\_uri  
JWT\_SECRET=your\_jwt\_secret  
REDIS\_URL=your\_redis\_url  
```

5. Start the development server:

```bash  
npm run dev  
```

# Usage

## Home Page

Displays coffee blend recommendations and a newsletter signup form.

## Product Page

Filter products and view them through search and brands page.

## Product Detailed Page

Review coffee products, add to cart, buy now, and manage reviews.

## Cart

Adjust quantity, remove items, or clear the cart. Proceed to checkout using Stripe.

## Orders

List all orders placed by the user.

## Profile

Update profile, shipping addresses, and preferences.

## Wishlist

Manage favorite coffee products.

## Admin Dashboard

Admins can create, edit, delete products and brands, manage orders, and view platform logs.

# Protected Routes

Users are authenticated before accessing routes like cart, orders, profile, and wishlist. Admins have additional access to the admin dashboard.

Example of Protected Routes:

```js  
const useAuth = () => {  
 const { user } = useContext(UserContext);  
 return user.loggedIn;  
};  
```

# Redis Caching

Redis is used to cache frequently accessed data, improving the performance of product fetching and other operations.

Example setup:

```bash  
npm install redis  
```

Example usage in the backend:

```js  
const redis = require("redis");  
const redisClient = redis.createClient();  
  
redisClient.on("error", (err) => {  
 console.error("Error connecting to Redis", err);  
});  
  
redisClient.on("connect", () => {  
 console.log("Connected to Redis");  
});  
  
const getProductDetails = async (productId) => {  
 const cachedProduct = await redisClient.get(productId);  
  
 if (cachedProduct) {  
 return JSON.parse(cachedProduct);  
 } else {  
 const product = await Product.findById(productId);  
 redisClient.set(productId, JSON.stringify(product));  
 return product;  
 }  
};  
```

# Bull Queue for Asynchronous Emails

Bull is used to handle asynchronous tasks, such as sending emails after an order is placed.

Example setup:

```bash  
npm install bull nodemailer  
```

Example usage in the backend:

```js  
const Queue = require("bull");  
const nodemailer = require("nodemailer");  
  
const emailQueue = new Queue("email");  
  
emailQueue.process(async (job) => {  
 const { email, subject, message } = job.data;  
  
 const transporter = nodemailer.createTransport({  
 service: "Gmail",  
 auth: {  
 user: "your-email@gmail.com",  
 pass: "your-email-password",  
 },  
 });  
  
 const mailOptions = {  
 from: "your-email@gmail.com",  
 to: email,  
 subject: subject,  
 text: message,  
 };  
  
 await transporter.sendMail(mailOptions);  
 console.log(`Email sent to ${email}`);  
});  
  
const sendOrderConfirmationEmail = (email, subject, message) => {  
 emailQueue.add({ email, subject, message });  
};  
  
sendOrderConfirmationEmail("customer@example.com", "Order Confirmation", "Thank you for your order!");  
```

# Technologies Used

* - Frontend: React.js, Ant Design
* - Backend: Node.js, Express
* - Payment: Stripe for handling payments
* - Database: MongoDB
* - Authentication: JWT (JSON Web Token)
* - Caching: Redis
* - Task Queue: Bull for asynchronous email sending
* - Deployment: Heroku (or any other platform)