A Comprehensive Technical Report on TasteTrove: An Elegantly Designed e-Commerce Solution for Spice Enthusiasts

RYAN NONO*, University of Windsor, Canada

TasteTrove[1] is an innovative e-commerce platform that offers a seamless online shopping experience for spice enthusiasts. It allows users to navigate through an array of spices, create personal accounts for a personalized experience, and proceed to a secure checkout. The application is developed using a range of technologies, including React[7], NodeJS, Typescript, Stripe[10], and Prisma ORM[8], among others. This report provides a detailed examination of the project, including its specifications, design, implementation, and deployment.

CCS Concepts: • **Software and its engineering → Designing software**.

Additional Key Words and Phrases: e-commerce, full stack, client side routing, express js[9], react[7], json web token[6], dynamic cart

ACM Reference Format:
Ryan Nono. 2023. A Comprehensive Technical Report on TasteTrove: An Elegantly Designed e-Commerce Solution for Spice Enthusiasts. *J. ACM* 37, 4, Article 111 (August 2023), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The TasteTrove[1] project is an e-commerce platform dedicated to delivering a wide variety of spices directly to consumers. At its core, the project has an intuitive online shopping interface that seamlessly guides users through the exploration and purchase of their desired products. TasteTrove is a harmonious blend of cutting-edge technologies and meticulous design, with the goal of creating an enriching shopping experience that satisfies the needs of spice enthusiasts worldwide.

1.1 Motivation

TasteTrove[1] was inspired by the recognition of a gap in the online market for a specialized platform catering to spice lovers. As more consumers turn to online shopping, it became evident that a well-curated and user-friendly platform dedicated to spices could fill this void and connect consumers with the diverse world of spices at their fingertips.

Moreover, the current global landscape underscores the significance of online shopping platforms for businesses. With an ever-increasing number of consumers shifting to online shopping, it has become paramount for businesses to have a robust digital presence to reach this expanding audience. TasteTrove serves as a valuable channel connecting spice suppliers with consumers, contributing to the digital transformation of the spice retail sector.

Author's address: Ryan Nono, ryan_nono@hotmail.com, University of Windsor, 401 Sunset Ave, Windsor, Ontario, Canada, N9B 3P4.

1.2 Importance

TasteTrove[1] is of great importance. As an online platform, it significantly broadens the reach of businesses, connecting them with consumers globally, which is a geographical advantage that traditional brick-and-mortar stores may lack. Furthermore, the platform's emphasis on creating an intuitive and user-friendly interface enhances the customer's shopping experience, thereby increasing customer satisfaction and loyalty.

Additionally, TasteTrove provides a critical avenue for businesses to adapt to the evolving retail landscape, in which digital presence and e-commerce have become vital for survival and growth. By providing an online marketplace specifically tailored for spices, TasteTrove can help businesses tap into new customer bases and market opportunities, further underscoring its importance.

In conclusion, TasteTrove is more than just an online shop; it is a platform that combines technology and commerce to shape the future of the spice retail industry, providing unparalleled value to both businesses and consumers.

2 RELEVANT LITERATURE REVIEW

2.1 Thao Huynh's Bachelor's Thesis "The Design and Development of an E-Commerce Web Application"

The task of building an effective, robust, and user-friendly e-commerce web application is a complex one, requiring an intricate knowledge of various technologies and their interplay. Thao Huynh's Bachelor's thesis "The Design and Development of an E-Commerce Web Application" [4] offers a comprehensive blueprint of the development process, making it an essential reference for our project. Huynh leverages HTML5, CSS3, Bootstrap 4, Node.js, JSON, and React[7] to construct a responsive, adaptable, and user-friendly e-commerce platform, focusing on facilitating mobile shopping and prioritizing functionality, browser compatibility, and responsiveness.

2.2 "SQL vs NoSQL: What You Need to Know"

This piece of literature comprehensively discusses the comparative merits of SQL and NoSQL database management systems [3]. Despite the adaptability of NoSQL databases in handling large volumes of unstructured data, the IBM Cloud Blog advocates for SQL databases for e-commerce applications due to their strong data consistency, complex query support, security features, and widespread developer familiarity. The focus on data consistency and security in SQL is crucial for our project, especially considering the importance of secure transactions and customer data protection in an e-commerce platform.

2.3 "State of the Developer Nation Q1 2021" Report by SlashData

Finally, the "State of the Developer Nation Q1 2021" report by SlashData [2] provides an updated overview of popular programming languages, libraries, and frameworks among global developers. The report states that JavaScript, with its dual functionality for both front-end and back-end development, is currently the most widely used programming language.

3 PROJECT DETAILS AND METHODOLOGY

This project was carried out in a systematic and structured manner to ensure its successful completion. We began by gathering and analyzing the requirements, identifying key functionalities, and analyzing the data requirements for each functionality.

Next, we designed a robust and normalized database schema using PostgreSQL[15] and established a connection between the server and the database using Prisma[8]. We also created REST API endpoints for CRUD operations for each entity, including products, customers, and orders. These endpoints were tested to ensure that they functioned as expected.

User authentication and authorization functionality were designed and implemented using JSON Web Tokens[6] (JWTs). Then, the React[7] development environment was set up, and the User Interface (UI) was designed with React components. We connected the frontend with the backend using Axios[12] for HTTP requests, and state management was implemented using React's Context API.

Once all functionalities were thoroughly tested, the app was optimized for production and subsequently deployed. Post-deployment testing was carried out to ensure that the application functioned as expected.

The main features of the online shop web app include user authentication, product display, cart management, and a checkout process that involves secure payment gateway integration. The system's architecture comprises a client-side (frontend) interface, a server-side (backend) component, and a relational database for data persistence.

The technologies selected for this project are purposefully chosen to align with the development of a dynamic, secure, and scalable online shop web app:

- **Frontend:** The frontend of the application is developed using a combination of HTML, CSS, and JavaScript, along with the ReactJS[7] library. HTML and CSS form the base for the structure and design of the user interface, while JavaScript ensures the interface is dynamic and interactive. ReactJS, being a component-based library, allows for efficient rendering of UIs and ensures maintainability and reusability of code. The checkout process incorporates Stripe[10] Elements to build a custom UI for secure payment.
- **Backend:** The server-side component is developed using Node.js, a runtime environment that executes JavaScript outside of a web browser, allowing us to handle server-side logic. We are leveraging Express.js[9], a lightweight and flexible Node.js web application framework, to structure our web application and create our RESTful API endpoints.
- **Database:** Postgress[15], a popular relational database, is employed for data persistence. It supports the complex relations between different data entities that are common in an online shop web app. It stores information about users, products, and orders in separate tables, 'users', 'products', 'orders', and 'order_items', respectively.
- **Routing:** On the client side, we are using the createBrowserRouter function for routing purposes. It allows us to define different views in the application, like the Home view, ProductPage view, and Auth view, as well as manage the checkout process.

The project adopts a modular and component-based development methodology. This approach ensures that each part of the application is developed and tested separately before integrating it into the larger system. This not only enhances the development speed but also improves the maintainability of the system, as changes in one module do not affect others.

Moreover, the system's design adheres to best practices like separation of concerns and the DRY (Don't Repeat Yourself) principle. Each component or module is responsible for a single functionality, and code repetition is minimized. This leads to a clean and efficient codebase that can scale smoothly as the project evolves.

4 DEFINITIONS

To ensure a complete understanding of the project, we will define some key terms and introduce additional ones relevant to our project:

- `Front-end development`: This refers to the development of the Graphical User Interface (GUI) that users interact with in a web application. It involves creating the design, structure, behavior, and content of everything seen on browser screens when using a web app. The languages primarily used for front-end development include HTML, CSS, and JavaScript.
- `Back-end development`: This is the server-side development of web applications or websites, focusing mainly on how the site works. It uses databases and servers to ensure that the website or web app functions correctly, processes the data needed to complete the user's commands, and serves the right content to users. For our project, we utilize Node.js and Express.js[9] for back-end development.
- `Online shop web app`: This refers to a web application that allows users to browse, select, and purchase goods or services online. The TasteTrove[1] is a perfect example of an online shop web app, which sells a variety of spices. The web app will include features like user authentication, product display, cart management, and a secure checkout process.
- `Stripe`[10]: A technology company that builds economic infrastructure for the internet. Businesses of every size—from new startups to public companies—use Stripe's software to accept payments and manage their businesses online. For our project, Stripe is the chosen payment gateway for processing online transactions.
- `React`: A JavaScript library for building user interfaces or UI components maintained by Facebook and a community of individual developers and companies. React helps us create a dynamic and responsive front-end for our web app.
- `NodeJS`: An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. It allows us to handle server-side logic of our application.
- `Typescript`: An open-source language that builds on JavaScript, one of the world's most used tools, by adding static type definitions. Typescript allows us to write more robust code and maintain a cleaner and more organized codebase.
- `Prisma ORM`: An open-source database toolkit. It replaces traditional ORMs and makes database access easy with an auto-generated and type-safe query builder that's tailored to your database schema. It enables us to seamlessly perform database operations.
- `PostgresSQL`: A relational database management system based on SQL (Structured Query Language). In our project, we use PostgresSQL[15] for storing and managing data related to users, products, and orders.
- `Express.js`: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. We use Express.js to structure our web application and create our API endpoints.
- `createBrowserRouter`: A function used for handling routing on the client side in our application. It defines different views such as home, product page, and auth view, along with managing the checkout process.

These definitions should provide a solid foundation for understanding the terminologies and technologies used in the development of our online shop web app.

5 SPECIFICATIONS

At its core, the TasteTrove[1] application is a comprehensive e-commerce solution, offering a multitude of advanced features designed to provide a seamless and intuitive online shopping experience. Utilizing the power of contemporary web technologies and frameworks, it presents a dynamic single-page application that enables smooth navigation across the platform without the need for page reloads.

5.1 Key Features of TasteTrove

- **User Registration and Login**: TasteTrove is equipped with a secure authentication system. It allows users to create personal accounts, manage their profiles, and securely log in to access personalized features and account-specific data.
- **Product Catalog**: The application showcases an extensive range of spices, each with detailed product descriptions. Customers can browse through this assortment effortlessly, with each product encapsulated in its own dedicated page.
- **Shopping Cart Functionality**: TasteTrove features an interactive shopping cart, offering users the ability to add their desired products with ease. The shopping cart keeps track of chosen items, quantities, and total pricing, allowing users to review and adjust their selections before proceeding to checkout.
- **Server-Side Product Management**: Behind the scenes, TasteTrove implements server-side product management, ensuring the application remains responsive and up-to-date with the latest product information and availability status.
- **Search Functionality**: The application incorporates an intelligent search function that can quickly filter through the product catalog based on user input, helping customers find the exact spices they're looking for.
- **Guest Checkout**: TasteTrove recognizes the need for spontaneity in online shopping. For users who wish to make quick purchases without creating an account, the application provides a guest checkout feature. This option streamlines the buying process while maintaining transaction security.
- **Secure Checkout**: At the end of the shopping journey, TasteTrove provides a secure and straightforward checkout process. It employs the Stripe[10] Payment Gateway to handle transactions, ensuring a safe and reliable transfer of payment data.
- **Dynamic Routing**: Leveraging the power of React Router[11], the application offers dynamic and nested routes, supporting unique paths for various pages such as home, product details, authentication (sign-in/register), and checkout processes.

In essence, TasteTrove[1] combines user-friendly design with robust functionality, crafting a unique and engaging shopping experience tailored specifically for the discerning spice enthusiast.

6 PLATFORM

The TasteTrove[1] application is constructed using an assortment of cutting-edge technologies, libraries, and deployment platforms. These were strategically chosen based on their individual strengths and the collective synergy they provide to align with the project's objectives: creating a dynamic, secure, and user-friendly e-commerce platform.

6.1 Front-end Development

React[7], combined with TypeScript, was chosen as the core of our front-end development. React's component-based architecture and dynamic rendering capabilities, in conjunction with TypeScript's static typing, allow us to build highly interactive, efficient, and error-prone user interfaces. We further enhanced the site's responsiveness and aesthetic appeal by incorporating TailwindCSS[5] and the Material UI library. Axios[12] was selected for managing HTTP requests due to its extensive feature set, including promise-based asynchronous operations and automatic JSON data transformation.

6.2 Back-end Development

The back-end of TasteTrove[1] is crafted with NodeJS and TypeScript, a combination that provides a powerful runtime environment and robust type checking. Express[9], a minimalist web framework for NodeJS, was chosen for its simplicity and flexibility in setting up server-side routing and middleware. To manage interactions with the PostgreSQL[15] database, we employed Prisma ORM[8], known for its type-safe database access and powerful query capabilities. Ensuring user data security is paramount in our design; thus, we use JSON Web Tokens (JWT)[6] for secure authentication and BCRYPT for password hashing.

6.3 Payment Processing

Stripe[10], a globally trusted online payment processing platform, was chosen to ensure secure and straightforward transaction management. Its comprehensive API and extensive documentation made it the ideal choice for integrating online payment functionality into TasteTrove[1].

6.4 Deployment

Railway.app[13] and Vercel[14] were selected as our deployment platforms for the server/database and client respectively. Railway.app provides a straightforward deployment process and robust support for PostgreSQL[15] and NodeJS applications, aligning perfectly with our back-end technologies. Vercel, on the other hand, offers seamless deployment solutions for front-end applications, particularly those built with React[7], making it an optimal choice for TasteTrove's[1] client-side deployment.

The strategic selection and integration of these platforms and technologies form the backbone of the TasteTrove application, allowing us to create an online shop web app that is not only efficient and secure but also offers an intuitive and enjoyable user experience.

7 DESIGN

The TasteTrove[1] application's design philosophy hinges on simplicity, user-centricity, and functionality, drawing upon established User Experience (UX) principles to drive its architecture. We've crafted an uncomplicated yet aesthetically appealing user interface (UI), ensuring it's easily navigable even for novice online shoppers.

Our design strategy places users at its core. For instance, the product browsing experience has been streamlined to enable users to effortlessly traverse through the vast array of spices. Each product card carries key information, such as name, price, and a brief description, at a glance, ensuring users can make informed choices before adding items to their cart. The shopping cart design is intuitive and provides an overview of selected items, their quantities, and the total cost, leading to a seamless checkout process.

The search functionality holds paramount importance in our design strategy. With the aim to empower users in quickly finding their desired spices, we've incorporated a highly responsive search feature. It not only facilitates searches based on the spice name but also provides suggestions as users type, making the process efficient and user-friendly.

8 EXPERIMENTAL SETUP

In the context of the TasteTrove[1] application, our experimental setup encompassed an array of stages that facilitated the seamless development, testing, and refinement of both front-end and back-end environments.

Our setup initiated with the establishment of a robust development environment using various technologies like Node.js and Express.js[9] for the back-end, React[7] for the front-end, and PostgreSQL[15] and Prisma ORM[8] for our database needs. This stack was selected keeping in mind its potential for creating a performant, secure, and scalable e-commerce platform.

The heart of our experimental setup was the creation and testing of REST API endpoints for performing CRUD (Create, Read, Update, Delete) operations for our core entities such as product, customer, and orders. This allowed us to effectively simulate and test the interactions that would take place between the front-end and back-end of the application, ensuring the endpoints responded as expected under various scenarios.

User authentication and authorization features were of critical importance to the security of our platform, and their design and implementation were carefully executed within this setup. These features underwent thorough testing to verify their integrity and reliability, ensuring a safe and secure environment for our users.

Beyond the technical setup, we also engaged in user testing sessions as part of our experimental approach. We solicited feedback from a diverse group of potential users to gain insights into the usability and performance of the TasteTrove[1] platform. This iterative process of gathering user feedback, making necessary adjustments, and testing again was instrumental in refining the user interface, improving the user journey, and enhancing overall platform performance. It also played a crucial role in shaping the product's final version, ensuring it not only met but exceeded user expectations.

In summary, our experimental setup allowed us to construct and fine-tune the TasteTrove[1] application in a controlled, methodical manner, placing equal emphasis on technical functionality and user experience.

9 IMPLEMENTATION DETAILS

The implementation of the TasteTrove[1] application involved a step-by-step approach, starting with the requirements gathering and analysis, followed by designing the database schema, setting up the server, connecting with the database, and building API endpoints. After implementing the user authentication and authorization, we proceeded to set up the React[7] environment and designed the user interface.

The frontend was then connected to the backend, and client-side routing was implemented. After thorough testing and debugging, the application was optimized for production and then deployed. Post-deployment testing was carried out to ensure the application functioned as expected.

9.1 System Architecture

The architecture of the system is based on a client-server model, where the server exposes a set of RESTful APIs, and the client consumes these APIs to provide the desired functionalities to the end users.

The main components of the system architecture include:

(1) **Client-Side (Frontend):** The frontend of the application is built using the ReactJS[7] library, which is known for its efficient rendering of complex user interfaces (UIs). The application uses a component-based architecture where each component is responsible for a particular part of the UI. The StripePaymentGateway, PaymentRoutingManager, and PaymentSubmissionForm components are some of the critical components responsible for handling the checkout process in the application. The client-side application also uses a router, constructed with createBrowserRouter, to manage different views of the application like Home, ProductPage, Auth, and the Checkout process. The router uses a declarative configuration approach to define routes and nested routes for different parts of the application.

(2) **Server-Side (Backend):** The backend server of the application is built using Node.js, which provides a scalable and efficient environment for building web servers and APIs. The server exposes RESTful APIs for managing users, products, and the checkout process. The server interacts with a relational database to persist data and performs various operations like user authentication, order management, and payment processing.

(3) **Database:** The database used in the system is a relational database, PostgresSQL[15]. The schema is designed with four main tables: 'users', 'products', 'orders', and 'order_items'. The 'users' table contains information about registered users. The 'products' table holds details about the products available for purchase. The 'orders' table keeps track of orders made by users, and the 'order_items' table contains information about each item within a given order.

(4) **Payment Gateway:** For handling payments, the application integrates with Stripe[10], a popular payment gateway. The StripePaymentGateway component in the frontend handles the integration with Stripe. It uses Stripe Elements for building the payment UI and provides options like amount, currency, and mode to the Stripe API for processing the payment.

The system follows best practices like modular design, separation of concerns, and DRY (Don't Repeat Yourself) principles. This allows for better maintainability and scalability as it can be extended with new features without affecting existing functionality. Furthermore, it ensures the system can handle a growing number of users and data efficiently.

9.2 Client-Side Routing

In the development of a single-page application like our project, client-side routing plays a crucial role. It allows us to manipulate the browser history and display different components to the user based on the URL, without requiring a full page refresh.

For this project, we've utilized the ‘createBrowserRouter‘ function to set up routing in our application. The router configuration comprises a root layout element (‘GlobalLayout‘), and an array of route objects each specifying a unique path and the component to render for that path.

Here's a brief overview of the routes in the application:

- **Home Route (‘/‘):** This is the root or default route of our application. When users navigate to the base URL of our application, they see the ‘Home‘ component.
- **Product Page Route (‘/product/:productId‘):** This route displays details for an individual product. The ‘:productId‘ in the path is a URL parameter that allows us to show information for different products based on the ID passed into the URL.

- **Authentication Routes (‘/auth/signin‘ and ‘/auth/register‘):** These routes are used for the user authentication process. The ‘signin‘ and ‘register‘ paths are nested routes under the ‘auth‘ parent route, which render the sign-in and registration forms, respectively.
- **Checkout Routes (‘/checkout‘ and ‘/checkout/complete‘):** The checkout routes handle the payment process. The ‘complete‘ path is a nested route under the ‘checkout‘ parent route, providing feedback to the user once they’ve completed the payment process.

Below is the complete configuration of the router:

```
const router = createBrowserRouter([
  {
    element: <GlobalLayout />,
    children: [
      {path: '/', element: <Home />},
      {path: 'product/:productId', element: <ProductPage />},
      {
        path: 'auth',
        element: <Auth />,
        children: [{path: 'signin'}, {path: 'register'}],
      },
      {
        path: 'checkout',
        element: <StripePaymentGateway />,
        children: [{path: 'complete'}],
      },
    ],
  },
]);

export default router;
```

As the application grows, this structure can be expanded to accommodate new routes and nested routes as necessary, keeping the user interface organized and ensuring a smooth navigation experience for our users.

10 HOME PAGE

The `Home` component is the heart of the TasteTrove[1] application, serving as the landing page for our users. It is a functional component developed in React[7], signifying that it is a JavaScript function that accepts an input (called ’props’) and returns a React element that describes how a section of the UI should appear.

Below are the main parts of this `Home` component:

10.1 Component Contexts

The `Home` component leverages two key contexts – `ProductsContext` and `FilterContext`.

(1) **ProductsContext**: The `ProductsContext` is utilized to fetch all the product data. This data is then mapped to generate a gallery of product cards, each representing an individual spice product available on the platform.
(2) **FilterContext**: The `FilterContext` provides functionality for filtering the displayed products based on user input. The filtering process checks if the user's input matches any part of the product name or category, and displays only the matching products.

10.2 Component Parts

The `Home` component is divided into two main sections:

(1) **The Hero Section**: This section consists of an engaging banner and a Call-To-Action (CTA) button. Upon clicking this button, the view scrolls down to the Product Gallery section, thus guiding the user towards the available products.
(2) **The Product Gallery Section**: This section displays the product cards that represent the variety of spices offered by TasteTrove[1]. While the product data is being fetched, a spinner (CircularProgress) is displayed to indicate loading. This design choice enhances the user experience by offering visual feedback during potentially long-loading operations.

The design of the `Home` component effectively guides users from their initial landing on the page, through an engaging Hero section, and finally to a comprehensive display of products. The use of contexts ensures efficient data handling and dynamic rendering based on user input, embodying a well-architected UI component that encapsulates React's[7] capabilities.

Lastly, the `Home` component is wrapped in HTML semantic elements (`header` and `main`) that define its structure. These semantic elements play a vital role in improving the accessibility of the web content by providing information about the type of content contained within them, making it easier for screen readers and other assistive technologies to interpret the page structure and content.

11 PRODUCT PAGE

The `ProductPage` component serves a pivotal role in the TasteTrove[1] application as it provides the detailed view of individual products, allowing users to understand the product's specifics and make informed decisions.

11.1 Component Functionality

Built as a functional component in React[7], `ProductPage` employs various hooks and functions to accomplish a range of tasks.

The product ID is obtained from the URL using the `useMemo` hook and `useLocation` hook provided by React Router[11]. Subsequently, the product details are fetched from a context object using the `useProductsContext` hook. The `useState` hook is utilized to manage the state for the currently selected product image. This allows users to view different images of the selected product.

Two main functions, `handleAddToCart` and `handleCheckout`, are designed within this component. The former enables the user to add the current product to their shopping cart, while the latter aids the user in proceeding to the checkout page. Both actions are executed upon button clicks.

11.2 Component Design

The `ProductPage` component returns a `main` element, which houses all the product details and associated actions. This includes a high-quality image of the product, its name, price, stock availability, category, and a detailed description. It also contains two call-to-action buttons for adding the product to the cart and proceeding to checkout.

In terms of design, the `ProductPage` component splits into two sections:

(1) **The Left Section**: This part of the layout displays the main product image and additional secondary images. Users can switch the main view to any of the secondary images by clicking on them.
(2) **The Right Section**: This part contains all the textual information about the product and the call-to-action buttons.

The `ProductPage` component's design ensures that users have all the necessary details they need to decide on their purchase, coupled with an effortless navigation system. Moreover, the use of contexts and hooks demonstrates the effective use of React's[7] capabilities to manage state and side effects. Semantic HTML tags, such as `main`, `section`, and `div`, ensure web accessibility and enhance the structure and readability of the code.

12 PAYMENT GATEWAY AND CHECKOUT PROCESS

This section describes the implementation of the checkout process and payment gateway using Stripe[10]. It includes three main components: `StripePaymentGateway`, `PaymentRoutingManager`, and `PaymentSubmissionForm`.

12.1 StripePaymentGateway Component

`StripePaymentGateway` is the main functional component representing the Stripe payment gateway. It uses the cart context to fetch cart items and calculates the subtotal, tax amount, and total amount for the payment.

It sets up the Stripe elements options and, based on whether the cart is empty or not, either displays a message with a redirection button or renders the `PaymentRoutingManager` component inside the Stripe Elements.

This component is critical to the checkout process as it prepares and presents the Stripe payment system within the application, providing a secure way for customers to complete their purchases.

12.2 PaymentRoutingManager Component

`PaymentRoutingManager` is a functional component that handles page routing for the checkout process. Based on the current location, the page type is set to either 'checkout' or 'checkoutComplete'. Depending on the page type, it either renders the `PaymentSubmissionForm` component or displays a message with a redirection button.

By managing the routing of the checkout process, this component enables smooth transitions between different stages of the checkout process, ensuring a seamless user experience.

12.3 PaymentSubmissionForm Component

The `PaymentSubmissionForm` component is the core of the checkout process. It uses Stripe and Elements hooks, as well as the cart and user contexts. It manages the user's address state, calculates the subtotal, tax amount, and total amount for the checkout, and sets up options for the Stripe address elements.

This component handles form submission, creates the payment intent, confirms the payment, and catches any errors that may occur. It also renders the payment and address elements, totals, submission button, and the cart products.

12.4 StripeAddressValue Type

`StripeAddressValue` is a TypeScript type that represents the value of a Stripe address. It contains properties for the user's name, first name, last name, address details, and phone number. This type is used in the `PaymentSubmissionForm` to manage the user's shipping address information.

Through these components and the use of the Stripe API, the application provides an intuitive, secure, and reliable checkout process for its users, contributing to an overall positive shopping experience.

13 DATABASE SCHEMA DESCRIPTION

In this section, we'll provide a comprehensive view of the database schema used in the TasteTrove[1] application. The application uses a PostgreSQL[15] database and leverages the Prisma ORM[8] (Object-Relational Mapping) for handling database operations. The schema layout is designed to optimize the efficiency and integrity of data management processes within the app.

13.1 User Model

The `User` model serves as the primary entity in our application, capturing essential details of a user including their first name, last name, email, password, and role. It also keeps track of each user's shopping cart and saved addresses. The role is defined by an enumeration type `Role`, which can be `BASIC` or `ADMIN`.

13.2 Cart and CartItem Models

The `Cart` model is associated with a user and contains a collection of `CartItem` objects. Each `CartItem` has a reference to a specific `Product` and maintains the quantity of that product in the cart.

13.3 RefreshToken Model

The `RefreshToken` model is used for managing user session tokens. This model contains a unique token for the user and timestamps indicating when the token was created and when it expires.

13.4 Product, ProductImage, and ProductCategory Models

The `Product` model represents the items available for purchase in the store. Each product has a unique name, a description, a price, a stock quantity, and a category, which is a relation to the `ProductCategory` model.

The `ProductImage` model maintains information about each product's images, including a URL for the image and a flag indicating whether it's the primary image for the product.

The `ProductCategory` model is used for categorizing products. Each category has a unique name and a list of products belonging to it.

13.5 Order and OrderItem Models

The `Order` model represents a user's order. It includes a total price, shipping address, payment intent ID from Stripe[10], and order status. The `OrderStatus` enumeration defines the various states an order can be in—payment initiated, payment succeeded, shipped, and delivered.

The `OrderItem` model captures specific details of each product in an order, including the product reference and the quantity ordered.

13.6 Address Model

The `Address` model holds the details of an address. These details include two lines for the street address, city, province, postal code, and country. It can be associated with multiple users and orders.

Through this schema, TasteTrove[1] manages all the necessary information to support its core features, such as user authentication, product management, cart management, order processing, and address management. This structure ensures data consistency and integrity throughout the application while providing a solid foundation for potential feature expansions.

14 RESULTS

The implementation of the TasteTrove[1] application has been a success. The application effectively serves as a one-stop-shop for spice enthusiasts, providing an extensive variety of spices, coupled with a seamless online shopping experience. The user-friendly interface, secure checkout process, and efficient search functionality have all been highly appreciated by users. Furthermore, the guest checkout feature, which allows for spontaneous shopping without necessitating account creation, has been a unique addition that differentiates TasteTrove from other e-commerce platforms.

The application has been running smoothly post-deployment with minimal issues, thanks to thorough testing and debugging. It has also demonstrated excellent scalability and performance, handling increased traffic with ease. The application's robust architecture, coupled with its attractive UI, has been key to its success.

15 DISCUSSION

The implementation of the TasteTrove[1] application presented a number of challenges, including ensuring a seamless user experience, securing the user data, and managing the state of the application. However, these challenges were successfully overcome by using a range of technologies and following a structured approach.

The use of React[7] for front-end development allowed the creation of a dynamic and intuitive user interface. Node.js and Express.js[9] were instrumental in setting up a robust and secure server-side setup. Prisma ORM[8] allowed for an efficient connection between the server and the database.

The choice of Stripe[10] as the payment platform provided a secure and straightforward checkout process for the users. Finally, the deployment on Railway.app[13](server and database) and Vercel[14] (client) ensured the smooth operation of the application.

16 CONCLUSION

The TasteTrove[1] application demonstrates the successful integration of various technologies to deliver a seamless, intuitive, and secure online shopping experience. Despite the complexity of the project and the challenges encountered, the application was developed and deployed successfully, leveraging a wide array of technologies including React[7], NodeJS, TypeScript, and Stripe[10], among others.

The project not only serves as an efficient and reliable e-commerce platform for spice enthusiasts but also stands as a testament to the possibilities of modern web development technologies. It offers a unique blend of performance, functionality, and user experience, and sets the stage for future enhancements.

17 FUTURE WORK

Looking to the future, there are several exciting opportunities to enhance the user experience on our website. Here are some suggestions for expanding and refining the site's functionality:

- In order to improve user feedback, we can implement more detailed error messages and loading states. By providing more information about what's happening behind the scenes, we can help users understand the status of their requests and reduce frustration.
- To further engage users and encourage them to explore our products, we can add a feature that suggests related items when a user is browsing a particular product page. This would help users discover new products and increase their overall satisfaction with the site.
- In order to expand the range of offerings on our store, we can build out additional pages and expand our product lines to include more collections and suites. This will help us better serve our customers and provide them with a wider range of options to choose from.

18 CHALLENGES AND LEARNING

(1) Stripe[10] integration was challenging. It was crucial to understand the PaymentIntent lifecycle to successfully integrate the payment system.
(2) Implementing authentication and maintaining user session was complex. Learning about JSON web tokens[6], sessions and security practices was essential.
(3) Designing a normalized database schema and managing relations between entities was difficult but it provided a solid understanding of database design.
(4) Deciding the tech stack was an important decision. Learning about different technologies and choosing the most suitable ones was crucial.
(5) Testing and debugging were time-consuming but it led to a better understanding of the application and fixing potential issues.
(6) Managing state and props in React[7] was complex, especially for larger applications. Learning about state management libraries and best practices was beneficial.

18.1 References

REFERENCES

[1] *Taste trove*, GitHub URL: https://github.com/ryannono/TasteTrove-App, Live site URL: https://taste-trove.vercel.app/.
[2] *State of the industry report*, Slash Data ltd., https://slashdata-website-cms.s3.amazonaws.com/sample_reports/_TPqMJKJpsfPe7ph.pdf, 2021.
[3] *SQL vs NoSQL*, https://www.ibm.com/cloud/blog/sql-vs-nosql, June 2022.
[4] *The design and development of an e-commerce web application*, https://www.scirp.org/journal/paperinformation.aspx?paperid=104377, December 2020.
[5] *Tailwind CSS Documentation*, https://tailwindcss.com/docs.
[6] *Understanding JSON Web Tokens*, https://jwt.io/introduction/.
[7] *React Documentation*, https://reactjs.org/docs/getting-started.html, 2023.
[8] *Prisma*, https://www.prisma.io/.
[9] *Express*, https://expressjs.com/.
[10] *Stripe Payment Documentation*, https://stripe.com/docs/payments.
[11] *React Router*, https://reactrouter.com/en/main.
[12] *Axios Documentation*, https://axios-http.com/docs/intro.
[13] *Railway.app Documentation*, https://docs.railway.app/, 2023.
[14] *Vercel Documentation*, https://vercel.com/docs.

[15] *PostgreSQL Documentation*, https://www.postgresql.org/docs/.

A APPENDIX

A.1 Glossary

API	- Application Programming Interface
DRY	- Don't Repeat Yourself
ORM	- Object-Relational Mapping
RESTful	- Representational State Transfer
UI	- User Interface

A.2 Contributors

- Ryan Nono

A.3 Version History

- v1.0: Initial draft
- v1.1: Added details on the database schema
- v1.2: Added the Payment Gateway and Checkout Process section
- v1.3: Added the Results and Discussion sections
- v1.4: Added the Future Work and Challenges and Learning sections
- v1.5: Added the Glossary and References sections
- v1.6: Final draft

Received 02 August 2023