

**BSc (Hons) in Computing
Level C/I/H**



**INDIVIDUAL ASSIGNMENT
LEVEL 5**

COMP50016: SERVER-SIDE-PROGRAMMING-2

Prepared By: [sanadhi karunasekara]

[(CB No.008124)]

Batch code :IF2231SE

Table of Content

| | |
|--|---------------|
| 1.Introduction..... | - 7 - |
| 2.Github Link | - 7 - |
| 3.ABOUT – POINT | - 7 - |
| 4.Mind Map/ER diagram..... | - 8 - |
| 5.Explanation of My solution..... | - 10 - |
| 5.1Technologies I have used | - 10 - |
| 6.Analytics I have implemented in the CRM | - 11 - |
| 7.Cruds I have implemented | - 13 - |
| 8.System Interface..... | - 15 - |
| HomePage..... | - 15 - |
| Login..... | - 16 - |
| Register..... | - 17 - |
| Product Description page..... | - 17 - |
| Filter..... | - 18 - |
| Search bar | - 18 - |
| Cart..... | - 18 - |
| Navigation Bar | - 18 - |
| Product Management | - 19 - |
| Add product..... | - 20 - |
| Category List | - 21 - |
| Add Category | - 21 - |
| Edit Category | - 22 - |
| Stock Mangement | - 22 - |
| Customer Management | - 23 - |

| | |
|---|---------------|
| Address Book..... | - 25 - |
| Purchase history..... | - 25 - |
| Customer pov purchase history..... | - 26 - |
| Checkout Page | - 26 - |
| Profile page..... | - 27 - |
| Dashboard | - 28 - |
| Analytics | - 29 - |
| MailTrap..... | - 30 - |
| Controllers | - 32 - |
| Analytics Controller..... | - 32 - |
| Order Controller..... | - 33 - |
| Cart Controller..... | - 34 - |
| Models | - 35 - |
| Size model..... | - 36 - |
| Colors Model | - 36 - |
| Category Model | - 37 - |
| Order Item Model | - 37 - |
| Cart Item Model | - 38 - |
| Listereners | - 39 - |
| Login Listener | - 39 - |
| Low Stock listener | - 40 - |
| Re-Order Event listener..... | - 41 - |
| Registration Listener | - 42 - |
| Online Registration Listener | - 43 - |
| Checkout Listener..... | - 44 - |
| Product Added to cart listener | - 45 - |

| | |
|--|---------------|
| Events..... | - 46 - |
| Checkout Event..... | - 46 - |
| Low stock event..... | - 47 - |
| Online register event..... | - 48 - |
| Site Register Event | - 49 - |
| Product added to cart event | - 50 - |
| Re-order event..... | - 51 - |
| Notifications | - 52 - |
| Order placed Notification | - 52 - |
| Re-order Notification | - 53 - |
| Low stock Notification | - 54 - |
| Add queue jobs..... | - 55 - |
| Cache Routes | - 56 - |
| Alpine Js..... | - 56 - |
| Chart Js..... | - 59 - |
| Livewire..... | - 60 - |
| Livewire size CRUD | - 61 - |
| Livewire Color CRUD | - 62 - |
| Routes | - 63 - |
| Code structure | - 64 - |
| Migrations | - 66 - |
| MySQL database snapshot..... | - 67 - |
| 10.Test cases..... | - 68 - |
| 11.1.Test case review process | - 71 - |
| 11.2.Deep Analysis of the test cases: | - 72 - |
| 12.Future Upgrade plan | - 73 - |

Table of Figures

| | |
|---|--------|
| Figure 1 Mind Map | - 9 - |
| Figure 2 Homepage | - 15 - |
| Figure 3 Login page | - 16 - |
| Figure 4 Register page..... | - 17 - |
| Figure 5 Product description page..... | - 17 - |
| Figure 6 Filter Options | - 18 - |
| Figure 7 Search functionality | - 18 - |
| Figure 8 Cart..... | - 18 - |
| Figure 9 Navigation bar..... | - 18 - |
| Figure 10 Product Management | - 19 - |
| Figure 11 Product Crud continued..... | - 20 - |
| Figure 12 Category list | - 21 - |
| Figure 13 Edit category | - 22 - |
| Figure 14 Stocks | - 22 - |
| Figure 15 Customer Management..... | - 23 - |
| Figure 16 Add Customer | - 24 - |
| Figure 17 Edit customer..... | - 24 - |
| Figure 18 Success message | - 25 - |
| Figure 19 Address Book | - 25 - |
| Figure 20 Purchase History | - 25 - |
| Figure 21 Purchase history 2..... | - 26 - |
| Figure 22 Checkout Page | - 26 - |
| Figure 23 Profile page | - 27 - |
| Figure 24 Dashboard..... | - 28 - |
| Figure 25 Analytics | - 29 - |
| Figure 26 Mail testing | - 30 - |
| Figure 27 Analytics Controller..... | - 32 - |
| Figure 28 Order Controller..... | - 33 - |
| Figure 29 Cart Controller | - 34 - |
| Figure 30 Product model..... | - 35 - |
| Figure 31 Size model | - 36 - |
| Figure 32 Color model..... | - 36 - |
| Figure 33 Category model..... | - 37 - |
| Figure 34 Order Item model | - 37 - |
| Figure 35 Cart Item Model..... | - 38 - |
| Figure 36 Login listener..... | - 39 - |
| Figure 37 low stock listener | - 40 - |
| Figure 38 Re-order listener..... | - 41 - |
| Figure 39 Registration Listener | - 42 - |
| Figure 40 Site registration listener..... | - 43 - |
| Figure 41 Checkout Listener | - 44 - |
| Figure 42 product added to cart listener..... | - 45 - |
| Figure 43 Checkout event | - 46 - |
| Figure 44 Low stock event | - 47 - |
| Figure 45 Online register event | - 48 - |
| Figure 46 Site register event | - 49 - |

| | |
|--|--------|
| Figure 47 product added to cart event..... | - 50 - |
| Figure 48 Re-order event..... | - 51 - |
| Figure 49 Order placed notification..... | - 52 - |
| Figure 50 Re-order notification..... | - 53 - |
| Figure 51 Low stock notification..... | - 54 - |
| Figure 52 Product hits..... | - 55 - |
| Figure 53 Cache Routes | - 56 - |
| Figure 54 pop up | - 56 - |
| Figure 55 show..... | - 57 - |
| Figure 56 Dropdown | - 57 - |
| Figure 57 Pop up 2 | - 58 - |
| Figure 58 Modal | - 58 - |
| Figure 59 Chart Js | - 59 - |
| Figure 60 Chart js 2..... | - 59 - |
| Figure 61 Livewire product crud | - 60 - |
| Figure 62 livewire Size crud | - 61 - |
| Figure 63 livewire color crud | - 62 - |
| Figure 64 Routes | - 63 - |
| Figure 65 Code structure part 1..... | - 64 - |
| Figure 66 Code structure part 2..... | - 65 - |
| Figure 67 Migrations | - 66 - |
| Figure 68 Database | - 67 - |

1.Introduction

In the dynamic landscape of ecommerce, establishing and maintaining robust customer relationships is paramount for sustained success. Recognizing this, we embarked on a journey to develop a sophisticated Customer Relationship Management (CRM) system tailored for our esteemed shoe-selling ecommerce application, "POINT." Leveraging the power of Laravel PHP, Laravel Jetstream, Alpine.js, Tailwind CSS, and Chart.js, our goal was to create a comprehensive solution that goes beyond traditional transactional interactions.

This report encapsulates the methodologies, technologies, and insights gained throughout the development process, providing an in-depth exploration of the key components and functionalities that shape the POINT CRM. From seamless user authentication to visually engaging data analytics using charts, our approach is rooted in enhancing the overall user experience and empowering our team with actionable insights.

2.Github Link

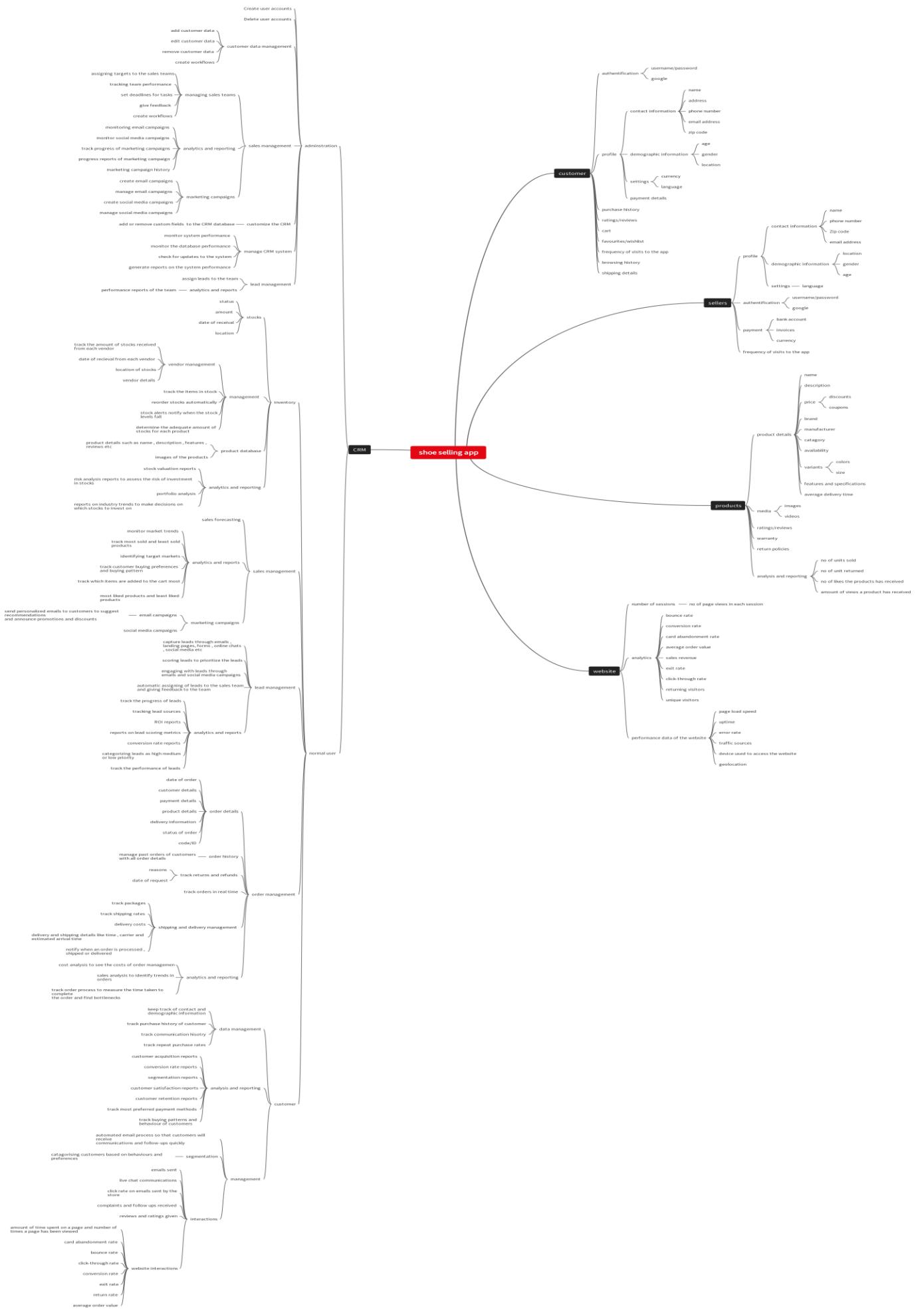
<https://github.com/SanadhiCB008124/crm1.git>

3.ABOUT – POINT

In the heart of Sri Lanka's vibrant e-commerce landscape, POINT emerges as a pioneering online shoe shop, redefining the way Sri Lankans experience and access footwear fashion. With a digital-first approach, POINT stands out as an exclusive online destination for trendy and high-quality shoes. Distinguished by its unique model, POINT operates solely in the virtual realm, without any physical branches, making it a true embodiment of the digital shopping era. Embracing the convenience of online transactions, customers across the island can explore a diverse array of footwear options, from casual sneakers to elegant formal shoes. The seamless user experience is complemented by efficient delivery services that bring the latest in fashion directly to doorsteps. POINT's commitment to providing a hassle-free and trendsetting shopping experience makes it a frontrunner in Sri Lanka's dynamic online retail landscape.

4.Mind Map/ER diagram

Figure 1 Mind Map



5.Explanation of My solution

5.1Technologies I have used

Laravel Jetstream

- Laravel Jetstream is a robust application scaffolding for Laravel that provides the necessary foundation for modern, dynamic web applications. It includes pre-built authentication, which I have successfully used in my project to authenticate users and for access control.

Livewire

- Livewire is a full-stack framework for Laravel that facilitates building dynamic interfaces to create interactive, reactive components using only PHP, enhancing the development speed and maintainability of web applications. I have made use of the above for the product CRUD and the Color and Size CRUDs.

Tailwind CSS

- Tailwind CSS is a utility-first CSS framework that provides low-level utility classes to build designs directly in your markup. It offers a highly customizable and efficient way to create modern and responsive user interfaces, allowing developers to focus on building features rather than wrestling with complex CSS.

Daisy Tailwind UI

- Daisy UI is a collection of components built on top of Tailwind CSS. It extends the capabilities of Tailwind by providing a set of customizable and ready-to-use UI components. This simplifies the process of creating consistent and visually appealing user interfaces.

Alpine.Js

- Alpine.js is a minimal JavaScript framework designed for building frontend components with minimal setup. It's particularly useful for enhancing interactivity in web applications without the need for a more extensive JavaScript framework. Alpine.js seamlessly integrates with Laravel and Livewire in this project for the purpose of pop-ups , confirmation messages and dropdowns. All these increase customer interaction and simplifies the process for the customer.

Chart.Js

- Chart.js is a popular JavaScript library for creating interactive and visually appealing charts and graphs. It supports a variety of chart types and is easy to integrate into web applications. Chart.js is used to visualize data and analytics within my CRM system.

Mailtrap

- Mailtrap is a service that provides a simulated SMTP server for testing email functionality in development environments. I have made use of mailtrap to send order and re-order confirmations and low stock alert emails.

MySql Database

- MySQL is the database management system I have used to hold and retrieve data for the CRM .

6.Analytics I have implemented in the CRM

What is the importance of Analytics ?

In the dynamic landscape of modern commerce, the significance of analytics in fortifying business intelligence and facilitating informed decision-making cannot be overstated. Analytics serves as the compass guiding enterprises through the vast sea of data generated in every transaction, user interaction, and operational process. By harnessing the power of analytics, businesses gain the invaluable ability to discern patterns, uncover trends, and extract actionable insights from the troves of information at their disposal.

A. Financial Metrics

Average Order Value

- Definition: The average value of orders placed.
- Implementation: Calculation based on total revenue and the number of orders.

Product Revenue

- Definition: Revenue generated from shoe products.
- Implementation: Calculation based on individual product sales.

COGS (Cost of Goods Sold)

- Definition: The direct costs associated with producing shoes.
- Implementation: Calculation based on the cost of inventory.

Profit Margin

- Definition: The percentage of profit from sales.
- Implementation: Calculated using the profit and revenue.

B. Operational Metrics

Checkouts Per Day, For the past week and Month

- Definition: The number of completed checkouts.
- Implementation: Daily , Weekly and monthly counts of completed checkouts.

Most Added to Cart

- Definition: Top 5 products added to the cart.
- Implementation: Analysis of cart event data.

Logins Per Day

- Definition: Daily count of user logins.
- Implementation: Log and analyze login events.

C. User Engagement Metrics

Total Site Registrations

- Definition: The total number of customers.
- Implementation: Count of registration events .

Total Site Registrations per day , For the past week and Past month.

- Definition: The total number of users registering on the site.
- Implementation: Count of registration events .

Total Online Registrations per day , For the past week and Past month.

- Definition: Registrations done through the online platform.
- Implementation: Count of online registration events.

Total Checkouts

- Definition: Overall count of successful checkouts.
- Implementation: Count of checkout events.

Total Orders

- Definition: The cumulative number of orders placed.
- Implementation: Count of order events.

Least Added Product to Cart

- Definition: The product with the lowest cart additions.
- Implementation: Analysis of cart event data.

D. Product Sales Metrics

Most Sold Colors, Categories, and Sizes

- Definition: Identification of top-performing colors, categories, and sizes.
- Implementation: Analysis of order item data.

E. User Behavior Metrics

Logins Per Day

- Definition: Daily count of user logins.
- Implementation: Log and analyze login events.

Average Session Rate

- Definition: The average duration of user sessions.
- Implementation: Analysis of session data.

Abandoned Cart Rate

- Definition: Percentage of initiated carts that were not checked out.
- Implementation: Calculation based on cart and checkout events.

F. Inventory Metrics

Inventory Turnover Rate

- Definition: The rate at which inventory is sold and replaced.
- Implementation: Calculation based on sales and stock levels.

7.Cruds I have implemented

7.1Product CRUD using Livewire:

- Effortless Product Management: The Product CRUD system seamlessly integrates Livewire to provide an intuitive and efficient way to manage products within the e-commerce platform.
- Dynamic Updates: Livewire enables real-time updates without the need for page refreshes, ensuring a dynamic and responsive user experience.
- User-Friendly Interface: The CRUD operations—Create, Read, Update, and Delete—are encapsulated in a user-friendly interface, streamlining product management tasks.
- Image Handling: Livewire simplifies image uploads and displays, enhancing the visual representation of products in the system.
- Validation and Error Handling: Livewire facilitates client-side validation and error handling, ensuring data integrity and a smooth user interaction.

7.2Size CRUD using Livewire:

- Seamless Size Management: Livewire is harnessed to create a Size CRUD system, providing a seamless way to manage different sizes for products.
- Responsive Interface: Livewire's real-time updates contribute to a responsive interface, allowing users to add, edit, and remove sizes without cumbersome page reloads.
- Intuitive User Interactions: The Livewire-powered Size CRUD ensures that users can easily interact with size-related data, promoting a straightforward user experience.
- Data Integrity: Livewire assists in implementing validation mechanisms, guaranteeing the accuracy and integrity of size-related information in the system.
- Efficient Size Editing: Livewire's capabilities make size editing a breeze, offering an efficient solution for adapting to changes in product size requirements.

7.3.Color CRUD using Livewire:

- Dynamic Color Management: Leveraging Livewire, the Color CRUD system facilitates the dynamic management of colors associated with products.
- Real-Time Updates: Livewire's ability to update content in real-time enhances the user experience, especially when modifying or adding new colors.
- User-Friendly Interaction: The Color CRUD interface, powered by Livewire, ensures that users can interact with color data in an intuitive and user-friendly manner.
- Validation Support: Livewire assists in implementing validation rules, ensuring that color data remains consistent and error-free.
- Streamlined Color Editing: Livewire simplifies the process of editing and updating color information, contributing to the overall efficiency of color management within the e-commerce system.

7.4.Customer CRUD:

- Comprehensive Customer Management: The Customer CRUD system provides a comprehensive solution for managing customer data within the e-commerce platform.
- Traditional CRUD Operations: Through traditional CRUD operations—Create, Read, Update, and Delete—the system allows for the seamless handling of customer information.
- Data Accuracy: The Customer CRUD ensures the accuracy of customer details, facilitating error-free interactions and transactions.
- Efficient Editing: Users can efficiently edit and update customer information, contributing to a smooth customer relationship management process.
- Accessibility: The Customer CRUD, devoid of Livewire, maintains accessibility across various platforms and devices, offering a consistent experience for administrators.

7.5.Category CRUD:

- Structured Category Management: The Category CRUD functionality offers a structured approach to managing product categories, organizing the inventory effectively.
- Classic CRUD Functionality: Utilizing traditional CRUD principles, the system enables administrators to perform essential category management tasks.
- Organized Product Categorization: The Category CRUD system plays a vital role in organizing and categorizing products, enhancing the overall user experience.
- Simplified Category Editing: Users can easily edit and update category details, ensuring that changes reflect accurately in the product catalog.
- Compatibility: The Category CRUD, developed without Livewire, maintains compatibility with various technologies and frameworks, ensuring flexibility in integration with existing systems.

8.System Interface

HomePage

The screenshot shows a website interface with the following elements:

- Header:** A navigation bar at the top with categories: Home, Mens, Womens, Kids, Sports. It also includes a search bar, user icons (notification, heart, profile), and a Log Out button.
- Main Image:** A large, dark photograph of a person's legs and feet wearing glowing sneakers. The glowing effect creates a trail of light on the ground.
- Fall Collection:** A section titled "Fall Collection" with the subtext "Check out our latest collections and shop your heart out". It features a "Shop Collection" button and a grid of nine smaller images showing various sneaker models and people wearing them.
- More to Explore:** A section titled "More to Explore" with three categories: Womens, Sports, and Mens. Each category has a representative image and a small circular button below it labeled with the category name.
- Footer:** A dark footer bar with social media icons (Facebook, Instagram, Twitter) and links to About, Privacy Policy, Terms & Conditions, and Contact Us.

Figure 2 Homepage

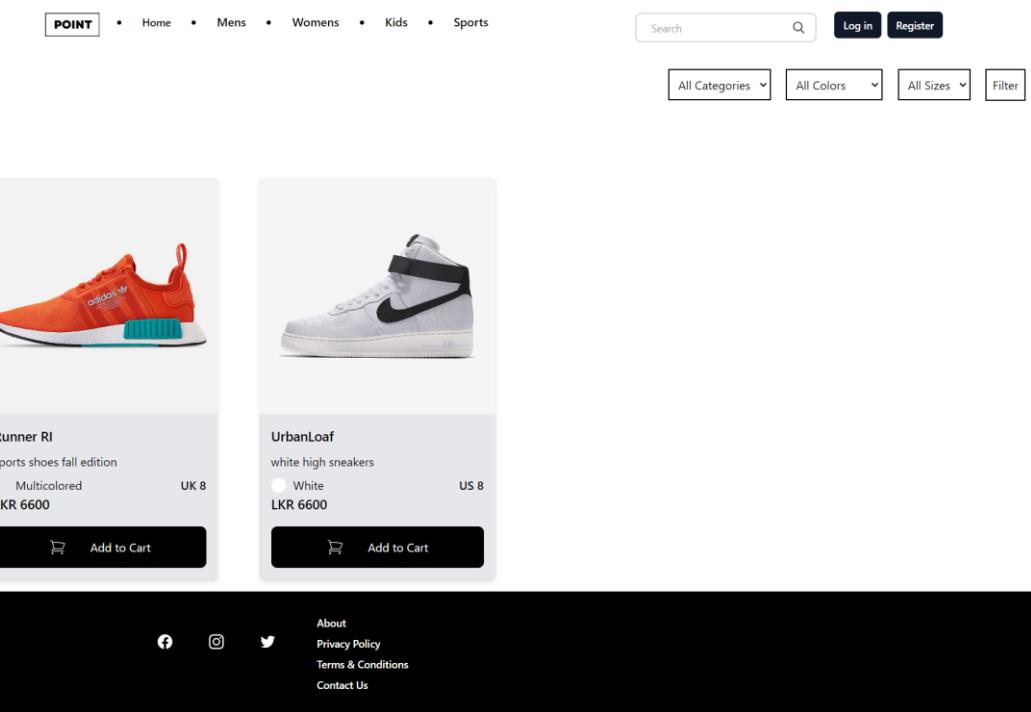


Figure 3 Mens'Products

Login

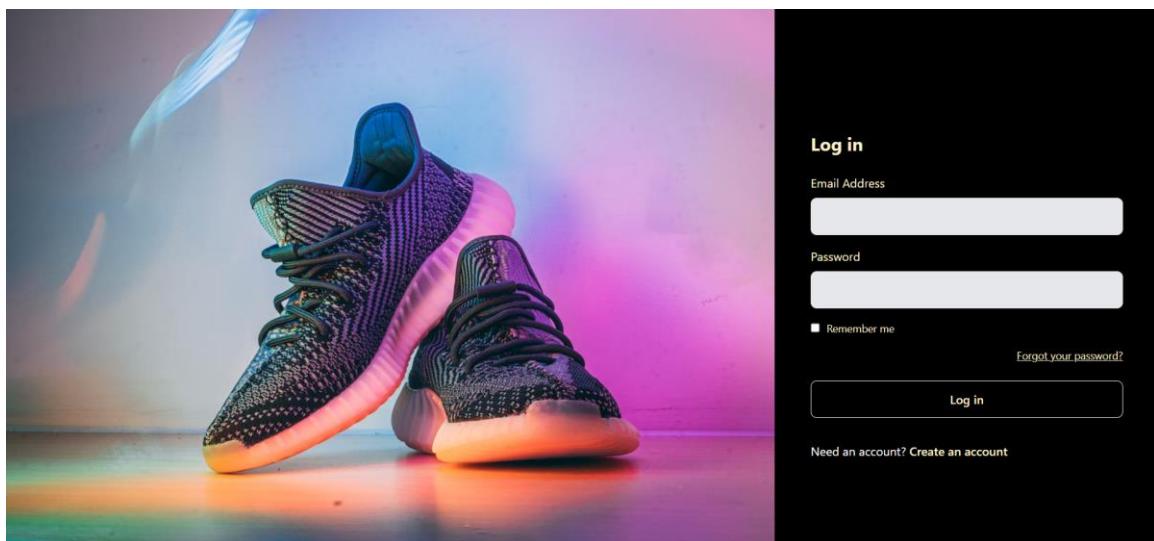


Figure 3 Login page

Register



The image shows a pair of sneakers with glowing orange soles and blue laces, set against a background of blurred lights in shades of pink, purple, and blue.

Register

Name:

Email:

Contact No:

Address:

Password:

Confirm Password:

[Already registered?](#)

[Register](#)

Figure 4 Register page

Product Description page

POINT • Home • Mens • Womens • Kids • Sports

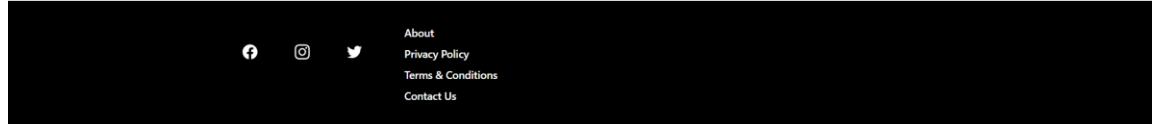
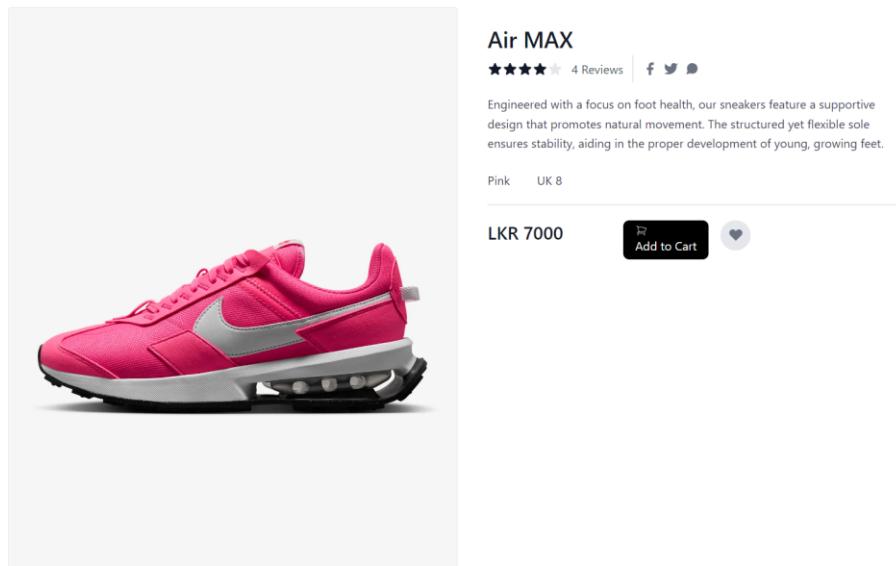


Figure 5 Product description page

Filter



Figure 6 Filter Options

Search bar



Figure 7 Search functionality

Cart

A screenshot of the shopping cart page. It shows two items in the cart: 'Chasker Boots' (Quantity: 2, LKR 4000) and 'Runner RI' (Quantity: 1, LKR 6600). To the right is a 'Purchase Resume' table with the following data:

| Subtotal | 14600 LKR |
|-----------------|------------------------------|
| Shipping Cost | 200 LKR Arrives on Jul 16 |
| Taxes | 15 LKR |
| Discount Coupon | 1460 LKR |
| Total | 13355 LKR |

At the bottom are 'Go to Checkout' and 'Continue Shopping' buttons.

Figure 8 Cart

Navigation Bar



Figure 9 Navigation bar

Product Management

Welcome sanadhi2002@gmail.com !



Dashboard

- [Product Management](#)
- [Stocks](#)
- [Customer Management](#)
- [Address Book](#)
- [Purchase history](#)
- [Analytics](#)
- [Profile](#)
- [Log Out](#)

Categories

Colors Manager

- Red X
- Blue X
- Green X
- Pink X
- Black X
- White X
- Multicolored X

Size Manager

UK 6 X UK 4 X UK 8 X UK 10 X US 4 X US 6 X US 8 X US 10 X

Create New Product

| No. | Title | Category | description | Price | Size | Color | Stocks | Image | Action |
|-----|------------------|----------|---|-------|------|--------------|--------|--|---|
| 1 | Chasker Boots | Womens | best boots of the year | 4000 | UK 6 | Blue | 44 |  | <button>Edit</button> <button>Delete</button> |
| 2 | Ubersonic | Sports | ubersonic | 5000 | US 4 | White | 22 |  | <button>Edit</button> <button>Delete</button> |
| 3 | Runner RI | Mens | sports shoes fall edition | 6600 | UK 8 | Multicolored | 8 |  | <button>Edit</button> <button>Delete</button> |
| 4 | Air MAX | Kids | Engineered with a focus on foot health, our sneakers feature a supportive design that promotes natural movement. The structured yet flexible sole ensures stability, aiding in the proper development of young, growing feet. | 7000 | UK 8 | Pink | 59 |  | <button>Edit</button> <button>Delete</button> |
| 5 | Velocity Strides | Mens | New collection 2023 | 4000 | UK 4 | White | 21 |  | <button>Edit</button> <button>Delete</button> |
| 6 | UrbanLoaf | Mens | white high sneakers | 6600 | US 8 | White | 33 |  | <button>Edit</button> <button>Delete</button> |

Figure 10 Product Management

Add product

The screenshot shows a 'Product Crud' application interface. At the top, there is a header with the title 'Add product'. Below the header is a large, light-gray rectangular input area divided into two main sections by a vertical line. Each section contains several input fields:

- Name:**
- Category:**
- Detail:**
- Price:**
- Color:**
- Detail:**
- Price:**
- Color:**
- Size:**
- No of stocks arrived:**
- Image:** No file chosen

Figure 11 Product Crud continued

Category List

Welcome sanadhi2002@gmail.com !



Add Category

| ID | Name | Edit | Delete |
|----|--------|----------------------|------------------------|
| 1 | Mens | Edit | Delete |
| 2 | Womens | Edit | Delete |
| 3 | Kids | Edit | Delete |
| 4 | Sports | Edit | Delete |

Dashboard

Product Management

Stocks

Customer Management

Address Book

Purchase history

Analytics

Profile

[Log Out](#)

Figure 12 Category list

Add Category

Welcome sanadhi2002@gmail.com !



Dashboard

Product Management

Stocks

Customer Management

Address Book

Purchase history

Analytics

Profile

[Log Out](#)

Name

[Save](#) [Back](#)

Edit Category

Welcome sanadhi2002@gmail.com !



Dashboard
Product Management
Stocks
Customer Management
Address Book
Purchase history
Analytics
Profile
[Log Out](#)

Name
Mens

Update **Back**

Figure 13 Edit category

Stock Management

Welcome sanadhi2002@gmail.com !



0.052306574645841%
Inventory Turnover Rate

Dashboard

Product Management

Stocks

Customer Management

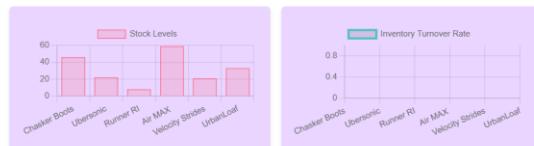
Address Book

Purchase history

Analytics

Profile

[Log Out](#)



Search Stocks...

| No. | Title | Category | Stocks | Image |
|-----|------------------|----------|-----------------------|-------|
| 1 | Chasker Boots | Womens | 46 | |
| 2 | Ubersonic | Sports | 22 | |
| 3 | Runner RI | Mens | 8 Low Stock Alert! | |
| 4 | Air MAX | Kids | 59 | |
| 5 | Velocity Strides | Mens | 21 | |
| 6 | UrbanLoaf | Mens | 33 | |

Figure 14 Stocks

Customer Management

Welcome sanadhi2002@gmail.com !



Register Customer

Search Customers...🔍

| Dashboard | ID | Name | Email | Contact | Address | Edit | Delete |
|---------------------|----|------------------|---------------------------|-----------------|---|--|--|
| Product Management | 2 | Sybil Williamson | xaxop@mailinator.com | 075 246 234 | 123 Elm Street, Springfield, IL 62701 | Update | Delete |
| Stocks | 3 | Wade Klein | vomewomomi@mailinator.com | 077 364 567 | No 26 niwasa mawatha Kandana | Update | Delete |
| Customer Management | 4 | Vielka Johnston | biwe@mailinator.com | 94 345 672 112 | 456 Oak Avenue, Pleasantville, CA 90210 | Update | Delete |
| Address Book | 5 | Anika Andrews | qucog@mailinator.com | 077 156 4623 | 789 Maple Lane, Rivertown, NY 10001 | Update | Delete |
| Purchase history | 6 | Blair Beard | gureto@mailinator.com | +94 76 234 5678 | 101 Pine Road, Greenfield, TX 75001 | Update | Delete |
| Analytics | 7 | Stewart Nguyen | nidubeq@mailinator.com | 076 125 0374 | 333 Cedar Court, Hilltop, AZ 85001 | Update | Delete |
| Profile | 8 | Dalton Wolf | bapa@mailinator.com | 077 375 2261 | No 34 , fonda avenue , maybelene | Update | Delete |
| Log Out | 9 | Florence Parks | tykyfa@mailinator.com | 078 654 2344 | Building 23, apartment gold | Update | Delete |

Figure 15 Customer Management

Add Customer

Name

Email

Contact No:

Address:

Save

Back

Figure 16 Add Customer

Update Customer

Name

Sybil Williamson

email

xaxop@mailinator.com

Contact No:

075 246 234

Address:

123 Elm Street, Springfield, IL 62701

Update

Back

Figure 17 Edit customer

customer added successfully

Add Customer

Figure 18 Success message

Address Book

Welcome sanadhi2002@gmail.com !



Search Customers... Q

| No. | Name | email | Address | Number |
|-----|------------------|---------------------------|---|-----------------|
| 2 | Sybil Williamson | xaxop@mailinator.com | 123 Elm Street, Springfield, IL 62701 | 075 246 234 |
| 3 | Wade Klein | vomewomomi@mailinator.com | No 26 niwasa mawatha Kandana | 077 364 567 |
| 4 | Vielka Johnston | biwe@mailinator.com | 456 Oak Avenue, Pleasantville, CA 90210 | 94 345 672 112 |
| 5 | Anika Andrews | qucog@mailinator.com | 789 Maple Lane, Rivertown, NY 10001 | 077 156 4623 |
| 6 | Blair Beard | gureto@mailinator.com | 101 Pine Road, Greenfield, TX 75001 | +94 76 234 5678 |
| 7 | Stewart Nguyen | nidubeq@mailinator.com | 333 Cedar Court, Hilltop, AZ 85001 | 076 125 0374 |
| 8 | Dalton Wolf | bapa@mailinator.com | No 34 , fonda avenue , maybelene | 077 375 2261 |
| 9 | Florence Parks | tylyfa@mailinator.com | Building 23, apartment gold | 078 654 2344 |

[Dashboard](#) [Product Management](#) [Stocks](#) [Customer Management](#) [Address Book](#) [Purchase history](#) [Analytics](#) [Profile](#) [Log Out](#)

Figure 19 Address Book

Purchase history

Welcome sanadhi2002@gmail.com !



Total Orders : 4

Order Number: Status: Payment Status: Date of Order: Payment Method:

| Order Number | Order Details | Customer | Order Status | Total | Payment Method | Payment Status | Biller Name | Shipping Address | Billing Address | Contact | Data of order |
|-------------------|---------------|-------------|---|---------|-------------------|---|----------------|----------------------------------|----------------------------------|---------------------|---------------------|
| ORD-6513BD4BE16C3 | Details | Wade Klein | pending | 405 LKR | Debit/Credit Card | Processed | Leandra Ford | Nihil odit aliquip a | Voluptatem numquam | Placeat quo veniam | 2023-09-27 05:27:39 |
| ORD-651870F0C4437 | Details | Blair Beard | pending | 405 LKR | Cash | Processed | Oprah Hunter | Eaque doloribus nisi | Iste aliquid sed ali | Cum tempor et iusto | 2023-09-30 19:03:12 |
| ORD-6519020A372AF | Details | Dalton Wolf | pending | 405 LKR | MasterCard | Processed | Dalton Wolf | No 34 , fonda avenue , maybelene | No 34 , fonda avenue , maybelene | 077 375 2261 | 2023-10-01 05:22:18 |
| ORD-6519027929B62 | Details | Dalton Wolf | pending | 405 LKR | MasterCard | Processed | Kevyn Griffith | Sit deserunt do nec | Voluptatum officiis | Sit mollit aut opti | 2023-10-01 05:24:09 |

[Dashboard](#) [Product Management](#) [Stocks](#) [Customer Management](#) [Address Book](#) [Purchase history](#) [Analytics](#) [Profile](#) [Log Out](#)

Figure 20 Purchase History

Customer pov purchase history

| Order Number | Order Details | Customer | Order Status | Total | Payment Method | Payment Status | Biller Name | Shipping Address | Billing Address | Contact | Data of order |
|-------------------|---------------|-------------|--------------|---|----------------|----------------|----------------|----------------------------------|----------------------------------|---------------------|---------------------|
| ORD-6519020A372AF | Details | Dalton | pending | 405 LKR | MasterCard | Processed | Dalton Wolf | No 34 , fonda avenue , maybelene | No 34 , fonda avenue , maybelene | 077 375 2261 | 2023-10-01 05:22:18 |
| | | | | Product Name: UrbanLoaf* 1 UNIT PRICE : 6600 LKR | | | | | | | |
| | | | | | | | | | | | |
| ORD-6519027929B62 | Details | Dalton Wolf | pending | 405 LKR | MasterCard | Processed | Kevyn Griffith | Sit deserunt do nec | Voluptatum officiis | Sit mollit aut opti | 2023-10-01 05:24:09 |
| | | | | | | | | | | | |



Figure 21 Purchase history 2

Checkout Page

Complete your checkout

First name

Last name

Special notes

Shipping address

Billing address

Contact No

Select payment method

Cash Debit/Credit Card Voucher Paypal MasterCard

Confirm Order

Continue shopping

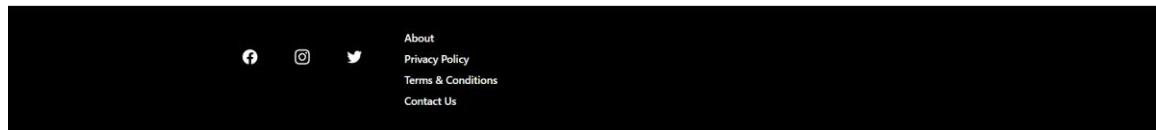


Figure 22 Checkout Page

Profile page

The screenshot shows a user's profile page with the following sections:

- Profile Information:** Allows updating Name (Sybil Williamson), Email (xaxop@mailinator.com), Contact (075 246 234), and Address (123 Elm Street, Springfield, IL 62701). A **SAVE** button is present.
- My Orders:** A section showing order history.
- Update Password:** Fields for Current Password, New Password, and Confirm Password. A **SAVE** button is present.
- Two Factor Authentication:** A message stating "You have not enabled two factor authentication." It explains the process and has an **ENABLE** button.
- Browser Sessions:** A list showing "Windows - Chrome 127.0.0.1, This device". A **LOG OUT OTHER BROWSER SESSIONS** button is present.
- Delete Account:** A message explaining account deletion. A **DELETE ACCOUNT** button is present.

At the bottom, there is a footer with social media links (Facebook, Instagram, Twitter) and links to About, Privacy Policy, Terms & Conditions, and Contact Us.

Figure 23 Profile page

Dashboard

Welcome sanadhi2002@gmail.com !



STATISTICS

Dashboard

Product Management

Stocks

Customer Management

Address Book

Purchase history

Analytics

Profile

[Log Out](#)

| | | | | | |
|--|---------------------------------------|--|-----------------------------------|--|--|
| | Users 8 | | Orders 4 | | Revenue LKR 1,620.00 |
| | Cost of Goods LKR 28,800.00 | | Profit Margin -1,677.78 | | Average Order value LKR 405.00 |

Figure 24 Dashboard

Analytics

Welcome sandhu2002@gmail.com !



Analytics

Dashboard

Product Management

Stocks

Customer Management

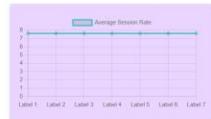
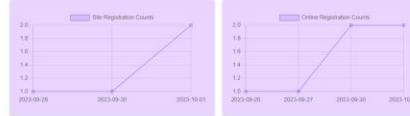
Address Book

Purchase history

Analytics

Profile

[Log Out](#)



Average Session Rate: 7.6156 minutes
Abandoned Cart Rate: 68.75%
Total Registration count: 10
Logins per day: 5

Last 7 days
Last Month

Onsite Registration count for today: 3

Last 7 days
Last Month

Website Registration count for today: 4

Last 7 days
Last Month

Checkouts for Today: 3

Product Analytics

Most Added-to-Cart Item



Top 5 Most Sold Products

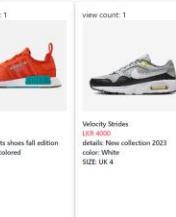
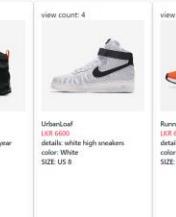


Product Hits

| DATE | VIEWS |
|------------|-------|
| 2023-09-26 | 31 |
| 2023-10-01 | 5 |



Top 5 Most Viewed Products



Least Viewed product



Figure 25 Analytics

MailTrap

The screenshot shows the MailTrap web application interface. On the left is a sidebar with navigation links: Home, Email Testing, Inboxes (selected), Email Sending, Email Marketing, Billing, Settings, and Help. The main area displays an inbox with several messages. One message is selected, showing its details:

Order Placed
From: Example <heilo@example.com>
To: <gureto@mailinator.com>
an hour ago
Show Headers

Message Content:

Hello, Blair Beard
Your order has been successfully placed.
Order Number: ORD-651870F0C4437
Total Amount: 405
your order will be delivered within 7 days.

Buttons: View Order

Header: Sanadhi Karunasekara, 2023-09-30 19:03, 13 KB

Figure 26 Mail testing

Controllers

Analytics Controller

Figure 27 Analytics Controller

Order Controller

```

    }

    @Test
    public void testAddProduct() {
        // Given
        String productId = "1";
        String productName = "SmartPhone";
        String quantity = "10";
        String unitPrice = "1000";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product(productId, productName, quantity, unitPrice, discount, status, description, image, category, brand, model);

        // When
        String response = mockMvc.perform(post("/api/products")
                .contentType(MediaType.APPLICATION_JSON)
                .content(asJsonString(product)))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Product added to cart.", response);
    }

    @Test
    public void testGetProductById() {
        // Given
        String productId = "1";
        String productName = "SmartPhone";
        String quantity = "10";
        String unitPrice = "1000";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product(productId, productName, quantity, unitPrice, discount, status, description, image, category, brand, model);

        // When
        String response = mockMvc.perform(get("/api/products/{id}", productId))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Product found.", response);
        assertEquals("SmartPhone", response);
        assertEquals("10", response);
        assertEquals("1000", response);
        assertEquals("0.05", response);
        assertEquals("In Stock", response);
        assertEquals("A new smartphone with a sleek design and powerful performance.", response);
        assertEquals("https://example.com/images/smartphone.jpg", response);
        assertEquals("Electronics", response);
        assertEquals("Samsung", response);
        assertEquals("Galaxy S20+", response);
    }

    @Test
    public void testUpdateProduct() {
        // Given
        String productId = "1";
        String productName = "SmartPhone";
        String quantity = "10";
        String unitPrice = "1000";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product(productId, productName, quantity, unitPrice, discount, status, description, image, category, brand, model);

        // When
        String response = mockMvc.perform(patch("/api/products/{id}", productId)
                .contentType(MediaType.APPLICATION_JSON)
                .content(asJsonString(product)))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Product updated successfully.", response);
    }

    @Test
    public void testDeleteProduct() {
        // Given
        String productId = "1";
        String productName = "SmartPhone";
        String quantity = "10";
        String unitPrice = "1000";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product(productId, productName, quantity, unitPrice, discount, status, description, image, category, brand, model);

        // When
        String response = mockMvc.perform(delete("/api/products/{id}", productId))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Product removed from cart.", response);
    }

    @Test
    public void testGetCart() {
        // Given
        String userId = "1";
        String quantity = "10";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product("1", "SmartPhone", "10", "1000", "0.05", "In Stock", "A new smartphone with a sleek design and powerful performance.", "https://example.com/images/smartphone.jpg", "Electronics", "Samsung", "Galaxy S20+");

        // When
        String response = mockMvc.perform(get("/api/cart"))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Cart contents: 1 SmartPhone (10 units) at $1000 each with a 5% discount.", response);
        assertEquals("1 SmartPhone", response);
        assertEquals("10", response);
        assertEquals("1000", response);
        assertEquals("0.05", response);
        assertEquals("In Stock", response);
        assertEquals("A new smartphone with a sleek design and powerful performance.", response);
        assertEquals("https://example.com/images/smartphone.jpg", response);
        assertEquals("Electronics", response);
        assertEquals("Samsung", response);
        assertEquals("Galaxy S20+", response);
    }

    @Test
    public void testIncreaseQuantity() {
        // Given
        String userId = "1";
        String productId = "1";
        String quantity = "10";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product("1", "SmartPhone", "10", "1000", "0.05", "In Stock", "A new smartphone with a sleek design and powerful performance.", "https://example.com/images/smartphone.jpg", "Electronics", "Samsung", "Galaxy S20+");

        // When
        String response = mockMvc.perform(post("/api/cart/increase/{product_id}/{quantity}", productId, quantity))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Quantity increased.", response);
    }

    @Test
    public void testDecreaseQuantity() {
        // Given
        String userId = "1";
        String productId = "1";
        String quantity = "10";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product("1", "SmartPhone", "10", "1000", "0.05", "In Stock", "A new smartphone with a sleek design and powerful performance.", "https://example.com/images/smartphone.jpg", "Electronics", "Samsung", "Galaxy S20+");

        // When
        String response = mockMvc.perform(post("/api/cart/decrease/{product_id}/{quantity}", productId, quantity))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Quantity decreased.", response);
    }

    @Test
    public void testEmptyCart() {
        // Given
        String userId = "1";
        String quantity = "10";
        String discount = "0.05";
        String status = "In Stock";
        String description = "A new smartphone with a sleek design and powerful performance.";
        String image = "https://example.com/images/smartphone.jpg";
        String category = "Electronics";
        String brand = "Samsung";
        String model = "Galaxy S20+";

        Product product = new Product("1", "SmartPhone", "10", "1000", "0.05", "In Stock", "A new smartphone with a sleek design and powerful performance.", "https://example.com/images/smartphone.jpg", "Electronics", "Samsung", "Galaxy S20+");

        // When
        String response = mockMvc.perform(delete("/api/cart"))
                .andExpect(status().isOk())
                .andReturn().getResponse().getContentAsString();

        // Then
        assertEquals("Cart is now empty.", response);
    }
}

```

Figure 28 Order Controller

Cart Controller

Figure 29 Cart Controller

Models

Product Model



The screenshot shows a code editor window with a dark theme. The title bar says "Product Model". The code is written in PHP and defines a Product model. It includes imports for Illuminate\Database\Eloquent\Factories\HasFactory, Illuminate\Database\Eloquent\Model, Illuminate\Support\Str, and Ramsey\Uuid\Type\Integer. The Product class extends Model and uses HasFactory. It has a protected \$fillable array with fields: name, detail, unit_price, image, category_id, stocks, slug, color_id, size_id, and views. A static booted() method sets up creating and updating events to generate slugs. It also has methods for category, color, size, and orderItems.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Str;
use Ramsey\Uuid\Type\Integer;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'name', 'detail', 'unit_price', 'image', 'category_id', 'stocks', 'slug', 'color_id', 'size_id', 'views'
    ];

    protected static function booted()
    {
        parent::booted();

        static::creating(function ($product) {
            $product->slug = Str::slug($product->name);
        });

        static::updating(function ($product) {
            $product->slug = Str::slug($product->name);
        });
    }

    public function category()
    {
        return $this->belongsTo(Category::class, 'category_id');
    }
    public function color()
    {
        return $this->belongsTo(Color::class, 'color_id');
    }
    public function size(){
        return $this->belongsTo(Size::class, 'size_id');
    }

    public function orderItems()
    {
        return $this->hasMany(OrderItem::class, 'product_slug', 'slug');
    }
}
```

codesnap.dev

Figure 30 Product model

Size model



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Size extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
    ];
}
```

codesnap.dev

Figure 31 Size model

Colors Model



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Color extends Model
{
    use HasFactory;

    protected $fillable=[

        'color'
    ];
}
```

codesnap.dev

Figure 32 Color model

Category Model



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    use HasFactory;

    protected $fillable = [
        'category_name',
    ];
}
```

codesnap.dev

Figure 33 Category model

Order Item Model



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class OrderItem extends Model
{
    use HasFactory;
    protected $fillable = ['order_id', 'product_slug', 'quantity'];

    public function order()
    {
        return $this->belongsTo(Order::class);
    }

    public function product()
    {
        return $this->belongsTo(Product::class, 'product_slug', 'slug');
    }
}
```

codesnap.dev

Figure 34 Order Item model

Cart Item Model



A screenshot of a code editor window titled "Cart Item Model". The code is written in PHP and defines a model for a shopping cart item. It includes imports for the Illuminate\Database\Eloquent\Factories\HasFactory and Illuminate\Database\Eloquent\Model traits. The CartItem class extends Model and uses HasFactory. It has a protected \$fillable array containing 'user_id', 'product_slug', and 'quantity'. A public product() method returns a belongsTo relationship with the Product model based on the product_slug and slug fields.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class CartItem extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'product_slug',
        'quantity',
    ];

    public function product()
    {
        return $this->belongsTo(Product::class, 'product_slug', 'slug');
    }
}
```

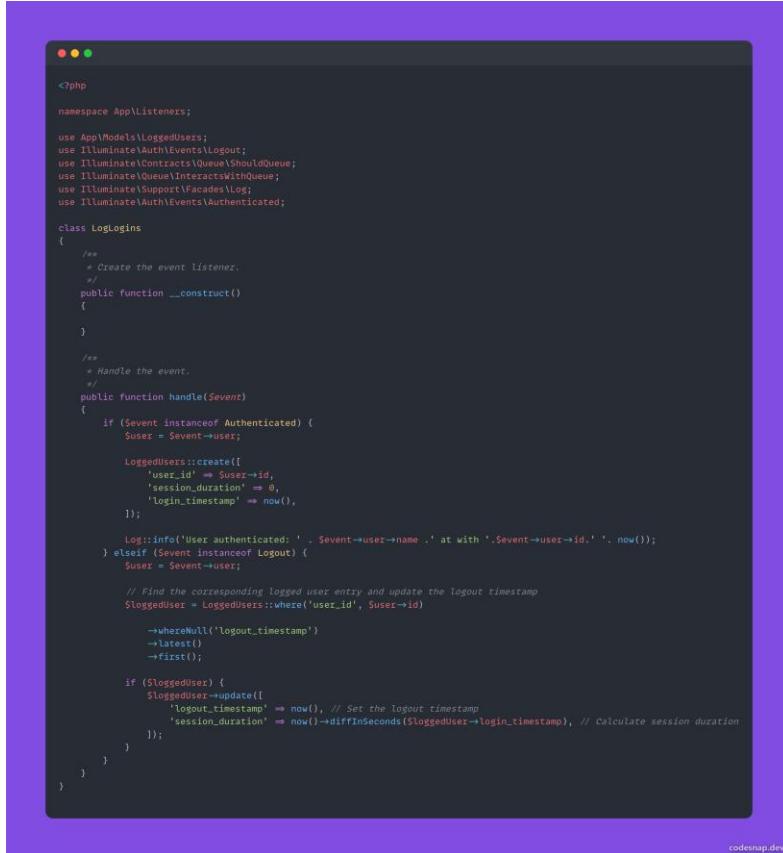
codesnap.dev

Figure 35 Cart Item Model

Listeners

Login Listener

This listener listens to the login activity which helps us to get to get data on logins.



The screenshot shows a terminal window with a dark background and white text. The code is a PHP class named LogLogins, which implements the Illuminate\Contracts\Queue\ShouldQueue interface. It handles the 'Auth\\Events\Authenticated' event to log user login information. It also handles the 'Auth\\Events\Logout' event to update the logout timestamp and calculate session duration. The code uses Eloquent models for User and LoggedUser, and the Log facade for logging.

```
<?php

namespace App\Listeners;

use App\Models\LoggedUsers;
use Illuminate\Auth\Events\Logout;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;
use Illuminate\Auth\Events\Authenticated;

class LogLogins
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {
    }

    /**
     * Handle the event.
     */
    public function handle(Sevent)
    {
        if ($event instanceof Authenticated) {
            $user = $event->user;

            Loggedusers::create([
                'user_id' => $user->id,
                'session_duration' => 0,
                'login_timestamp' => now(),
            ]);

            Log::info('User authenticated: ' . $event->user->name . ' at with ' . $event->user->id . ' ' . now());
        } elseif ($event instanceof Logout) {
            $user = $event->user;

            // Find the corresponding logged user entry and update the logout timestamp
            $loggedUser = Loggedusers::where('user_id', $user->id)
                ->whereNull('logout_timestamp')
                ->latest()
                ->first();

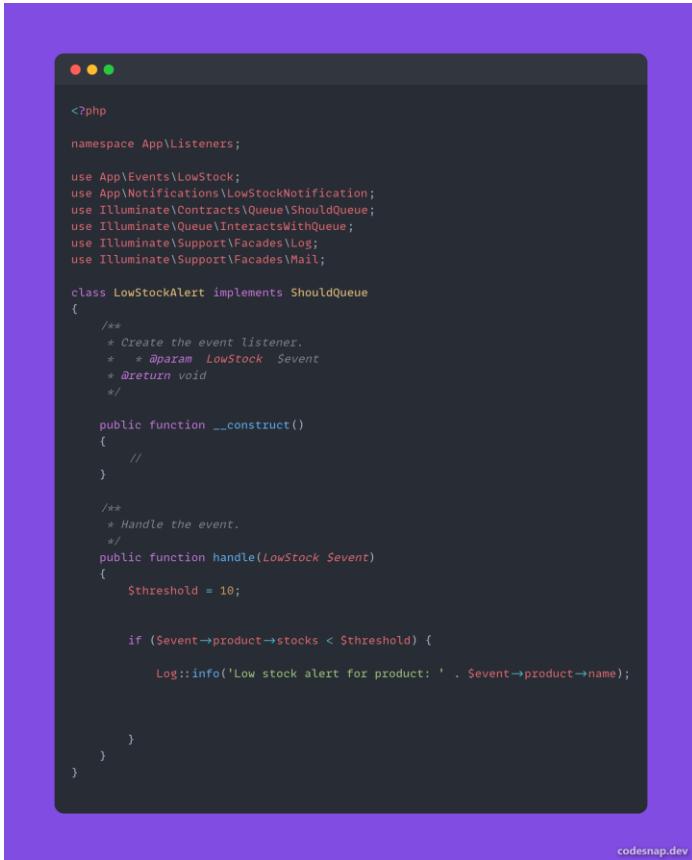
            if ($loggedUser) {
                $loggedUser->update([
                    'logout_timestamp' => now(), // Set the logout timestamp
                    'session_duration' => now()->diffInSeconds($loggedUser->login_timestamp), // Calculate session duration
                ]);
            }
        }
    }
}
```

codesnap.dev

Figure 36 Login listener

Low Stock listener

This listener listens if the stocks of any product falls below 10. If so , an notification is sent about this alert.



A screenshot of a code editor displaying a PHP file. The file contains a class named 'LowStockAlert' which implements the 'ShouldQueue' interface. The class has a constructor and a handle method. The handle method checks if a product's stock is less than a threshold (set to 10) and logs an info message if it is. The code is annotated with PHPDoc comments explaining the purpose of the class and its methods.

```
<?php

namespace App\Listeners;

use App\Events\LowStock;
use App\Notifications\LowStockNotification;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Mail;

class LowStockAlert implements ShouldQueue
{
    /**
     * Create the event listener.
     * @param LowStock $event
     * @return void
     */

    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(LowStock $event)
    {
        $threshold = 10;

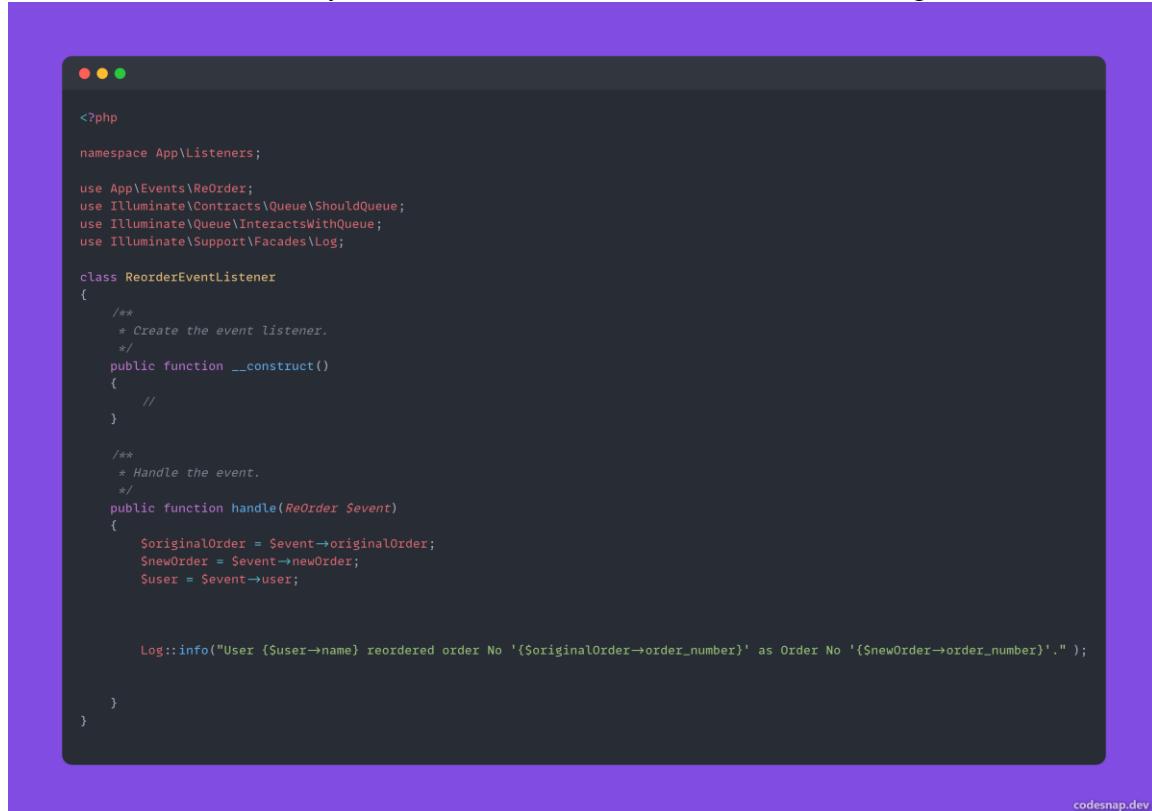
        if ($event->product->stocks < $threshold) {
            Log::info('Low stock alert for product: ' . $event->product->name);
        }
    }
}
```

codesnap.dev

Figure 37 low stock listener

Re-Order Event listener

This listener listens to any re-orders and sends a notification confirming the order.



A screenshot of a code editor window showing a PHP file named `ReorderEvent.php`. The code defines a class `ReorderEventListener` that implements the `ShouldQueue` contract. It has a constructor and a `handle` method that logs a message to the console. The code is syntax-highlighted with colors for different language elements.

```
<?php

namespace App\Listeners;

use App\Events\ReOrder;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;

class ReorderEventListener
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(ReOrder $event)
    {
        $originalOrder = $event->originalOrder;
        $newOrder = $event->newOrder;
        $user = $event->user;

        Log::info("User {$user->name} reordered order No '{$originalOrder->order_number}' as Order No '{$newOrder->order_number}'.");
    }
}
```

codesnap.dev

Figure 38 Re-order listener

Registration Listener

This listener listens to registrations happening through the website.



A screenshot of a code editor window titled 'codesnap.dev'. The code is written in PHP and defines a class named 'RegistrationLogger' within the 'App\Listeners' namespace. The class implements the 'ShouldQueue' and 'InteractsWithQueue' contracts from the Illuminate\Queue library. It has a constructor and a single method, 'handle', which takes an 'OnlineRegister' event as a parameter. Inside the handle method, it uses the 'OnlineRegistrations' model to create a new registration entry with 'user_id' and 'register_date' fields. Finally, it logs an info message using the 'Log' facade with the registered user's email.

```
<?php

namespace App\Listeners;

use App\Events\OnlineRegister;
use App\Models\OnlineRegistrations;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;

class RegistrationLogger
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(OnlineRegister $event): void
    {
        OnlineRegistrations::create(
            [
                'user_id' => $event->user->id,
                'register_date' => now(),
            ]
        );

        Log::info('User registered: ' . $event->user->email);
    }
}
```

codesnap.dev

Figure 39 Registration Listener

Online Registration Listener

This listener listens to the manual registrations done via the admin using the customer CRUD.



The screenshot shows a code editor window with a dark theme. The code is written in PHP and defines a class named `SiteRegisterListener`. The class implements the `ShouldQueue` and `InteractsWithQueue` contracts from the Illuminate\Queue namespace. It uses the `Log` facade for logging information. The `handle` method takes a `SiteRegister` event as input and creates a new `SiteRegistration` model with the user's ID and registration date. A log message is then sent indicating the user has registered.

```
<?php

namespace App\Listeners;

use App\Events\SiteRegister;
use App\Models\SiteRegistration;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;

class SiteRegisterListener
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {

    }

    /**
     * Handle the event.
     */
    public function handle(SiteRegister $event): void
    {
        $user = $event->user;

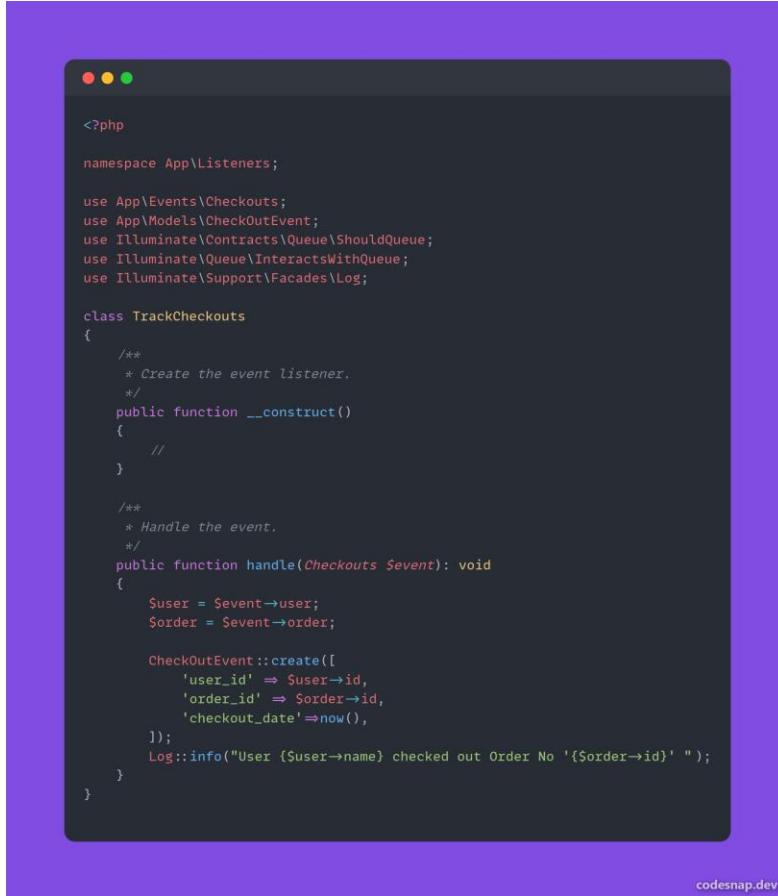
        SiteRegistration::create([
            'user_id' => $user->id,
            'register_date' => now(),
        ]);
        Log::info("User {$user->name} registered.");
    }
}
```

codesnap.dev

Figure 40 Site registration listener

Checkout Listener

This listener listens to orders being checked out and sends notifications confirming the order.



The screenshot shows a code editor window with a dark theme. The code is written in PHP and defines a class named `TrackCheckouts` which implements the `ShouldQueue` and `InteractsWithQueue` contracts from the Illuminate\Queue namespace. The class has a constructor and a `handle` method. The `handle` method takes a `Checkouts $event` parameter and performs the following actions:

- It retrieves the user and order from the event.
- It creates a new `CheckOutEvent` instance with the user ID, order ID, and current date.
- It logs an info message to the console indicating the user checked out the order.

```
<?php

namespace App\Listeners;

use App\Events\Checkouts;
use App\Models\CheckOutEvent;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;

class TrackCheckouts
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(Checkouts $event): void
    {
        $user = $event->user;
        $order = $event->order;

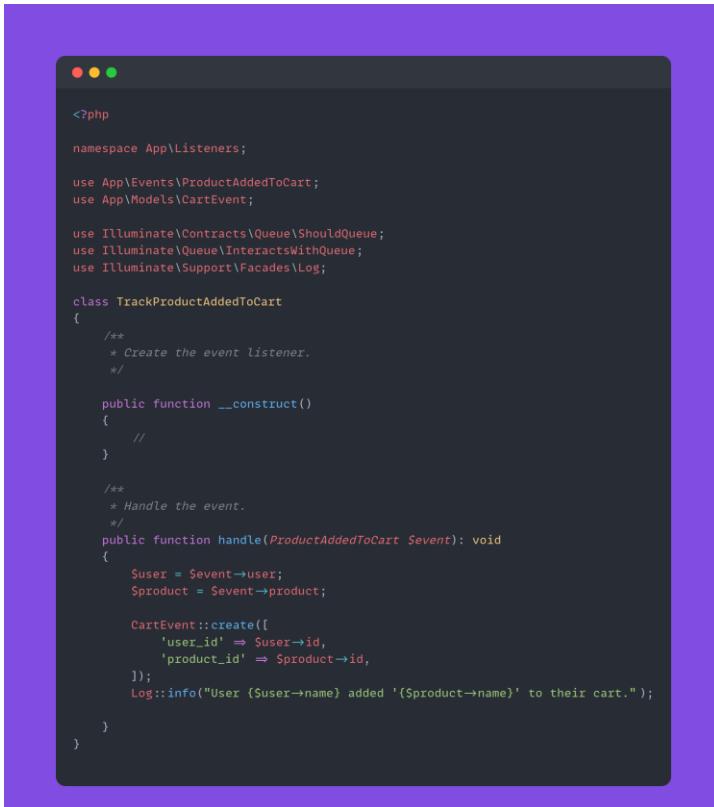
        CheckOutEvent::create([
            'user_id' => $user->id,
            'order_id' => $order->id,
            'checkout_date' => now(),
        ]);
        Log::info("User {$user->name} checked out Order No '{$order->id}'");
    }
}
```

codesnap.dev

Figure 41 Checkout Listener

Product Added to cart listener

This listener listens to products being added to cart which helps us with analytics such as abandoned cart rate.



```
<?php

namespace App\Listeners;

use App\Events\ProductAddedToCart;
use App\Models\CartEvent;

use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Support\Facades\Log;

class TrackProductAddedToCart
{
    /**
     * Create the event listener.
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(ProductAddedToCart $event): void
    {
        $user = $event->user;
        $product = $event->product;

        CartEvent::create([
            'user_id' => $user->id,
            'product_id' => $product->id,
        ]);
        Log::info("User {$user->name} added '{$product->name}' to their cart.");
    }
}
```

codesnap.dev

Figure 42 product added to cart listener

Events

Checkout Event



The screenshot shows a code editor window with a dark theme. The code is written in PHP and defines a class named `Checkouts` within the `App\Events` namespace. The class implements several traits: `Dispatchable`, `InteractsWithSockets`, and `SerializesModels`. It has two properties: `$user` and `$order`. The `__construct` method initializes these properties. The `broadcastOn` method returns an array containing a `PrivateChannel` instance with the name 'channel-name'. The code uses various Laravel service provider classes from the `Illuminate` namespace.

```
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class Checkouts
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

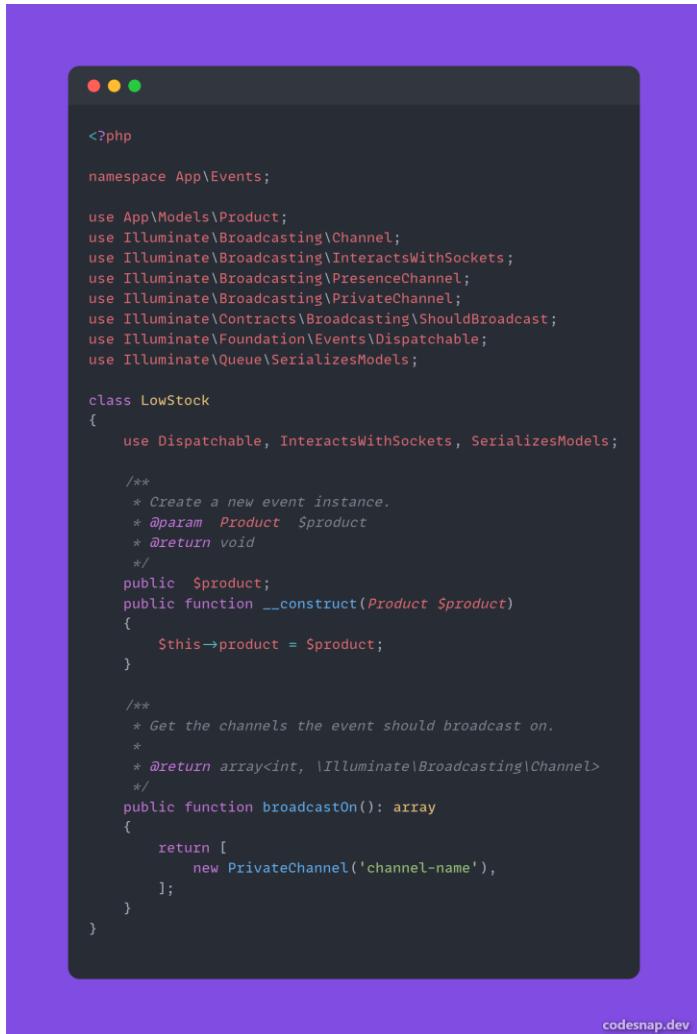
    /**
     * Create a new event instance.
     */
    public $user;
    public $order;
    public function __construct($user, $order)
    {
        $this->user = $user;
        $this->order = $order;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

codesnap.dev

Figure 43 Checkout event

Low stock event



```
<?php

namespace App\Events;

use App\Models\Product;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class LowStock
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

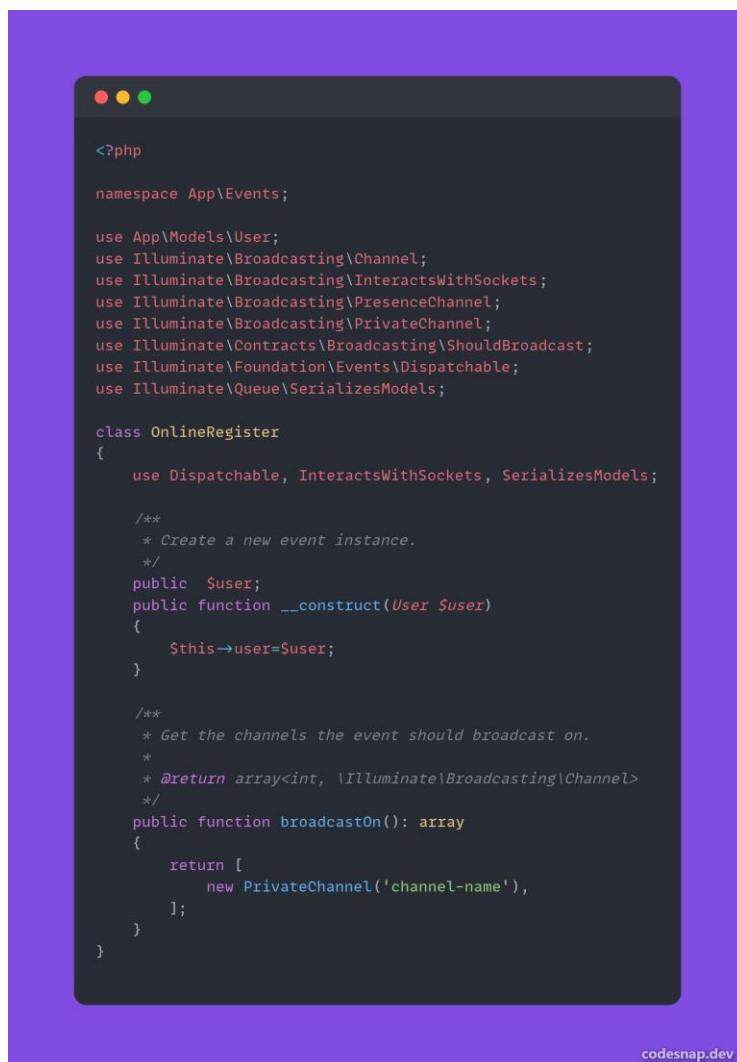
    /**
     * Create a new event instance.
     * @param Product $product
     * @return void
     */
    public $product;
    public function __construct(Product $product)
    {
        $this->product = $product;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

codesnap.dev

Figure 44 Low stock event

Online register event



A screenshot of a code editor window showing a PHP file. The file contains a class definition for 'OnlineRegister' which extends 'Dispatchable', 'InteractsWithSockets', and 'SerializesModels'. It includes a constructor that takes a 'User' object and assigns it to a private property. The 'broadcastOn()' method returns an array containing a 'PrivateChannel' object with the name 'channel-name'. The code uses Laravel's broadcasting traits and contracts.

```
<?php

namespace App\Events;

use App\Models\User;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class OnlineRegister
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    public $user;
    public function __construct(User $user)
    {
        $this->user=$user;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

codesnap.dev

Figure 45 Online register event

Site Register Event



The screenshot shows a Mac OS X terminal window with a dark theme. The window title bar has three colored dots (red, yellow, green) at the top left. The main area of the window displays a block of PHP code. At the bottom right of the window, there is a small watermark-like text "codesnap.dev".

```
<?php

namespace App\Events;

use App\Models\User;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

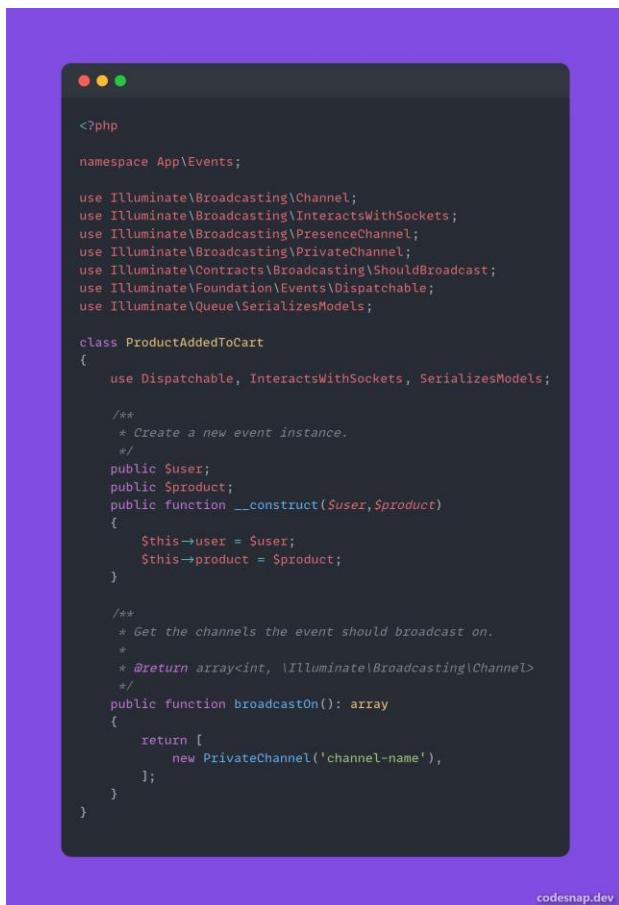
class SiteRegister
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    public $user;
    public function __construct(User $user)
    {
        $this->user = $user;
    }

    /**
     * Get the channels the event should broadcast on.
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

Figure 46 Site register event

Product added to cart event



The screenshot shows a code editor window with a dark theme. The code is written in PHP and defines a class named `ProductAddedToCart`. The class extends `Dispatchable`, `InteractsWithSockets`, and `SerializesModels`. It has properties `$user` and `$product`, and a constructor `__construct($user, $product)` that initializes these properties. The `broadcastOn()` method returns an array containing a `PrivateChannel` instance with the name 'channel-name'. The code uses annotations like `/**`, `* Create a new event instance.`, and `* @return array<int, \Illuminate\Broadcasting\Channel>`.

```
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class ProductAddedToCart
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    public $user;
    public $product;
    public function __construct($user, $product)
    {
        $this->user = $user;
        $this->product = $product;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

codesnap.dev

Figure 47 product added to cart event

Re-order event



```
<?php

namespace App\Events;

use App\Models\Order;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class ReOrder
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    public $originalOrder;
    public $newOrder;
    public $user;

    public function __construct(Order $originalOrder, Order $newOrder, $user)
    {
        $this->originalOrder = $originalOrder;
        $this->newOrder = $newOrder;
        $this->user = $user;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

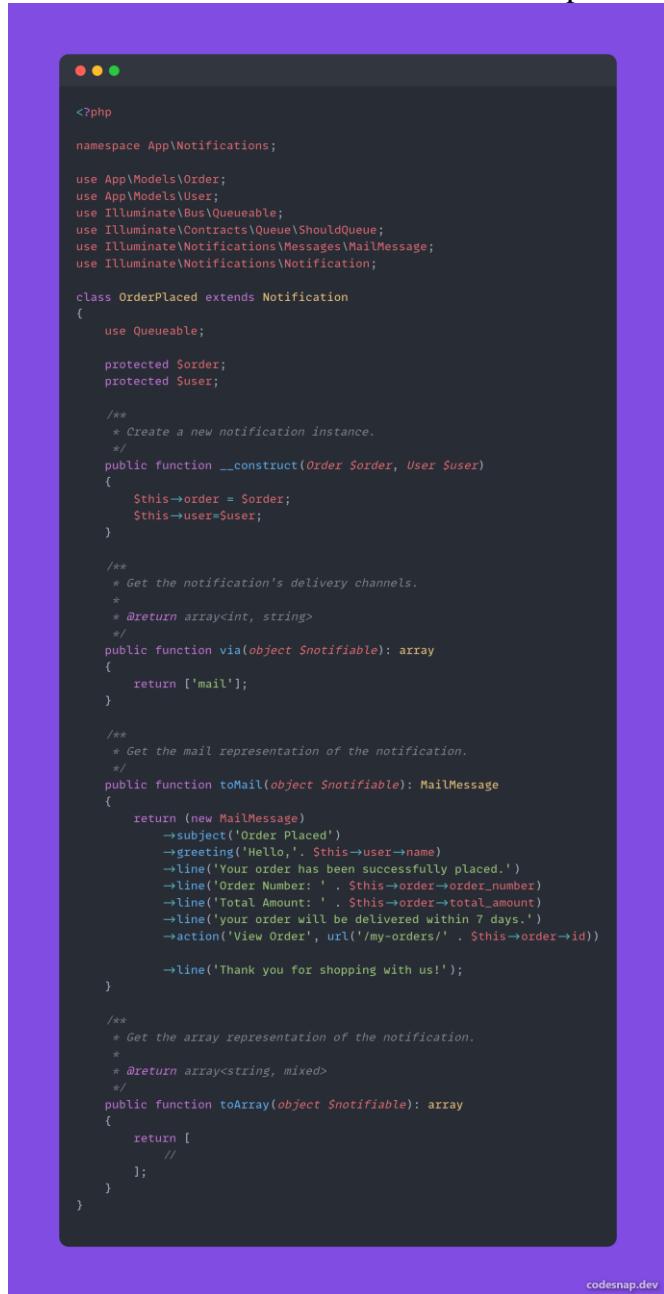
codesnap.dev

Figure 48 Re-order event

Notifications

Order placed Notification

Notification sent when a customer makes a purchase.



The screenshot shows a code editor window with a dark theme. The code is written in PHP and defines a class named `OrderPlaced` which extends the `Notification` class. The class uses the `Queueable` trait. It has protected properties `$order` and `$user`. The `__construct` method initializes these properties. The `via` method returns an array containing the string `'mail'`. The `toMail` method creates a `MailMessage` object with a subject of `'Order Placed'`, a greeting to the user, and a body containing information about the order being successfully placed, the order number, the total amount, and a delivery note. It also includes a link to view the order. The `toArray` method returns an empty array. The code is annotated with PHPDoc comments explaining the methods and their parameters.

```
<?php

namespace App\Notifications;

use App\Models\Order;
use App\Models\User;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;

class OrderPlaced extends Notification
{
    use Queueable;

    protected $order;
    protected $user;

    /**
     * Create a new notification instance.
     */
    public function __construct(Order $order, User $user)
    {
        $this->order = $order;
        $this->user = $user;
    }

    /**
     * Get the notification's delivery channels.
     *
     * @return array<int, string>
     */
    public function via(object $notifiable): array
    {
        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     */
    public function toMail(object $notifiable): MailMessage
    {
        return (new MailMessage)
            ->subject('Order Placed')
            ->greeting("Hello, " . $this->user->name)
            ->line("Your order has been successfully placed.")
            ->line("Order Number: " . $this->order->order_number)
            ->line("Total Amount: " . $this->order->total_amount)
            ->line("your order will be delivered within 7 days.")
            ->action('View Order', url('/my-orders/' . $this->order->id))

            ->line('Thank you for shopping with us!');
    }

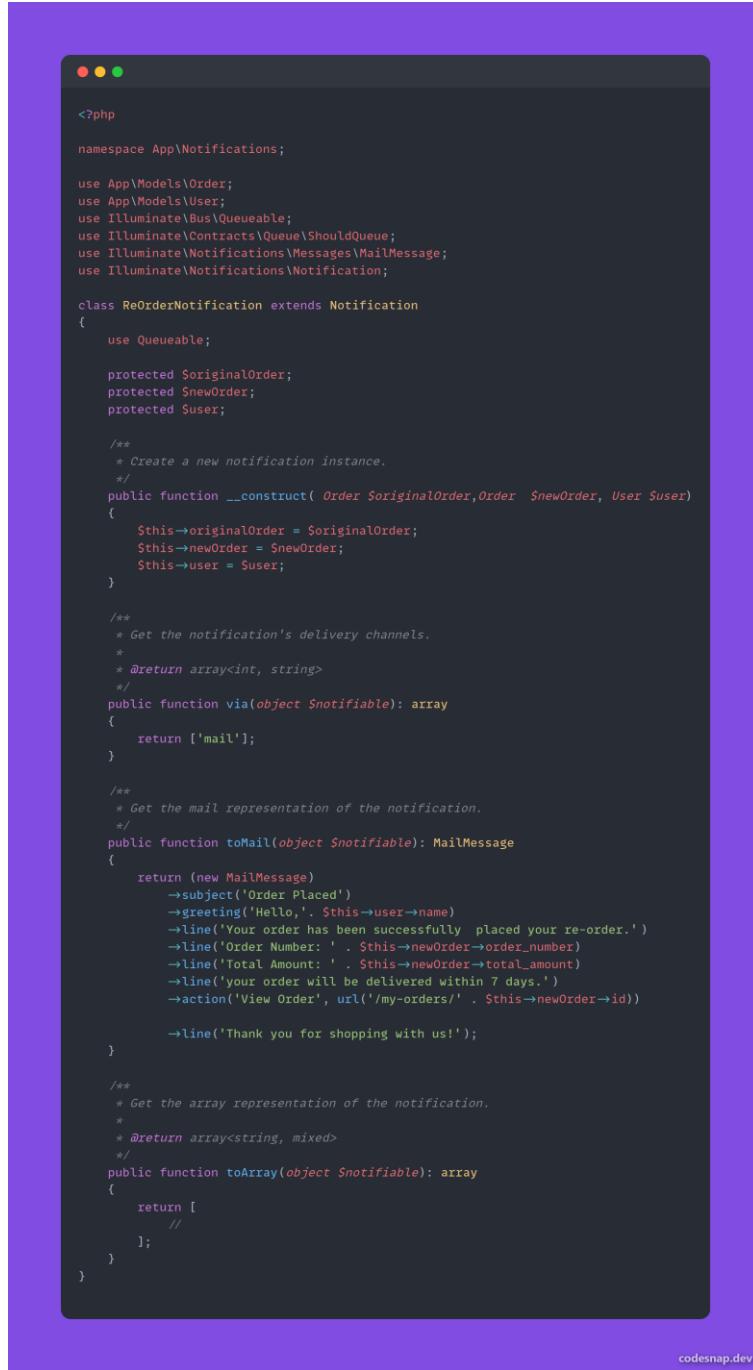
    /**
     * Get the array representation of the notification.
     *
     * @return array<string, mixed>
     */
    public function toArray(object $notifiable): array
    {
        return [
            //
        ];
    }
}
```

codesnap.dev

Figure 49 Order placed notification

Re-order Notification

The notification sent when a customer makes an re-order.



A screenshot of a code editor displaying a PHP class named `ReOrderNotification`. The code is part of the `App\Notifications` namespace and extends the `Notification` class. It includes methods for constructing the notification, determining delivery channels (via email), creating a mail message, and returning an array representation of the notification. The mail message contains a greeting, a subject ('Order Placed'), and a body with details about the re-order, including the user's name, order number, total amount, and a link to view the order.

```
<?php

namespace App\Notifications;

use App\Models\Order;
use App\Models\User;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;

class ReOrderNotification extends Notification
{
    use Queueable;

    protected $originalOrder;
    protected $newOrder;
    protected $user;

    /**
     * Create a new notification instance.
     */
    public function __construct( Order $originalOrder, Order $newOrder, User $user )
    {
        $this->originalOrder = $originalOrder;
        $this->newOrder = $newOrder;
        $this->user = $user;
    }

    /**
     * Get the notification's delivery channels.
     *
     * @return array<int, string>
     */
    public function via(object $notifiable): array
    {
        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     */
    public function toMail(object $notifiable): MailMessage
    {
        return (new MailMessage)
            ->subject('Order Placed')
            ->greeting('Hello, ' . $this->user->name)
            ->line('Your order has been successfully placed your re-order.')
            ->line("Order Number: " . $this->newOrder->order_number)
            ->line("Total Amount: " . $this->newOrder->total_amount)
            ->line('your order will be delivered within 7 days.')
            ->action('View Order', url('/my-orders/' . $this->newOrder->id))

            ->line('Thank you for shopping with us!');
    }

    /**
     * Get the array representation of the notification.
     *
     * @return array<string, mixed>
     */
    public function toArray(object $notifiable): array
    {
        return [
            //
        ];
    }
}
```

codesnap.dev

Figure 50 Re-order notification

Low stock Notification

The notification sent when the stock of a product falls below 10.



A screenshot of a code editor displaying a PHP file. The file contains a class definition for 'LowStockNotification' which extends 'Notification'. It includes methods for creating a new instance, determining delivery channels (via email), generating a mail message, and getting an array representation of the notification. The code uses Laravel's notification system, specifically the MailMessage and Queueable traits.

```
<?php

namespace App\Notifications;

use App\Models\Product;
use Illuminate\Bus\Queueable;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;

class LowStockNotification extends Notification
{
    use Queueable;

    protected $product;
    protected $recipientEmail;

    /**
     * Create a new notification instance.
     */
    public function __construct(Product $product)
    {
        $this->product = $product;
    }

    /**
     * Get the notification's delivery channels.
     *
     * @return array<int, string>
     */
    public function via(object $notifiable): array
    {
        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     */
    public function toMail(object $notifiable): MailMessage
    {
        return (new MailMessage)
            ->subject('Low stock alert for product: ' . $this->product->name)
            ->greeting('Hello,')
            ->line('Product ID: ' . $this->product->id)
            ->line('This is a low stock alert for the product.');

        // You don't need to use $this->recipientEmail here
    }

    /**
     * Get the array representation of the notification.
     *
     * @return array<string, mixed>
     */
    public function toArray(object $notifiable): array
    {
        return [
            //
        ];
    }
}
```

codesnap.dev

Figure 51 Low stock notification

Add queue jobs

Queue Jobs is used to get the view count for each product when a user visits the product description page



The screenshot shows a code editor window with a dark theme. At the top, there are three small circular icons: red, yellow, and green. The code itself is a PHP file named `ProductViews.php`. It defines a class `ProductViews` that implements the `ShouldQueue` interface. The class has a protected property `$product` and two methods: `__construct` and `handle`. The `__construct` method initializes the `$product` property. The `handle` method increments the `'views'` attribute of the `$product` object.

```
<?php

namespace App\Jobs;

use App\Models\Product;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldBeUnique;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class ProductViews implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
    protected $product;

    /**
     * Create a new job instance.
     */
    public function __construct(Product $product)
    {
        $this->product = $product;
    }

    /**
     * Execute the job.
     */
    public function handle()
    {
        $this->product->increment('views');
    }
}
```

codesnap.dev

Figure 52 Product hits

Cache Routes

Routes are cached to increase performance and retrieve routes faster.

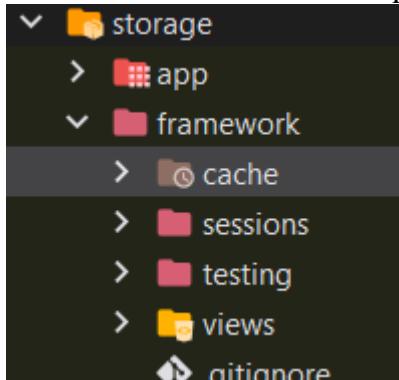


Figure 53 Cache Routes

Alpine Js

Alpine js was used throughout the project for pop-ups, modals and dropdowns as shown below.

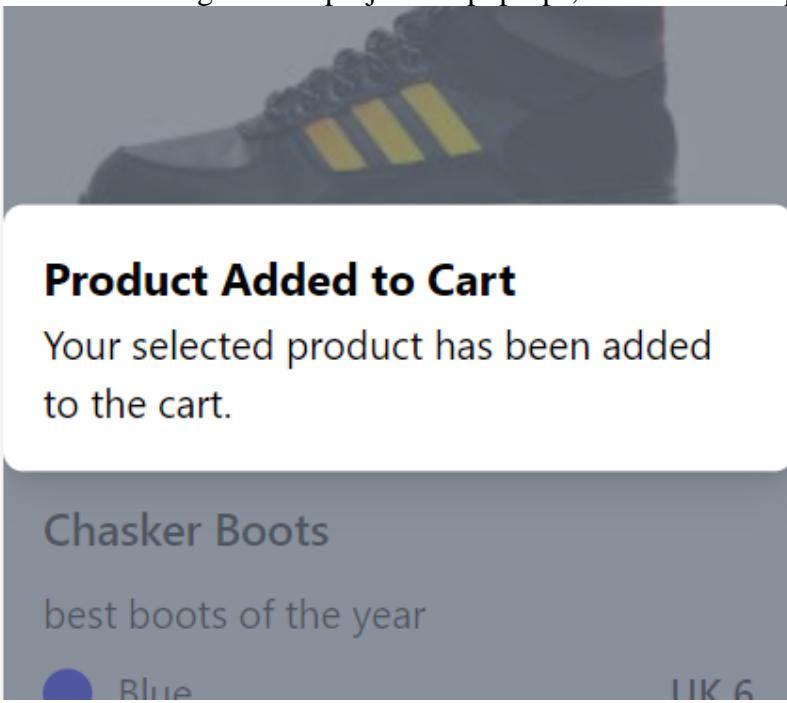


Figure 54 pop up

| Order Number | Order Details | Customer | Order Status |
|-------------------|---------------|--|--------------|
| ORD-6519020A372AF | Details | Dalton | pending |
| | | <p>Product Name: UrbanLoaf* 1</p> <p>UNIT PRICE : 6600 LKR</p> | |

Buy Again

| | | | |
|-------------------|---------|-------------|---------|
| ORD-6519027929B62 | Details | Dalton Wolf | pending |
| | | | |

Buy Again

Figure 55 show

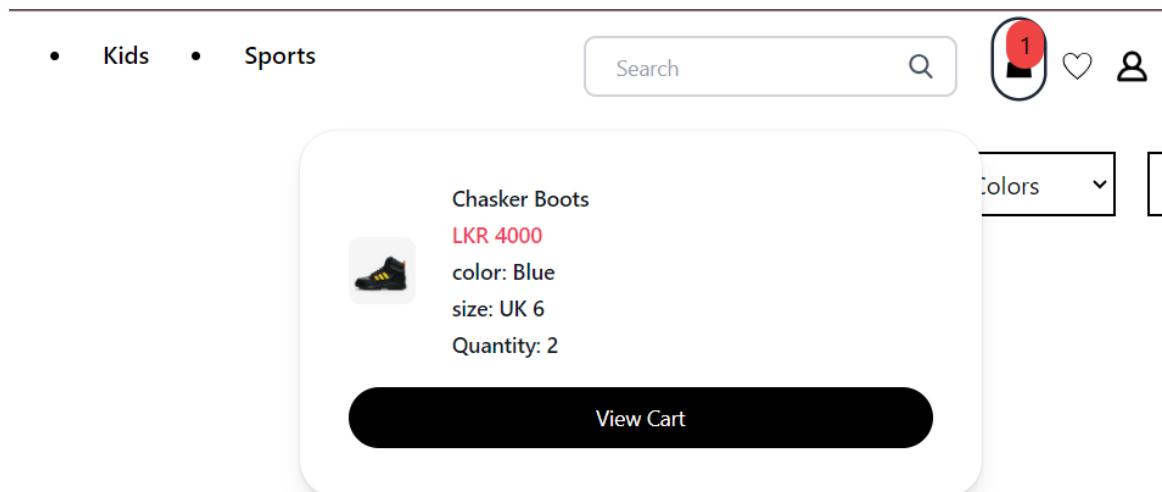


Figure 56 Dropdown

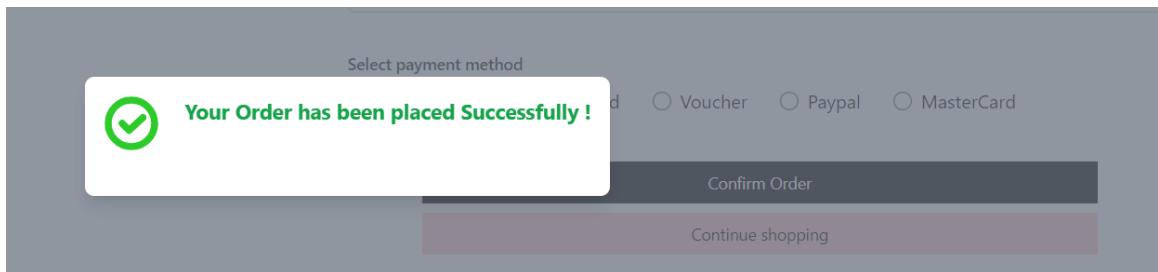


Figure 57 Pop up 2

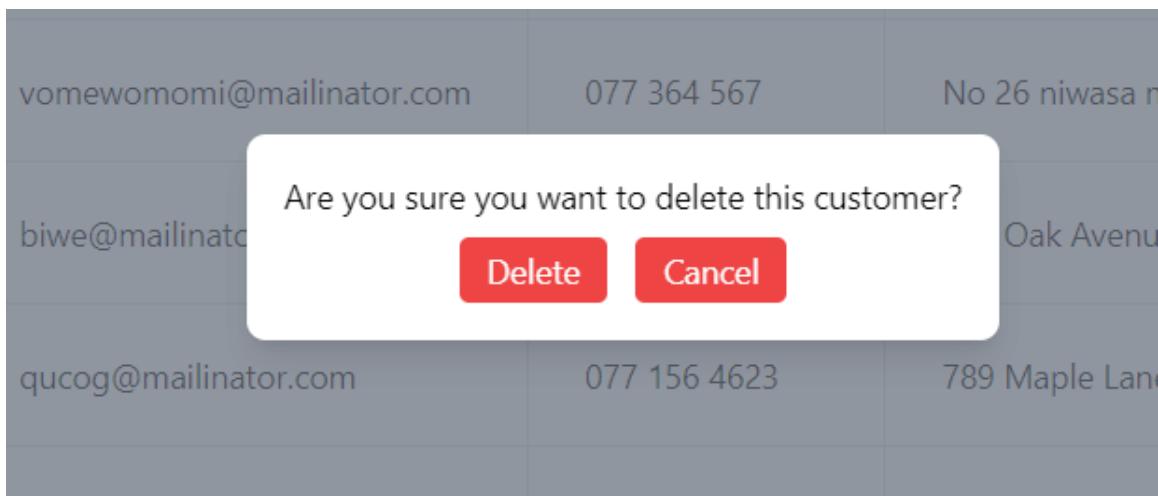


Figure 58 Modal

Chart Js



Figure 59 Chart Js

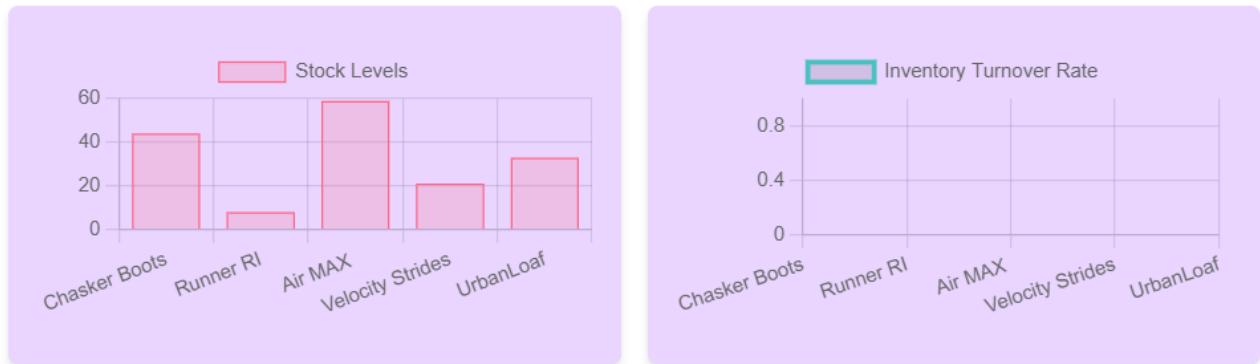


Figure 60 Chart js 2

Livewire

Livewire for product CRUD



```
#!/usr/bin/env php
--> composer require livewire/laravel
use App\Models\Category;
use App\Models\Color;
use App\Models\Feature;
use App\Models\Image;
use App\Models\Product;
use App\Models\Size;
use Illuminate\Foundation\Bootstrap;

class Product extends Component
{
    use WritableAttributes;
    use HasModelAttributes;
    use HasModelRelationships;
    use HasModelTranslations;
    use HasModelProduct;
    use HasModelImage;
    use HasModelSize;
    use HasModelColor;

    static $rules = [
        'name' => 'required|max:255',
        'category_id' => 'required|exists:categories,id',
        'image' => 'image|mimes:jpeg,png,jpg,gif,svg|max:2048',
        'size_id' => 'required|exists:sizes,id',
        'color_id' => 'required|exists:colors,id',
        'feature_ids' => 'array|required',
        'details' => 'array|required',
        'stock' => 'integer|required',
        'price' => 'float|required',
        'status' => 'boolean|required'
    ];
}

public function mount()
{
    $this->category = Category::all();
    $this->size = Size::all();
    $this->color = Color::all();
    $this->feature = Feature::all();
}

public function mountSize()
{
    $size = Size::all();
    $category = Category::all();
    $color = Color::all();
    $feature = Feature::all();

    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

public function mountCategory()
{
    $category = Category::all();
    $size = Size::all();
    $color = Color::all();
    $feature = Feature::all();

    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

public function mountColor()
{
    $color = Color::all();
    $category = Category::all();
    $size = Size::all();
    $feature = Feature::all();

    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

public function mountFeature()
{
    $feature = Feature::all();
    $category = Category::all();
    $size = Size::all();
    $color = Color::all();

    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

private function saveProduct(Product $product)
{
    $product->category_id = $this->category_id;
    $product->size_id = $this->size_id;
    $product->color_id = $this->color_id;
    $product->feature_ids = $this->feature_ids;
    $product->details = $this->details;
    $product->stock = $this->stock;
    $product->price = $this->price;
    $product->status = $this->status;

    $product->save();

    session(['success' => $product->id . 'Product updated successfully.']);
    session(['message' => 'Product updated successfully.']);
    $this->resetInputFields();
}

public function create()
{
    $category = Category::all();
    $size = Size::all();
    $color = Color::all();
    $feature = Feature::all();

    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

public function update()
{
    $category = Category::all();
    $size = Size::all();
    $color = Color::all();
    $feature = Feature::all();

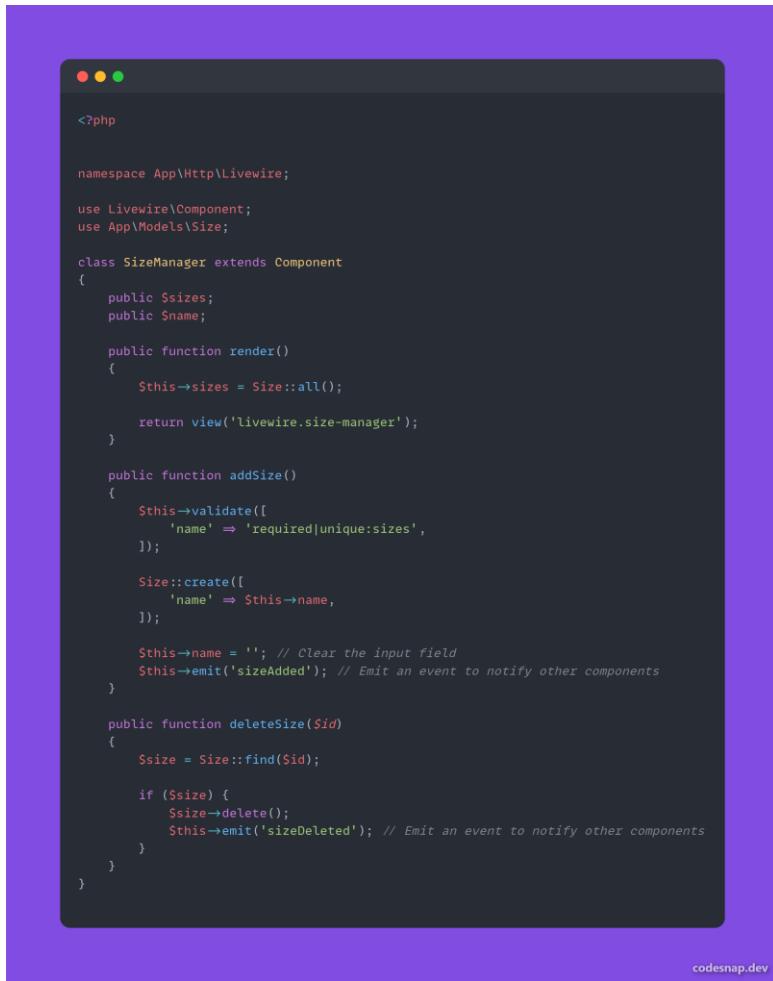
    return view('livewire.products', ['products' => Product::where('category_id', 'category_id')
        ->where('size_id', 'size_id')
        ->where('color_id', 'color_id')
        ->where('feature_ids', 'in', 'feature_ids')
        ->where('stock', '!=', 0)
        ->where('status', '!=', false)
        ->get()]);
}

public function delete($id)
{
    $product = Product::find($id);
    $product->delete();

    session(['success' => $product->id . 'Product deleted successfully.']);
    session(['message' => 'Product deleted successfully.']);
    $this->resetInputFields();
}
```

Figure 61 Livewire product crud

Livewire size CRUD



```
<?php

namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Size;

class SizeManager extends Component
{
    public $sizes;
    public $name;

    public function render()
    {
        $this->sizes = Size::all();

        return view('livewire.size-manager');
    }

    public function addSize()
    {
        $this->validate([
            'name' => 'required|unique:sizes',
        ]);

        Size::create([
            'name' => $this->name,
        ]);

        $this->name = ''; // Clear the input field
        $this->emit('sizeAdded'); // Emit an event to notify other components
    }

    public function deleteSize($id)
    {
        $size = Size::find($id);

        if ($size) {
            $size->delete();
            $this->emit('sizeDeleted'); // Emit an event to notify other components
        }
    }
}
```

codesnap.dev

Figure 62 livewire Size crud

Livewire Color CRUD



A screenshot of a code editor displaying a PHP file named `ColorsManager.php`. The code implements a Livewire component for managing colors. It includes methods for mounting the component, adding new colors, and deleting existing ones. The code uses Laravel's Eloquent ORM for interacting with a `Color` model.

```
<?php

namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Color;

class ColorsManager extends Component
{
    public $colors;
    public $newColor;

    public function mount()
    {
        $this->colors = Color::all();
    }

    public function addColor()
    {
        $this->validate([
            'newColor' => 'required|unique:colors,color',
        ]);

        Color::create(['color' => $this->newColor]);

        $this->colors = Color::all();
        session()->flash('success', 'Color added successfully.');
    }

    public function deleteColor($id)
    {
        $color = Color::find($id);

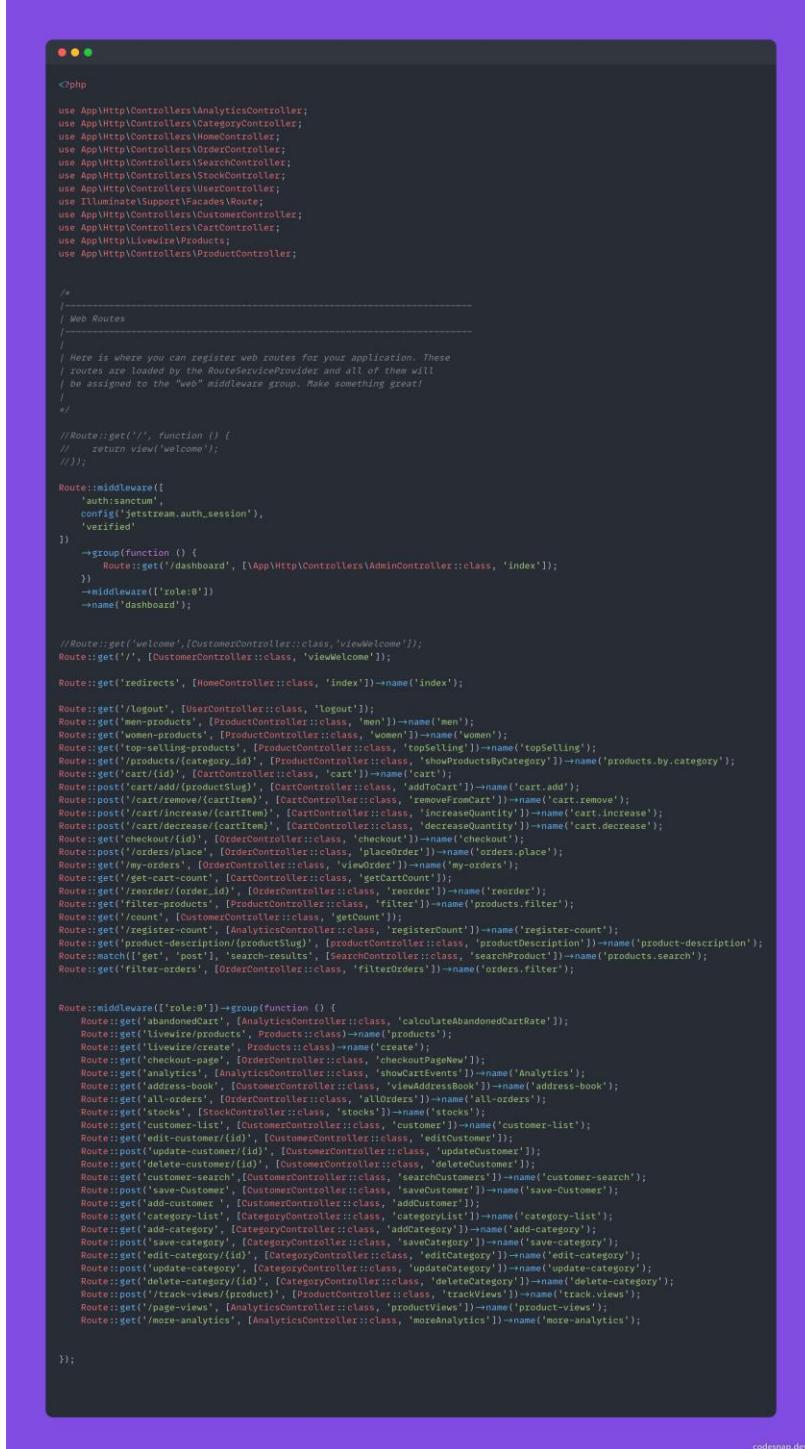
        if ($color) {
            $color->delete();
            $this->emit('colorDeleted'); // Emit an event to notify other components
        }
    }

    public function render()
    {
        return view('livewire.colors-manager');
    }
}
```

codesnap.dev

Figure 63 livewire color crud

Routes



The screenshot shows a code editor displaying a PHP file named 'routes/web.php'. The file contains the configuration for web routes in a Laravel application. It includes imports for various controllers and uses the Route facade to define routes for different actions like get, post, and middleware. The code is well-organized with comments explaining the purpose of certain sections.

```
<?php

use App\Http\Controllers\AnalyticsController;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\SearchController;
use App\Http\Controllers\StockController;
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\CustomerController;
use App\Http\Controllers\CartController;
use App\LiveWire\Products;
use App\Http\Controllers\ProductController;

// Web Routes
// -----
// Here is where you can register web routes for your application. These
// routes are loaded by the RouteServiceProvider and all of them will
// be assigned to the "web" middleware group. Make something great!
//

//Route::get('/', function () {
//    return view('welcome');
//});

Route::middleware([
    'auth:sanctum',
    config('jetstream.auth_session'),
    'verified'
])
    ->group(function () {
        Route::get('dashboard', [App\Http\Controllers\AdminController::class, 'index']);
    })
    ->middleware(['role:0'])
    ->name("dashboard");

//Route::get('welcome',[CustomerController::class,'viewWelcome']);
Route::get('/', [CustomerController::class, 'viewWelcome']);

Route::get('redirects', [HomeController::class, 'index'])->name('index');

Route::get('logout', [UserController::class, 'logout']);
Route::get('men-products', [ProductController::class, 'men'])->name('men');
Route::get('women-products', [ProductController::class, 'women'])->name('women');
Route::get('top-selling-products', [ProductController::class, 'topSelling'])->name('topSelling');
Route::get('/products/{category_id}', [ProductController::class, 'showProductsByCategory'])->name('products.by.category');
Route::get('cart/{id}', [CartController::class, 'cart'])->name('cart');
Route::post('cart/add/{productSlug}', [CartController::class, 'addToCart'])->name('cart.add');
Route::post('cart/remove/{productSlug}', [CartController::class, 'removeFromCart'])->name('cart.remove');
Route::post('cart/increase/{cartItem}', [CartController::class, 'increaseQuantity'])->name('cart.increase');
Route::post('cart/decrease/{cartItem}', [CartController::class, 'decreaseQuantity'])->name('cart.decrease');
Route::get('checkout/{id}', [OrderController::class, 'checkout']);
Route::post('orders/place', [OrderController::class, 'placeOrder'])->name('orders.place');
Route::get('my-orders', [OrderController::class, 'viewOrder'])->name('my-orders');
Route::get('get-cart-count', [CartController::class, 'getCartCount']);
Route::get('/reorder/{order_id}', [OrderController::class, 'reorder'])->name('reorder');
Route::get('filter-products', [ProductController::class, 'filter'])->name('products.filter');
Route::get('/count', [CustomerController::class, 'getCount']);
Route::get('/register-count', [AnalyticsController::class, 'registerCount'])->name('register-count');
Route::get('product-description/{productSlug}', [ProductController::class, 'productDescription'])->name('product-description');
Route::match(['get', 'post'], 'search-results', [SearchController::class, 'searchProduct'])->name('products.search');
Route::get('filter-orders', [OrderController::class, 'filterOrders'])->name('orders.filter');

Route::middleware(['role:0'])->group(function () {
    Route::get('abandonedCart', [AnalyticsController::class, 'calculateAbandonedCartRate']);
    Route::get('livewire/products', Products::class)->name('products');
    Route::get('livewire/create', Products::class)->name('create');
    Route::get('checkout-page', OrderController::class, 'checkoutPageNew');
    Route::get('analytics', AnalyticsController::class, 'showCartEvents')->name('Analytics');
    Route::get('address-book', CustomerController::class, 'viewAddressBook')->name('address-book');
    Route::get('all-orders', OrderController::class, 'allOrders')->name('all-orders');
    Route::get('stocks', StockController::class, 'stocks')->name('stocks');
    Route::get('customer-list', CustomerController::class, 'customer')->name('customer-list');
    Route::get('edit-customer/{id}', CustomerController::class, 'editCustomer');
    Route::post('update-customer/{id}', CustomerController::class, 'updateCustomer');
    Route::get('customer-delete/{id}', CustomerController::class, 'deleteCustomer');
    Route::get('customer-search', CustomerController::class, 'searchCustomer')->name('customer-search');
    Route::post('save-Customer', CustomerController::class, 'saveCustomer')->name('save-Customer');
    Route::get('add-customer', CustomerController::class, 'addCustomer');
    Route::get('category-list', CategoryController::class, 'categoryList')->name('category-list');
    Route::get('add-category', CategoryController::class, 'addCategory')->name('add-category');
    Route::post('save-category', CategoryController::class, 'saveCategory')->name('save-category');
    Route::get('edit-category/{id}', CategoryController::class, 'editCategory')->name('edit-category');
    Route::post('update-category', CategoryController::class, 'updateCategory')->name('update-category');
    Route::get('delete-category/{id}', CategoryController::class, 'deleteCategory')->name('delete-category');
    Route::post('/track-views/{product}', [ProductController::class, 'trackViews'])->name('track-views');
    Route::get('/page-views', [AnalyticsController::class, 'productViews'])->name('product-views');
    Route::get('/more-analytics', [AnalyticsController::class, 'moreAnalytics'])->name('more-analytics');
});

});
```

codesnap.dev

Figure 64 Routes

Code structure

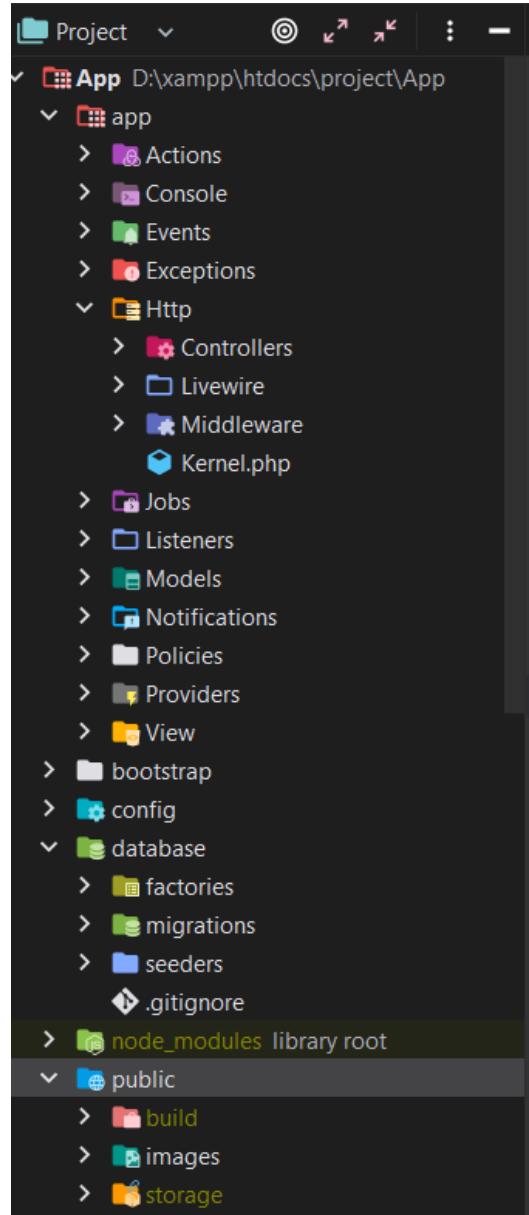


Figure 65 Code structure part 1

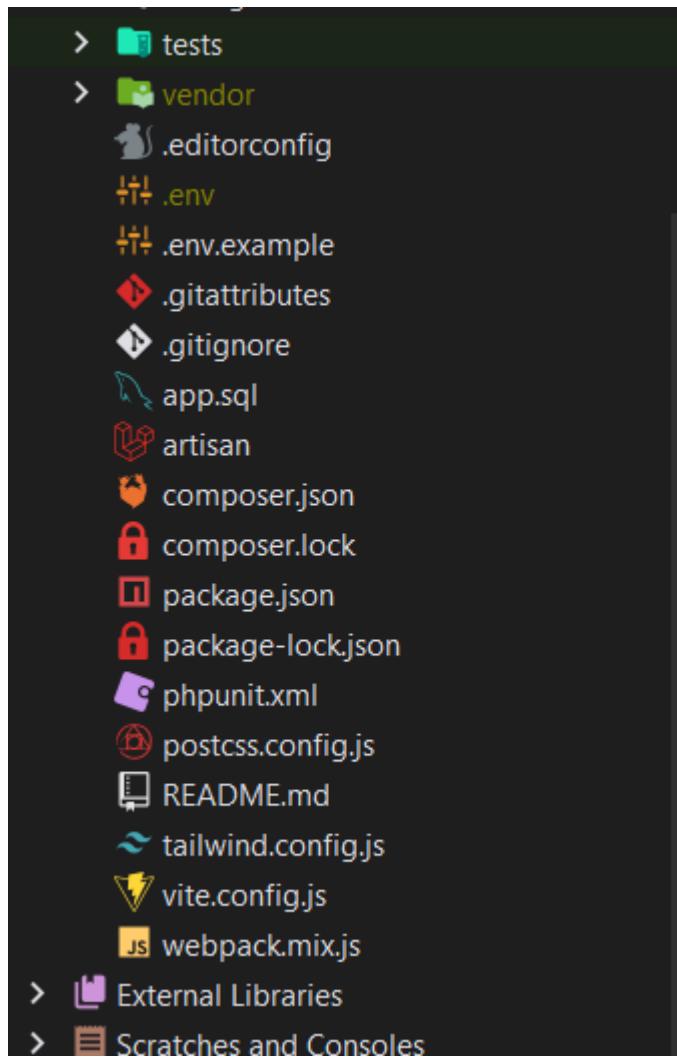


Figure 66 Code structure part 2

Migrations

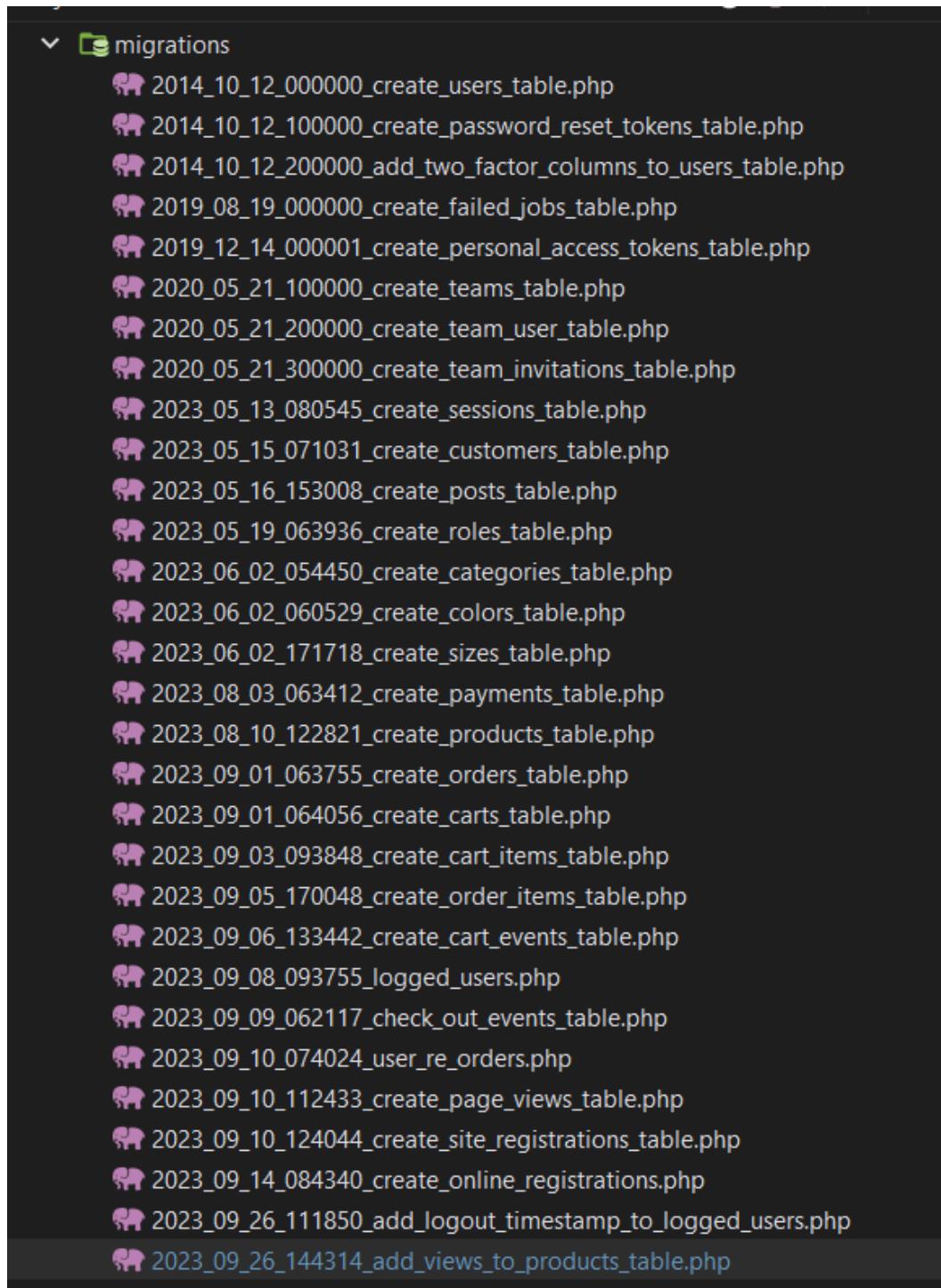


Figure 67 Migrations

MySQL database snapshot

The screenshot shows the MySQL Workbench interface with a database named 'test'. The main area displays a table of 28 tables, each with columns for Table name, Action (Browse, Structure, Search, Insert, Empty, Drop), Rows, Type (InnoDB), Collation (utf8mb4_unicode_ci), Size, and Overhead. The 'orders' table has the most rows (1,520). The 'migrations' table has the largest size (192.0 KIB). The 'users' table has the largest overhead (64.0 KIB).

| Table | Action | Rows | Type | Collation | Size | Overhead |
|------------------------|---|--------------|---------------|---------------------------|----------------|------------|
| cart | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| cart_events | Browse Structure Search Insert Empty Drop | 17 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| cart_items | Browse Structure Search Insert Empty Drop | 4 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| categories | Browse Structure Search Insert Empty Drop | 4 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| check_out_events | Browse Structure Search Insert Empty Drop | 5 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| colors | Browse Structure Search Insert Empty Drop | 7 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| customers | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| failed_jobs | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| logged_users | Browse Structure Search Insert Empty Drop | 1,520 | InnoDB | utf8mb4_unicode_ci | 192.0 KIB | - |
| migrations | Browse Structure Search Insert Empty Drop | 30 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| online_registrations | Browse Structure Search Insert Empty Drop | 6 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| orders | Browse Structure Search Insert Empty Drop | 5 | InnoDB | utf8mb4_unicode_ci | 64.0 KIB | - |
| order_items | Browse Structure Search Insert Empty Drop | 6 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| page_views | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| password_reset_tokens | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| payments | Browse Structure Search Insert Empty Drop | 5 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| personal_access_tokens | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| posts | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| products | Browse Structure Search Insert Empty Drop | 6 | InnoDB | utf8mb4_unicode_ci | 80.0 KIB | - |
| re_orders | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 64.0 KIB | - |
| roles | Browse Structure Search Insert Empty Drop | 3 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| sessions | Browse Structure Search Insert Empty Drop | 1 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| sites_registrations | Browse Structure Search Insert Empty Drop | 2 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| sizes | Browse Structure Search Insert Empty Drop | 8 | InnoDB | utf8mb4_unicode_ci | 16.0 KIB | - |
| teams | Browse Structure Search Insert Empty Drop | 7 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| team_invitations | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| team_user | Browse Structure Search Insert Empty Drop | 0 | InnoDB | utf8mb4_unicode_ci | 32.0 KIB | - |
| users | Browse Structure Search Insert Empty Drop | 9 | InnoDB | utf8mb4_unicode_ci | 48.0 KIB | - |
| 28 tables | Sum | 1,645 | InnoDB | utf8mb4_general_ci | 1.2 MiB | 0 B |

Below the table, there are buttons for 'Check all' and 'With selected'. At the bottom, there are links for 'Print' and 'Data dictionary', and a 'Create new table' dialog with fields for 'Table name' (empty) and 'Number of columns' (4). A 'Create' button is also present. The bottom tabs include 'Console'.

Figure 68 Database

10.Test cases

| Test Case ID | Description | Pre-condition | Test Procedure | Test Inputs | Expected Output | Actual Output | Result |
|--------------|---|--|---|--|--|--|--------|
| TC-01 | User login functionality | User is registered | Enter valid credentials | Email and password | User logs in successfully | User logs in successfully | PASS |
| TC-02 | User register functionality | User is registered | Enter customer details | Customer name , email , contact and address and password | User registers logs in successfully | User registers logs in successfully | PASS |
| TC-03 | User can view the product description page for each product | Application is open | Click on a product to view the product description page | None | Product description page is displayed | Product description page is displayed | PASS |
| TC-04 | User can add an Item to cart | Application is open and user is registered | Click on the Add to cart button | None | The product is added to cart and a success message saying product added to cart is displayed | The product is added to cart and a success message saying product added to cart is displayed | PASS |
| TC-045 | User can remove an Item to cart | There are items in the cart | Click on the remove icon next to an item in the cart | None | The product is removed from the cart | The product is removed from the cart | PASS |
| TC-06 | User can increase or decrease the quantity of an item in the cart | There are items in the cart | Click on the increase or decrease icon | None | The quantity is increased or decreased as expected | The quantity is increased or decreased as expected | PASS |
| TC-07 | User can checkout an order successfully | Application is open and user is registered | Click on the checkout button | None | The order is made and a email notifying the user about the order is sent to the user | The order is made and a email notifying the user about the order is sent to the user | PASS |
| TC-08 | User can re-order an order | Application is open | Click on the re-order | None | The re-order is made and a | The re-order is made and a | PASS |

| | | | | | | | |
|-------|---|---|--|---|---|---|------|
| | successfully | and user is registered and there are pre-orders | button under a previous order | | email notifying the user about the re-order is sent to the user | a email notifying the user about the re-order is sent to the user | |
| TC-09 | User can view their purchase history | Application is open and user is registered and there are pre-orders | Click on the all orders button | None | The purchase history of the user is displayed to the user | The purchase history of the user is displayed to the user | PASS |
| TC-10 | Verify that the user can search for a product | Application is open | Search for a product in the search bar | Product name | The searched product is displayed if the product exists or a message saying "product not found " is displayed | The searched product is displayed if the product exists or a message saying "product not found " is displayed | PASS |
| TC-11 | Admin can register a user | Admin is logged in and the user doesn't exist | Enter customer details | Customer name, email ,address, and contact number | User is registered successfully and a success message saying "user registered successfully" is displayed | User is registered successfully and a success message saying "user registered successfully" is displayed | PASS |
| TC-12 | Admin can update a user | Admin is logged in and the user exist | Enter the new customer details | Customer name, email ,address, and contact number | User is updated successfully and a success message saying "user updated successfully" is displayed | User is updated successfully and a success message saying "user updated successfully" is displayed | PASS |
| TC-13 | Admin can remove a user | Admin is logged in and the user exists | Click on the remove button | none | User is removed successfully and a success message saying "user removed successfully" | User is removed successfully and a success message saying "user removed | PASS |

| | | | | | is displayed | successfully" is displayed | |
|-------|---|--|--|---|--|--|------|
| TC-14 | Admin can add a new product | Admin is logged in and the product doesn't exist | Enter product details | Product name, description, price, category, size , color and the stock amount | The product is added and a success message saying "product added successfully is displayed" | The product is added and a success message saying "product added successfully is displayed" | PASS |
| TC-15 | Admin can update a product | Admin is logged in and the product exists | Enter the new product details | Product name, description, price, category, size , color and the stock amount | The product is updated successfully and a success message saying "product updated successfully is displayed" | The product is updated successfully and a success message saying "product updated successfully is displayed" | PASS |
| TC-16 | Admin can remove a product | Admin is logged in and the product exists | Click on the remove button | None | The product is removed successfully and a success message saying "product removed successfully is displayed" | The product is removed successfully and a success message saying "product removed successfully is displayed" | PASS |
| TC-17 | Admin can view store analytics | Admin is logged | Navigate to the dashboard and the analytics page | None | The analytics are displayed | The analytics are displayed | PASS |
| TC-18 | Admin can view stock details | Admin is logged | Navigate to the stock management page | None | The stock details are displayed | The stock details are displayed | PASS |
| TC-19 | Admin is alerted if the stock level of a product falls below 10 | Admin is logged in and the product exists | Navigate to the stock management page | None | An alert saying "alert low stock levels" is displayed next to the relevant product | An alert saying "alert low stock levels" is displayed next to the relevant product | PASS |

| Test Case ID | Description | Pre-condition | Test Procedure | Test Inputs | Expected Output | Actual Output | Result |
|--------------|--|--|--|--|--|--|--------|
| TC-01 | User can easily login | User is registered | Enter valid credentials | Email and password | User logs in successfully | User logs in successfully | PASS |
| TC-02 | User can easily register | Application is open | Enter customer details | Customer name, email, address and password | Product description page is displayed | Product description page is displayed | PASS |
| TC-03 | User can intuitively search for a product | Application is open | Click on the Add to cart button | None | The product is added to cart and a success message saying product added to cart is displayed | The product is added to cart and a success message saying product added to cart is displayed | PASS |
| TC-04 | User can easily make a purchase without any difficulties | Application is open and user is registered | Make a purchase | None | The order is placed and a confirmation email is sent | The order is placed and a confirmation email is sent | PASS |
| TC-04 | User can easily understand and use the application | Application is open | Click on the remove icon next to an item in the cart | None | The product is removed from the cart | The product is removed from the cart | PASS |

11.1. Test case review process

The test cases provided cover a comprehensive range of functionalities in the POINT e-commerce application, ensuring that both user and admin interactions are thoroughly tested for correctness and reliability. Let's delve into the reasons behind the creation of these test cases and analyze their significance in ensuring the robustness of the system.

User Authentication and Registration (TC-01 to TC-02):

Reason for Creation: Essential for securing user data and ensuring authorized access.

Analysis: Verifies that users can seamlessly log in or register, crucial for a personalized shopping experience.

Product Interaction (TC-03 to TC-06):

Reason for Creation: Core functionalities of an e-commerce platform.

Analysis: Ensures users can interact with products by viewing details, adding to cart, adjusting quantities, and removing items.

Order Management (TC-07 to TC-09):

Reason for Creation: Critical for a smooth transaction and tracking of user orders.

Analysis: Tests the order placement, reorder, and viewing order history functionalities.

Search Functionality (TC-10):

Reason for Creation: Enhances user experience by allowing easy product discovery.

Analysis: Verifies that users can find products efficiently using the search bar.

Admin Operations (TC-11 to TC-19):

Reason for Creation: Admin functionalities are pivotal for managing products, users, and maintaining stock levels.

Analysis: Tests admin actions like user management, product CRUD operations, analytics viewing, and stock management.

Overall Application Usability (TC-01 to TC-04):

Reason for Creation: Ensures that the application is user-friendly and meets usability expectations.

Analysis: Verifies that users can intuitively navigate, perform actions, and have a seamless experience.

11.2. Deep Analysis of the test cases:

Completeness: The test suite covers a wide array of features, leaving minimal room for functionality gaps.

Reliability: Rigorous testing of critical functionalities ensures that users and admins can trust the system's performance.

Security: Authentication-related tests help maintain the confidentiality of user data.

Scalability: Admin-related test cases ensure the system can handle and manage growing product catalogs and user databases.

Usability: Test cases ensure that the application is not only functional but also user-friendly, meeting the expectations of a diverse user base.

In conclusion, these test cases collectively form a robust quality assurance process, assuring stakeholders that POINT's e-commerce application is not only feature-rich but also reliable, secure, and user-friendly.

12. Future Upgrade plan

1. Cloud Migration:

Explore the possibility of migrating the application to a cloud infrastructure for scalability and easier deployment.

2. Microservices Architecture:

Evaluate the adoption of a microservices architecture to enhance modularity and maintainability.

3. Mobile App Integration:

Develop a mobile application or optimize the existing application for mobile responsiveness to cater to a broader audience.

4. AI and Machine Learning Integration:

Investigate opportunities to integrate AI and machine learning for personalized recommendations and improved user experience.

5. Internationalization and Localization:

Implement features for internationalization and localization to make the application accessible to a global audience.

6. Automated Testing Framework:

Implement an automated testing framework to streamline regression testing and improve test coverage.

7. Enhanced Analytics:

Expand analytics capabilities to provide more comprehensive insights into user behavior and application performance.

8. Continuous Integration/Continuous Deployment (CI/CD):

Implement CI/CD pipelines to automate the deployment process and ensure faster and more reliable releases.

9. API Integration:

Provide APIs to allow seamless integration with third-party services and platforms.

10. User Feedback Mechanism:

Implement a robust user feedback mechanism to gather insights for continuous improvement.

13. References

The PHP framework for web artisans (no date) *Laravel*. Available at: <https://laravel.com/docs/10.x/readme> (Accessed: 30 September 2023).

Installation (no date) *Installation / Laravel Jetstream*. Available at: <https://jetstream.laravel.com/installation.html> (Accessed: 30 September 2023).

Install tailwind CSS with Laravel (no date) *Tailwind CSS*. Available at: <https://tailwindcss.com/docs/guides/laravel> (Accessed: 30 September 2023).

The QA Lead (2023) *10 best SAAS testing tools to verify & validate SAAS applications in 2023, The QA Lead*. Available at: <https://theqalead.com/tools/best-saas-testing-tools/> (Accessed: 30 September 2023).

Top 10 CRMS for software & SAAS companies: Streamlining Software-as-a-service with integrated customer relationship management tools (2023) *Smith.ai*. Available at: <https://smith.ai/blog/top-10-crms-for-software-saas-companies-streamlining-software-as-a-service-with-integrated-customer-relationship-management-tools> (Accessed: 30 September 2023).