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

Table of Figures

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

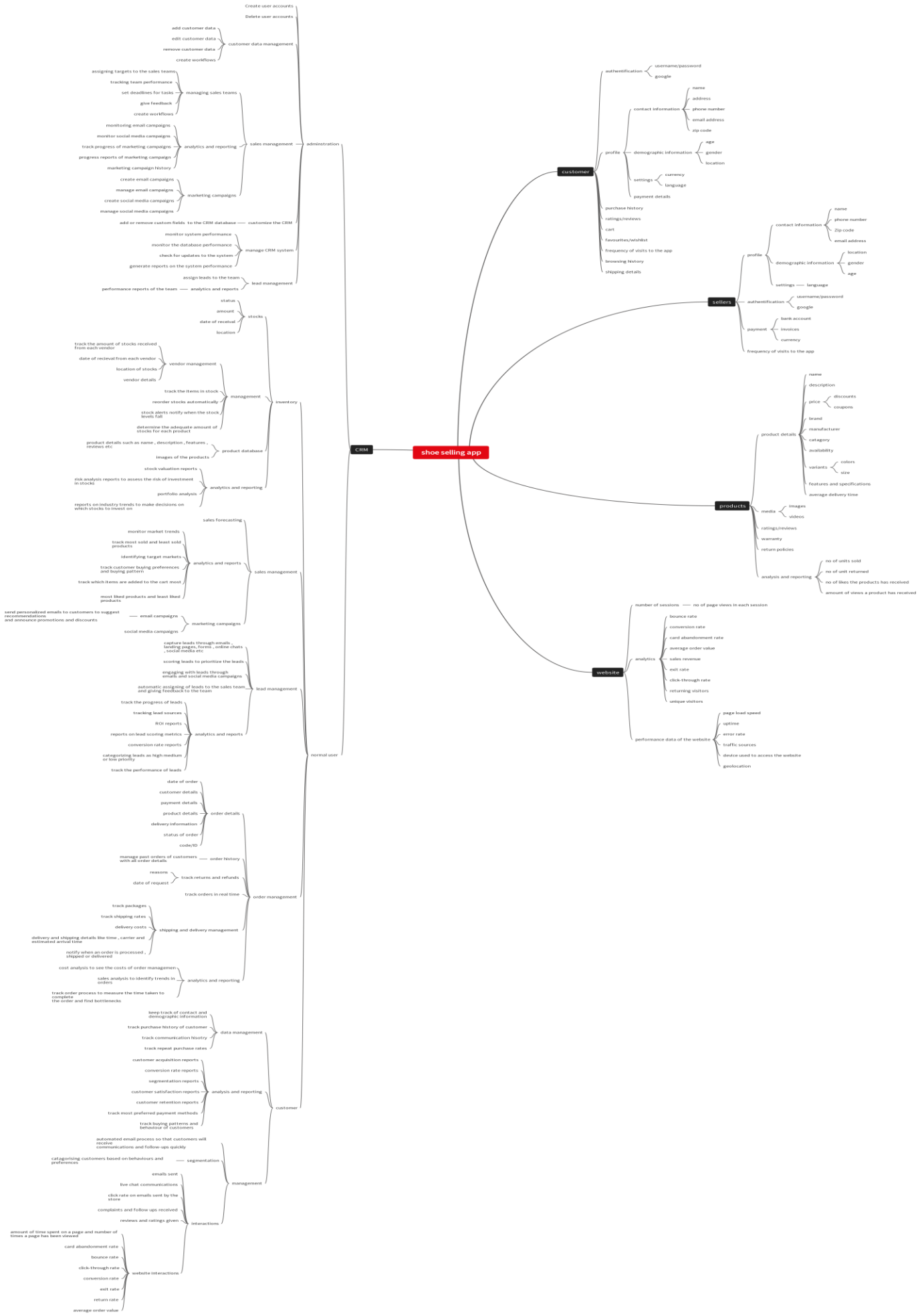
https://github.com/SanadhiCB008124/crm1.git

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 Datatabase

- 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

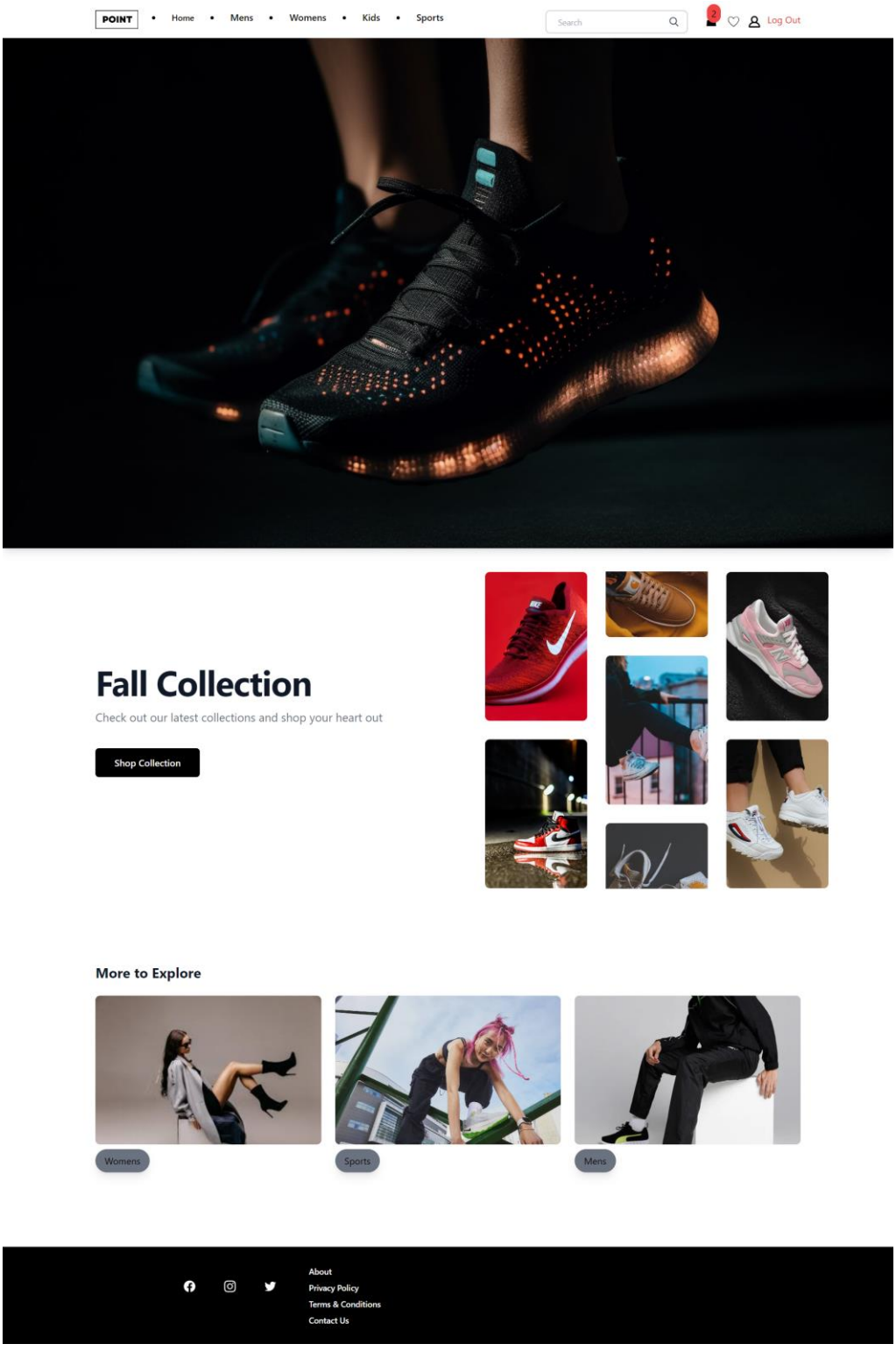


Figure 2 Homepage

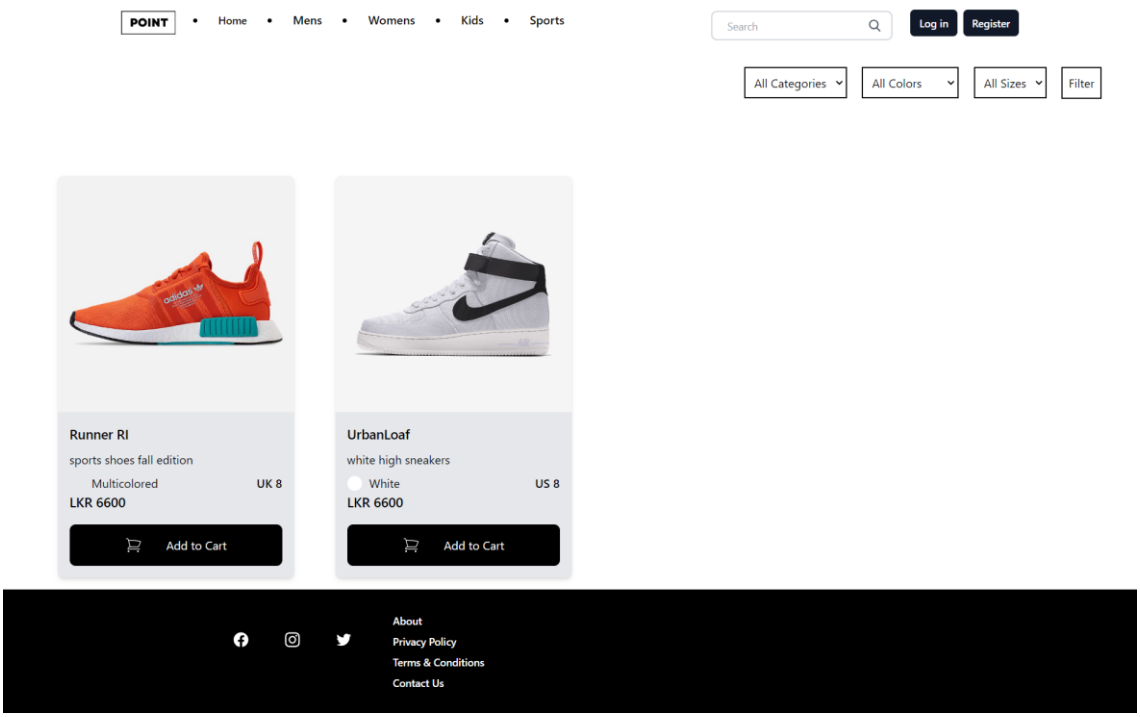


Figure 3 Mens'Products

Login

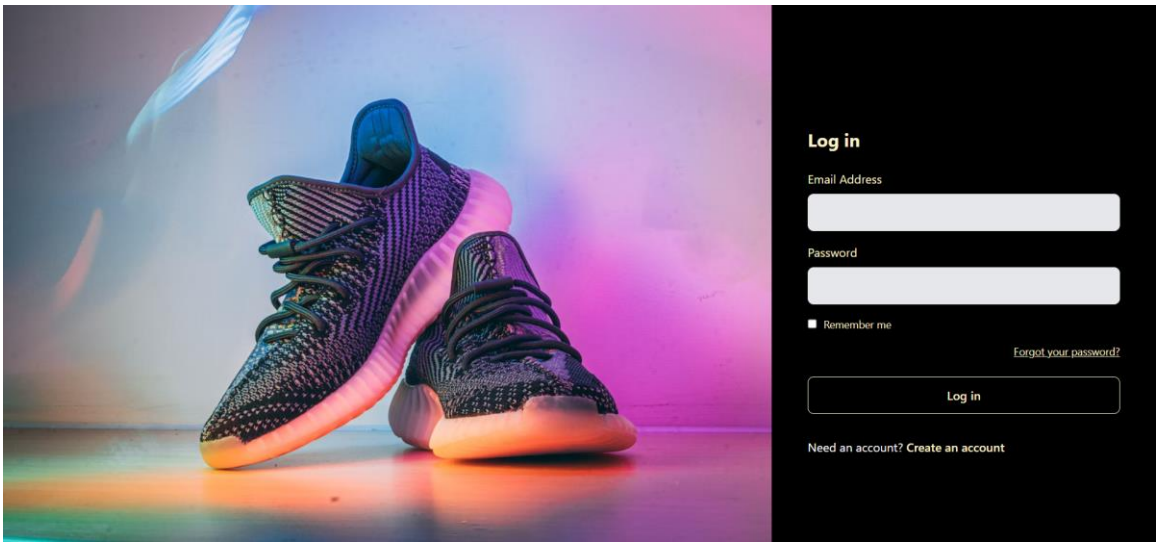


Figure 3 Login page

Register

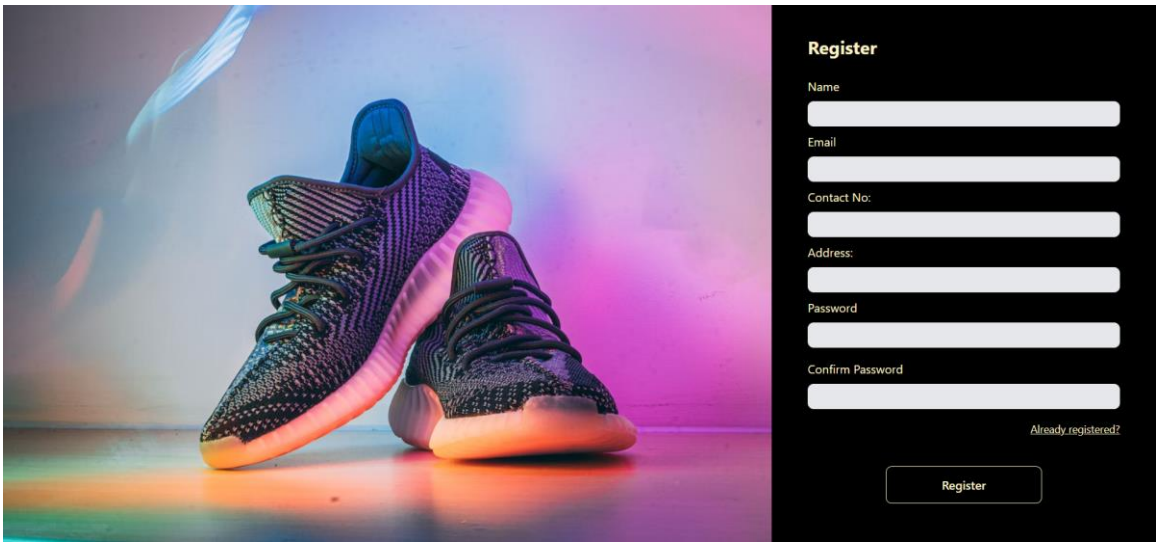


Figure 4 Register page

Product Description page

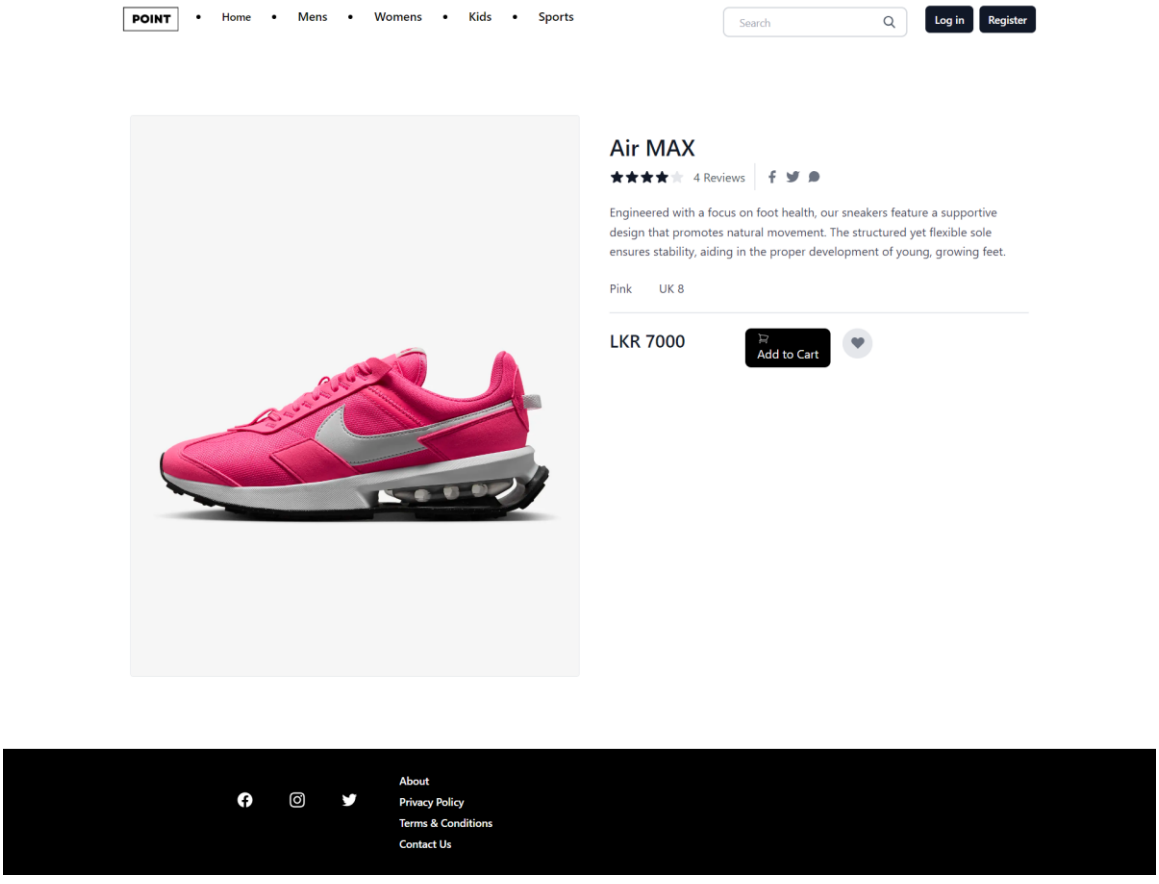


Figure 5 Product description page

Filter

Figure 6 Filter Options

Search bar

Figure 7 Search functionality

Cart

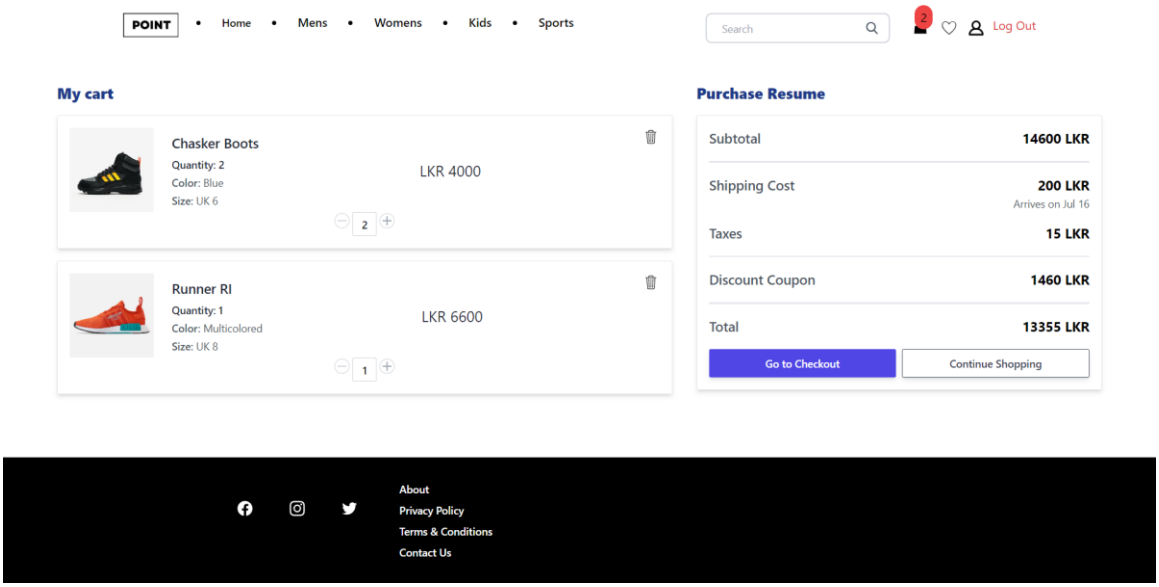


Figure 8 Cart

Navigation Bar

Figure 9 Navigation bar

Product Management

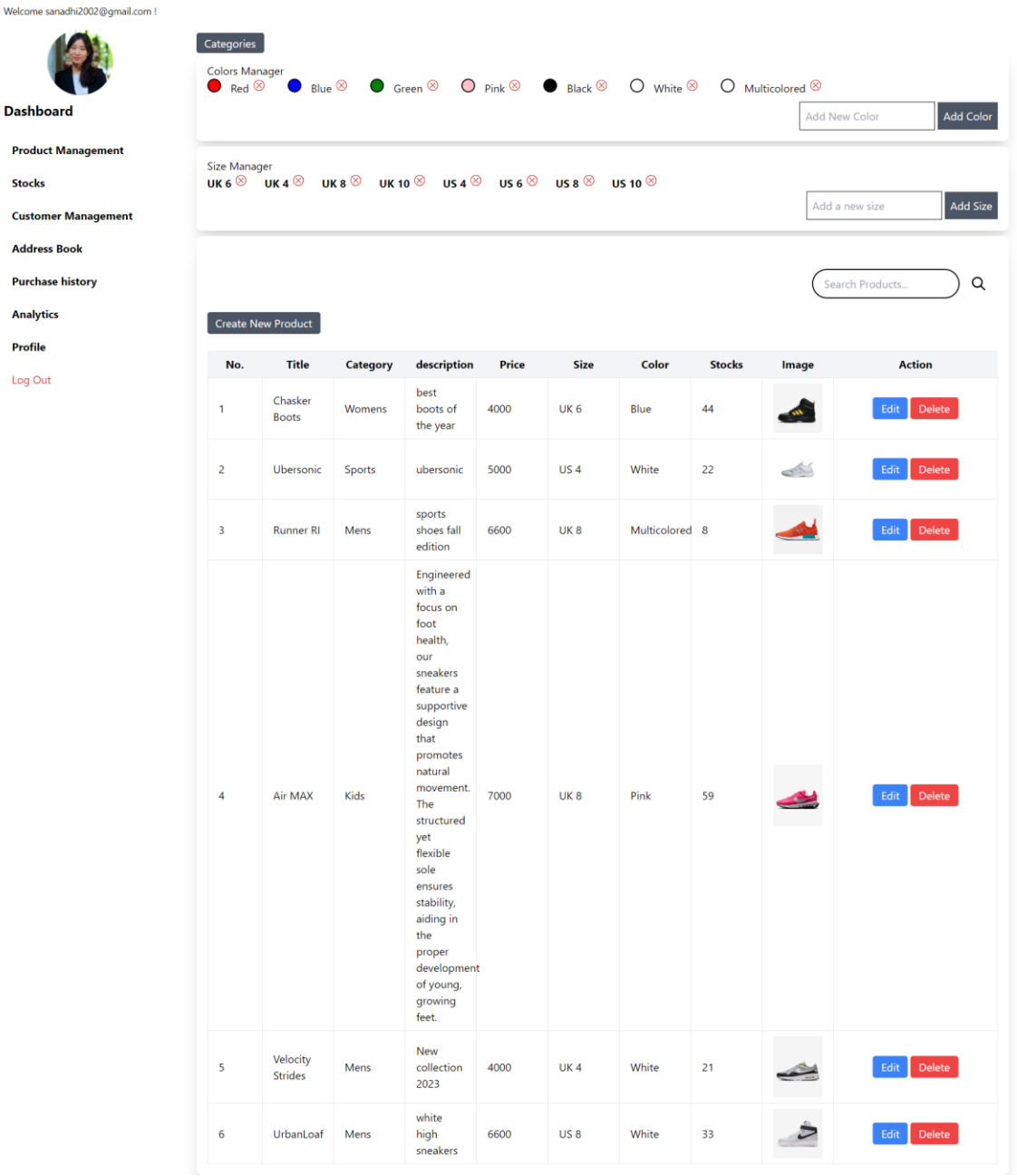


Figure 10 Product Management

Add product

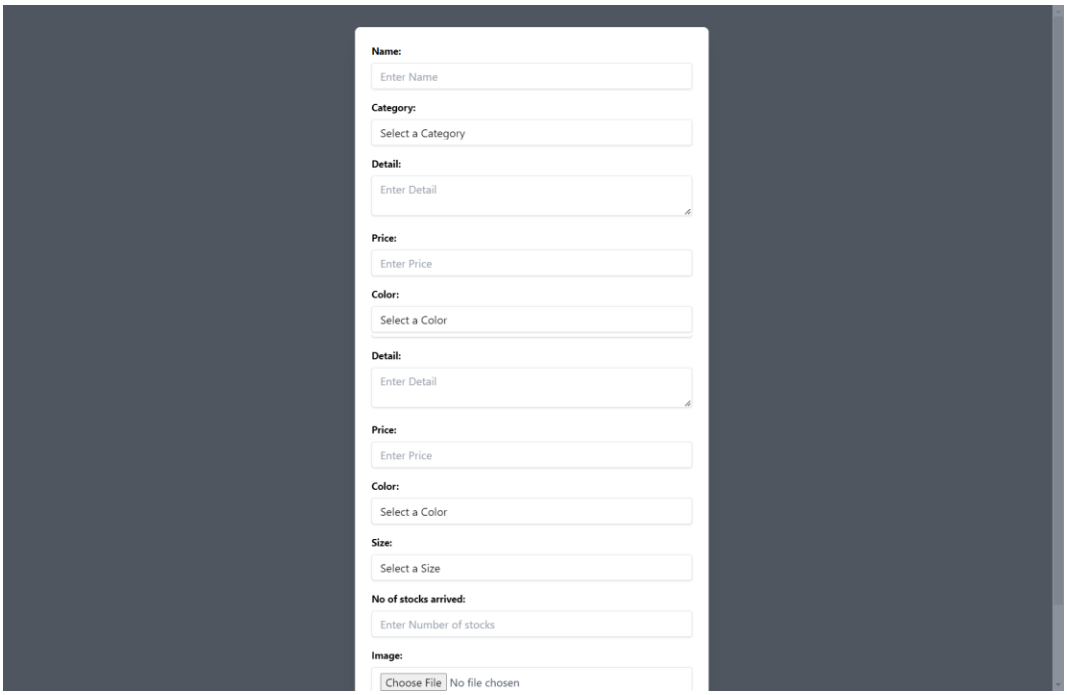


Figure 11 Product Crud continued

Category List

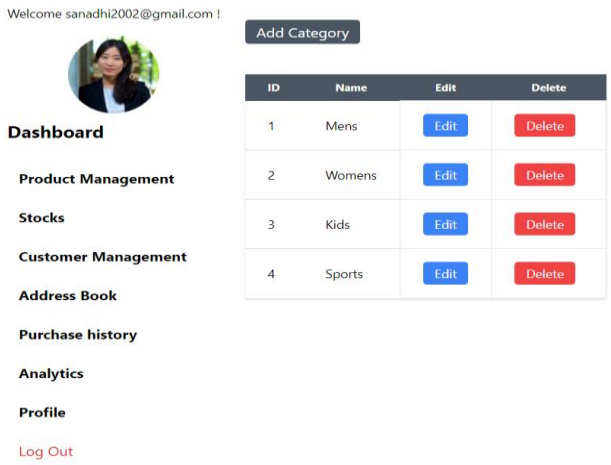
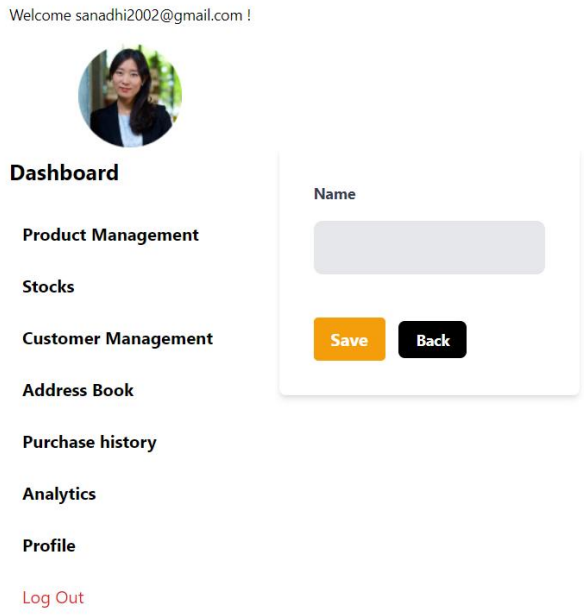


Figure 12 Category list

Add Category



Edit Category

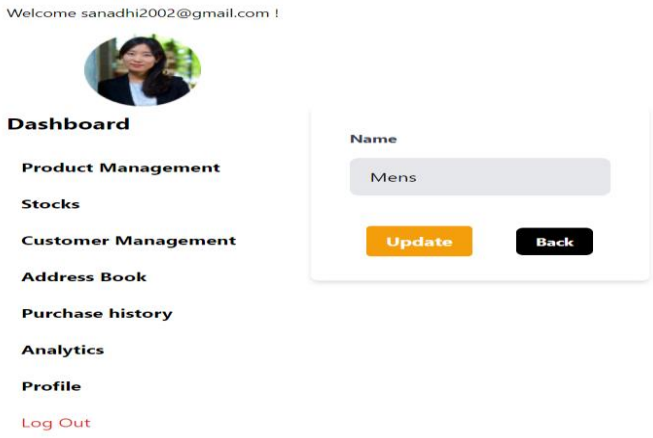


Figure 13 Edit category

Stock Mangement

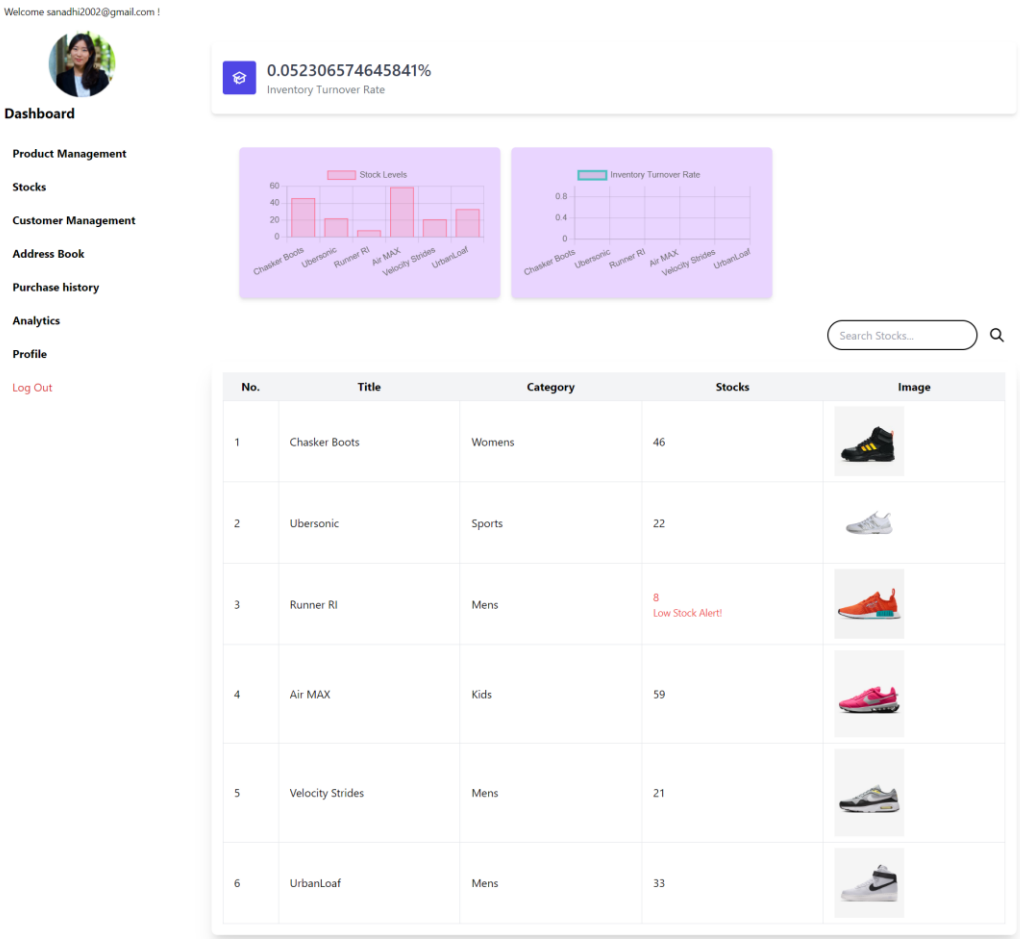


Figure 14 Stocks

Customer Management

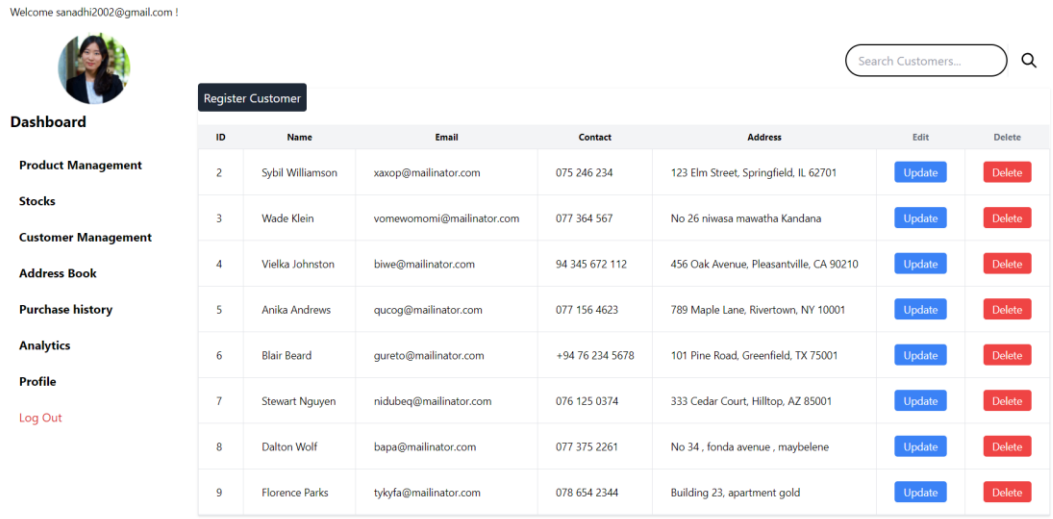


Figure 15 Customer Management

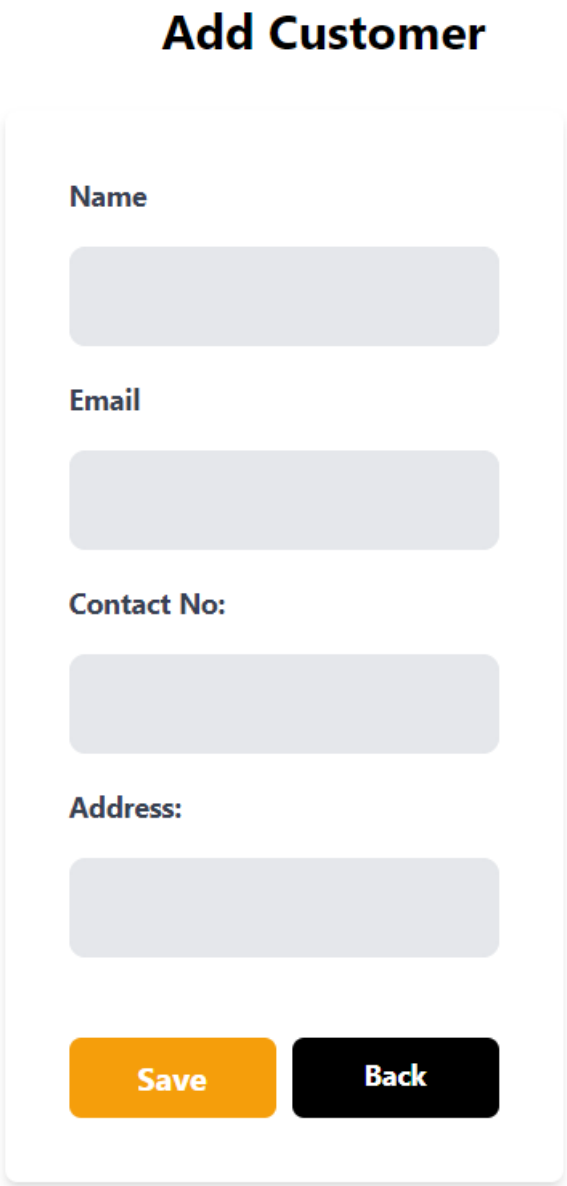


Figure 16 Add Customer

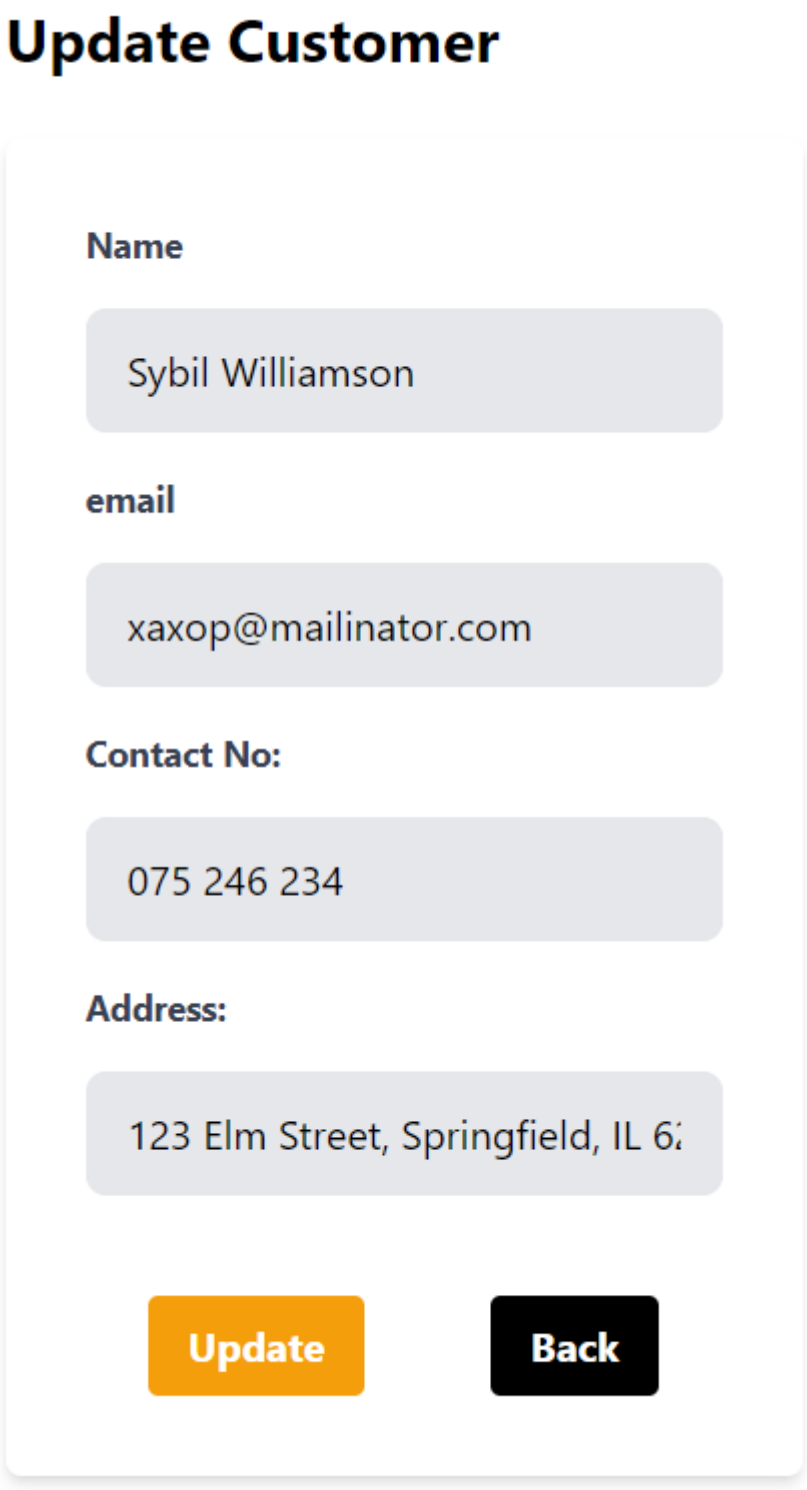


Figure 17 Edit customer

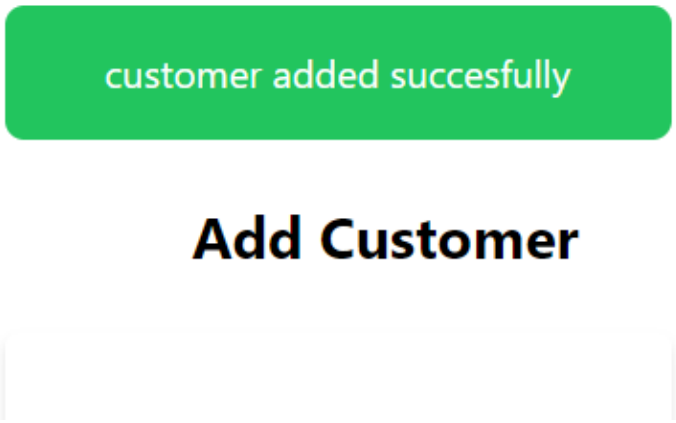


Figure 18 Success message

Address Book

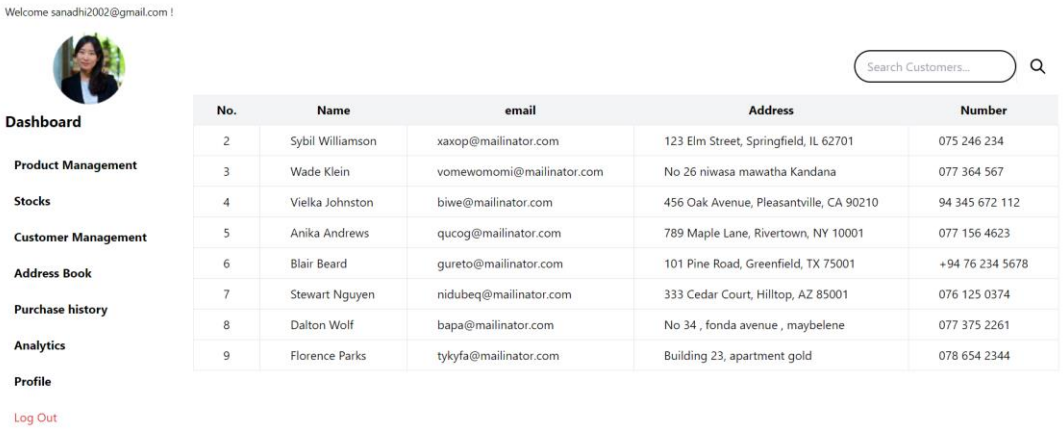


Figure 19 Address Book

Purchase history

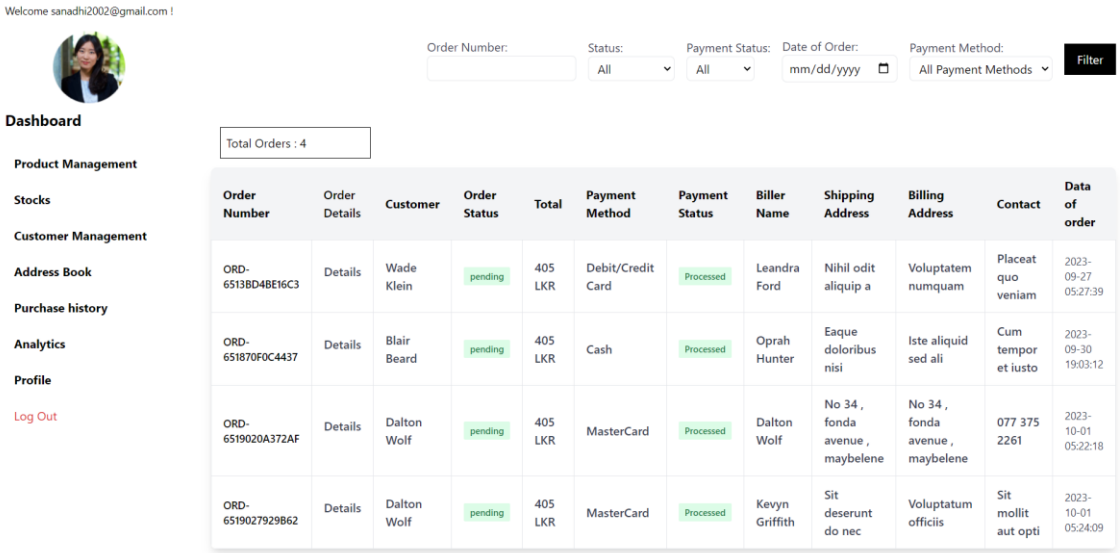


Figure 20 Purchase History

Customer pov purchase history

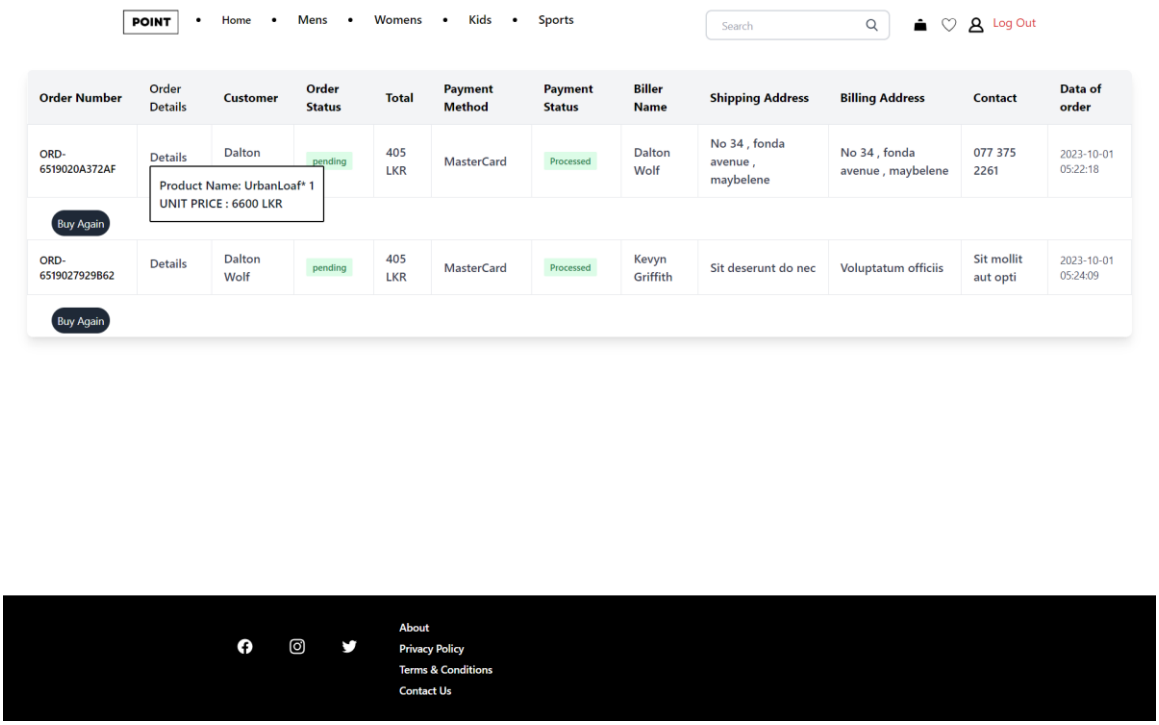


Figure 21 Purchase history 2

Checkout Page

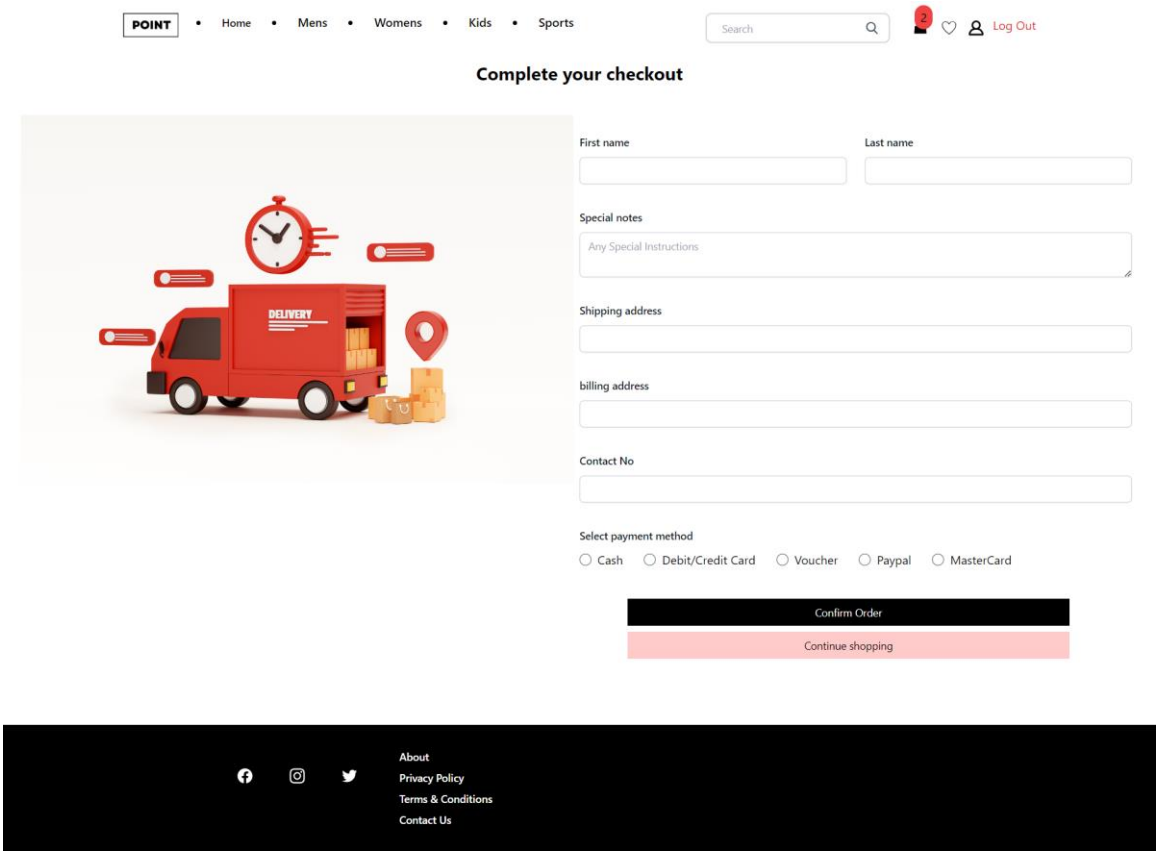


Figure 22 Checkout Page

Profile page

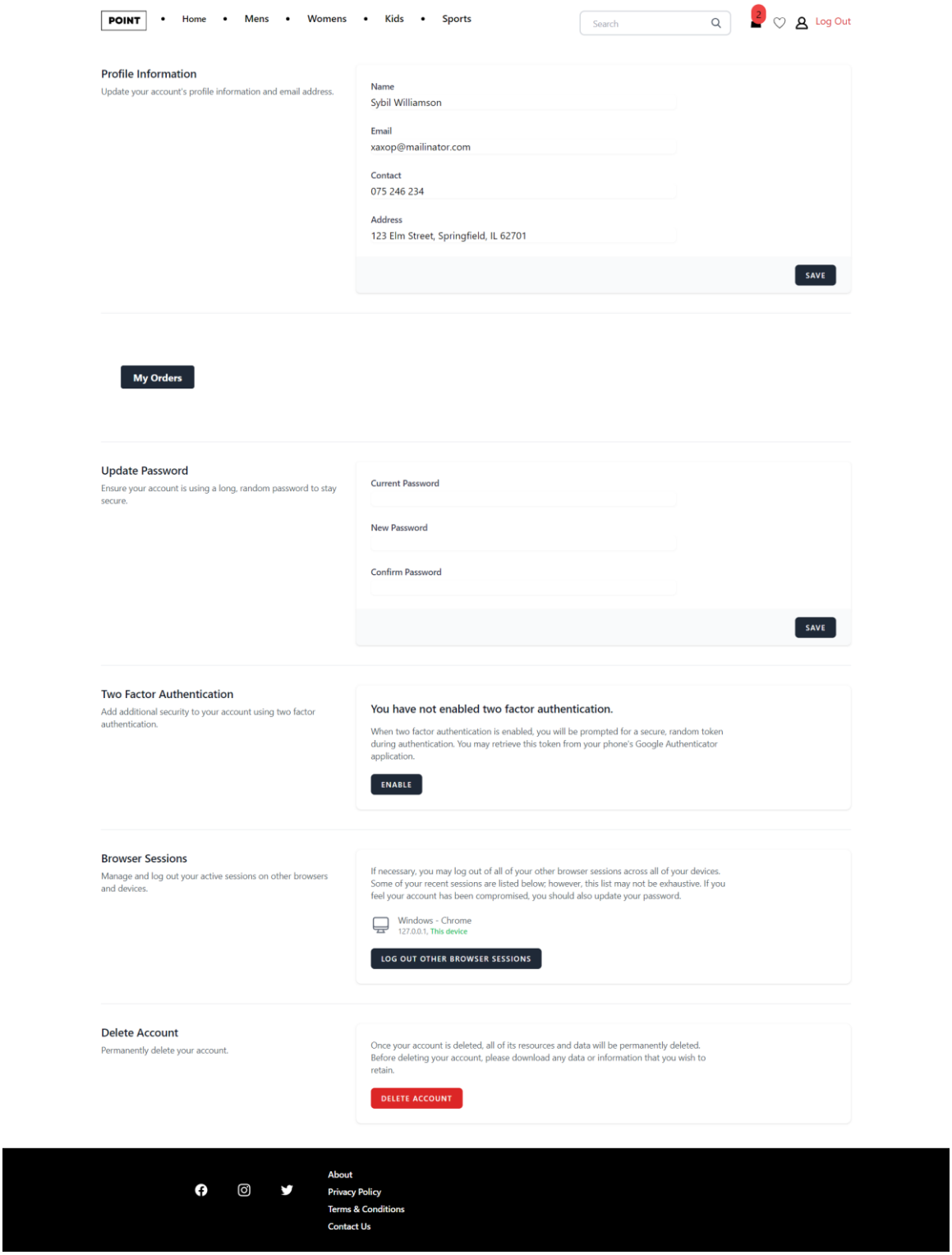


Figure 23 Profile page

Dashboard

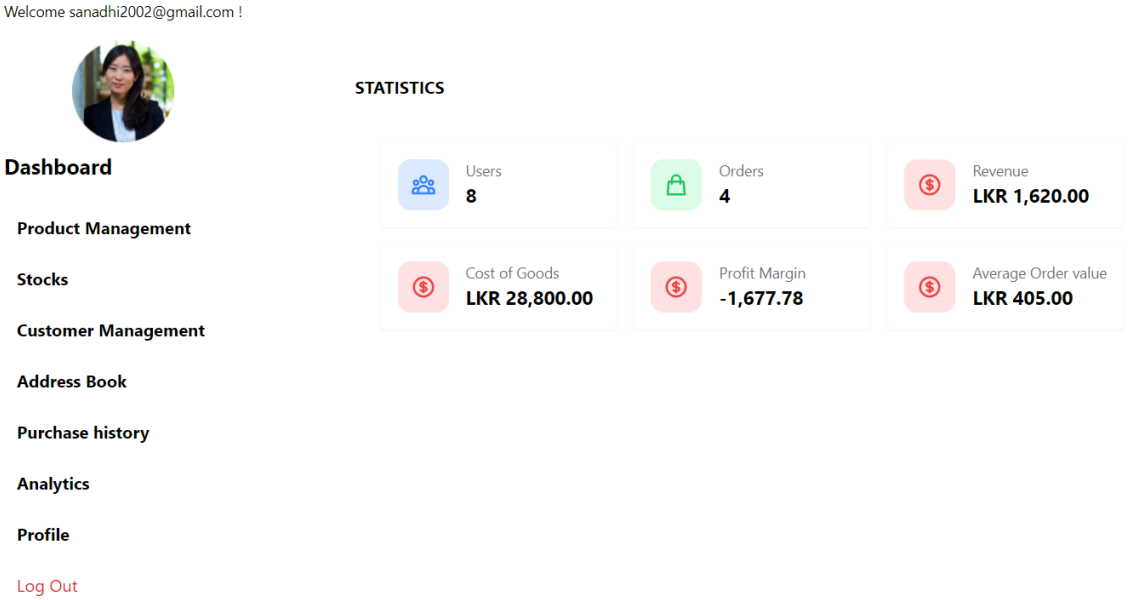


Figure 24 Dashboard

Analytics

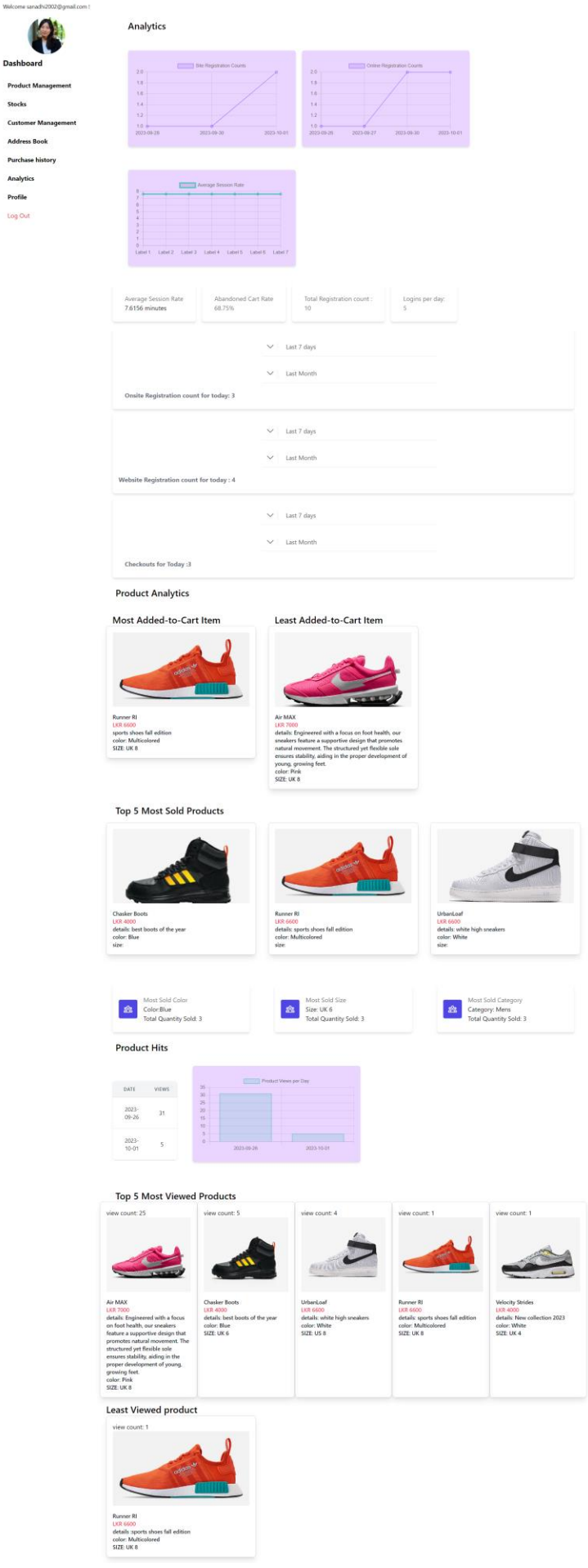


Figure 25 Analytics

MailTrap

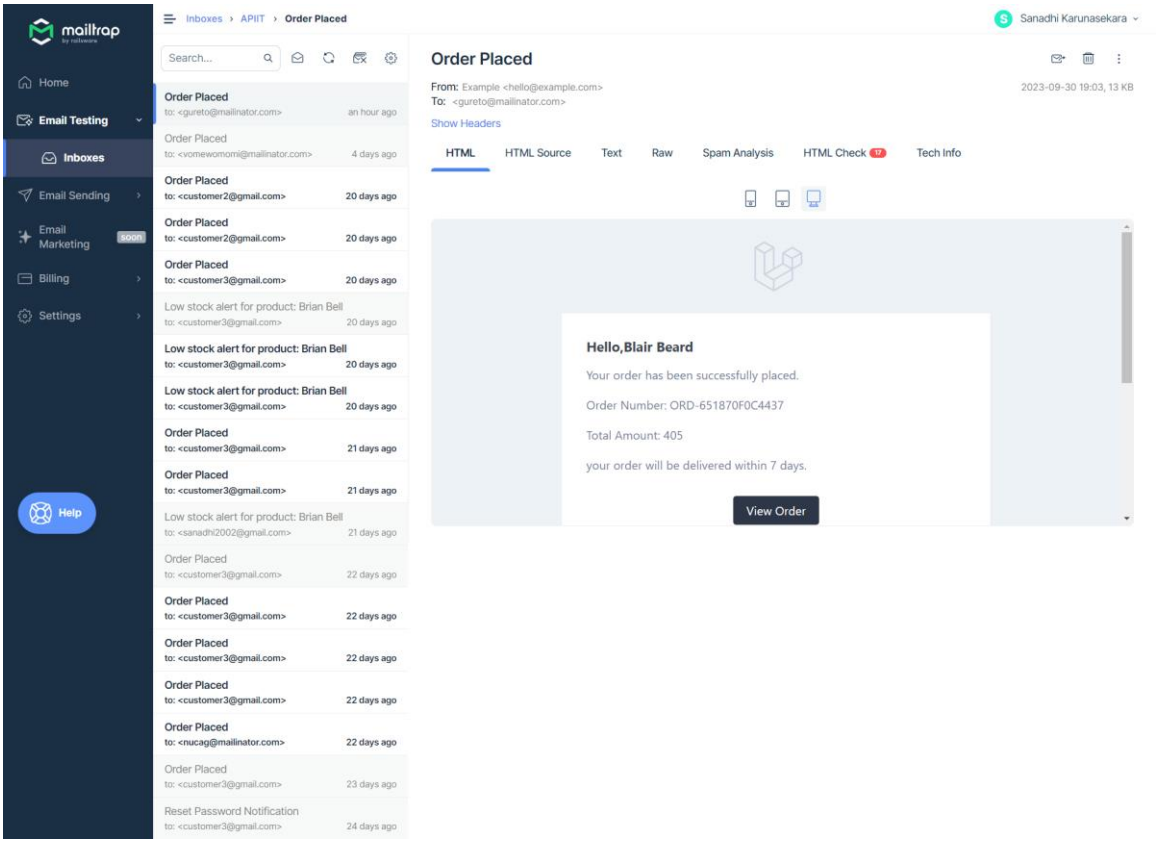


Figure 26 Mail testing

Controllers

Analytics Controller

Figure 27 Analytics Controller

Order Controller

Figure 28 Order Controller

Cart Controller

Figure 29 Cart Controller

Models

Product Model


```php
<?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');
    }



}
```

Figure 30 Product model

Size model

```php
<?php

namespace App\Models;

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

class Size extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',

    ];
}
```


Figure 31 Size model

Colors Model

```php
<?php

namespace App\Models;

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

class Color extends Model
{
    use HasFactory;

    protected  $fillable=[
         'color'
    ];
}
```


Figure 32 Color model

Category Model


```php
<?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',

    ];
}
```

Figure 33 Category model

Order Item Model


```php
<?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');
    }

}
```

Figure 34 Order Item model

Cart Item Model

```php
<?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');
    }

}
```


Figure 35 Cart Item Model

Listereners

Login Listener

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

```php
<?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($event)
    {
        if ($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.

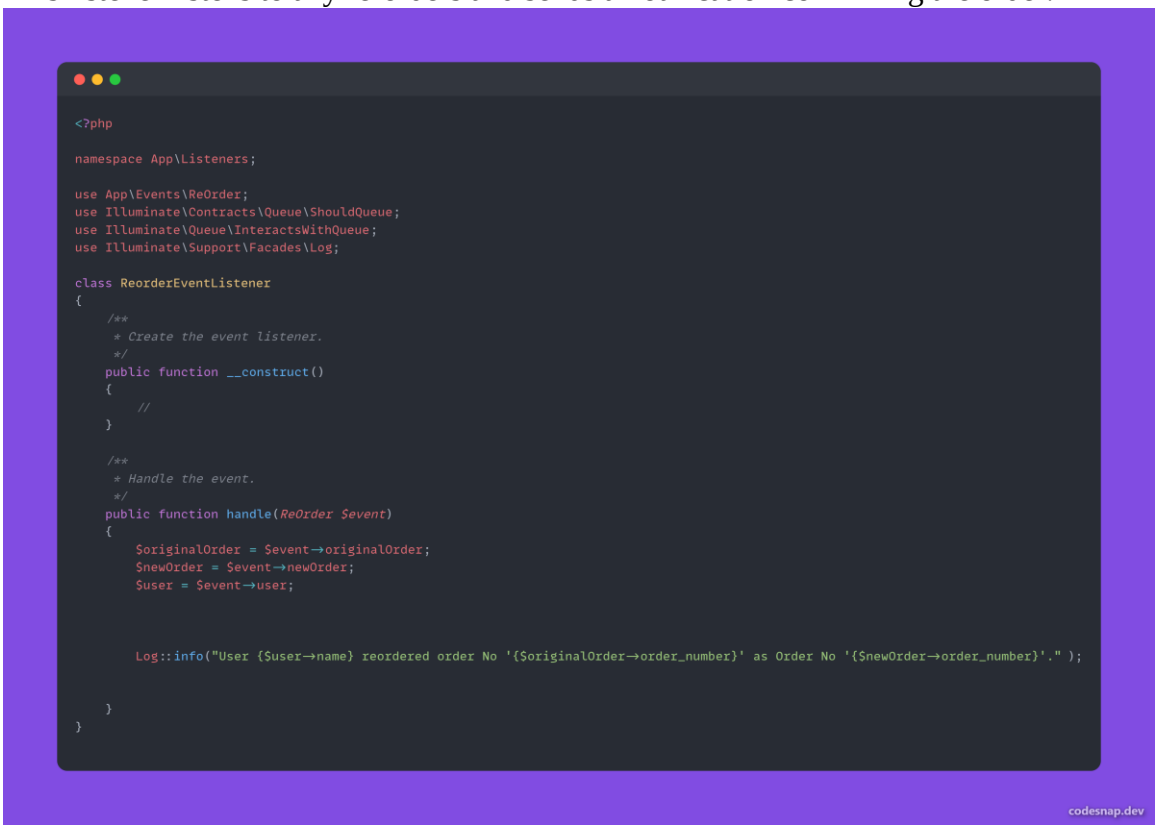
```php
<?php

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.




```php
<?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}'." );


    }
}
```

Figure 38 Re-order listener

Registration Listener

This listener listens to registrations happening through the website.

Figure 39 Registration Listener

Online Registration Listener

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

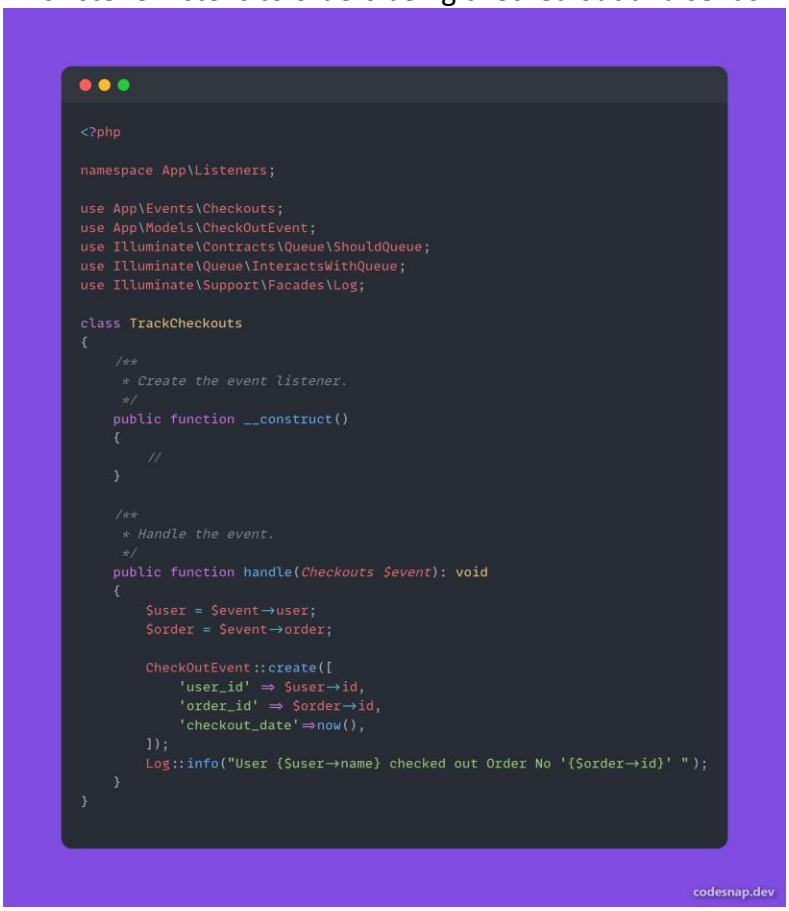
```php
<?php

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.



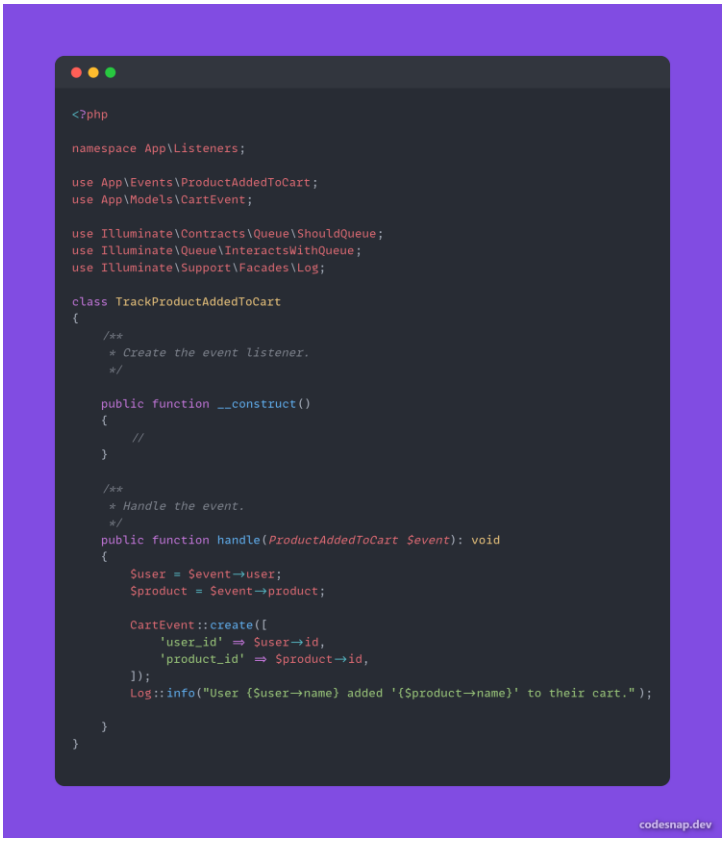
```php
<?php

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
<?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.");

    }
}
```

Figure 42 product added to cart listener

Events

Checkout Event


```php
<?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'),
        ];
    }
}
```

Figure 43 Checkout event

Low stock event


```php
<?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'),
        ];
    }
}
```

Figure 44 Low stock event

Online register event


```php
<?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'),
        ];
    }
}
```

Figure 45 Online register event

Site Register Event

```php
<?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.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('channel-name'),
        ];
    }
}
```

codesnap.dev

Figure 46 Site register event

Product added to cart event

Figure 47 product added to cart event

Re-order event

Figure 48 Re-order event

Notifications

Order placed Notification

Notification sent when a customer makes a purchase.

```php
<?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.

```php
<?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.

```php
<?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 ->to($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


```php
<?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');
    }
}
```

Figure 52 Product hits

Cache Routes

Routes are cached to increase performance and retrieve routes faster.

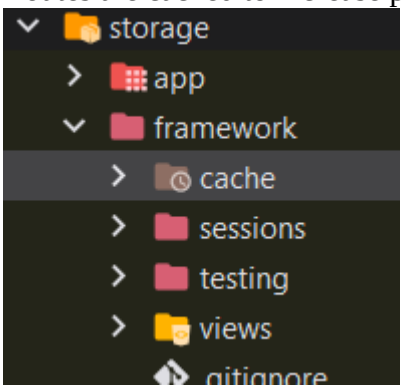


Figure 53 Cache Routes

Alpine Js

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

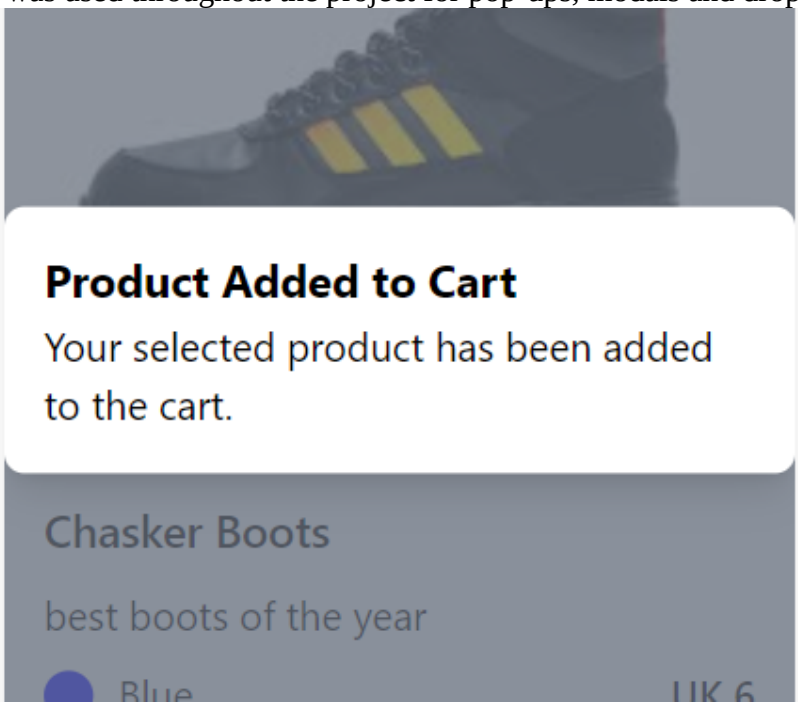


Figure 54 pop up

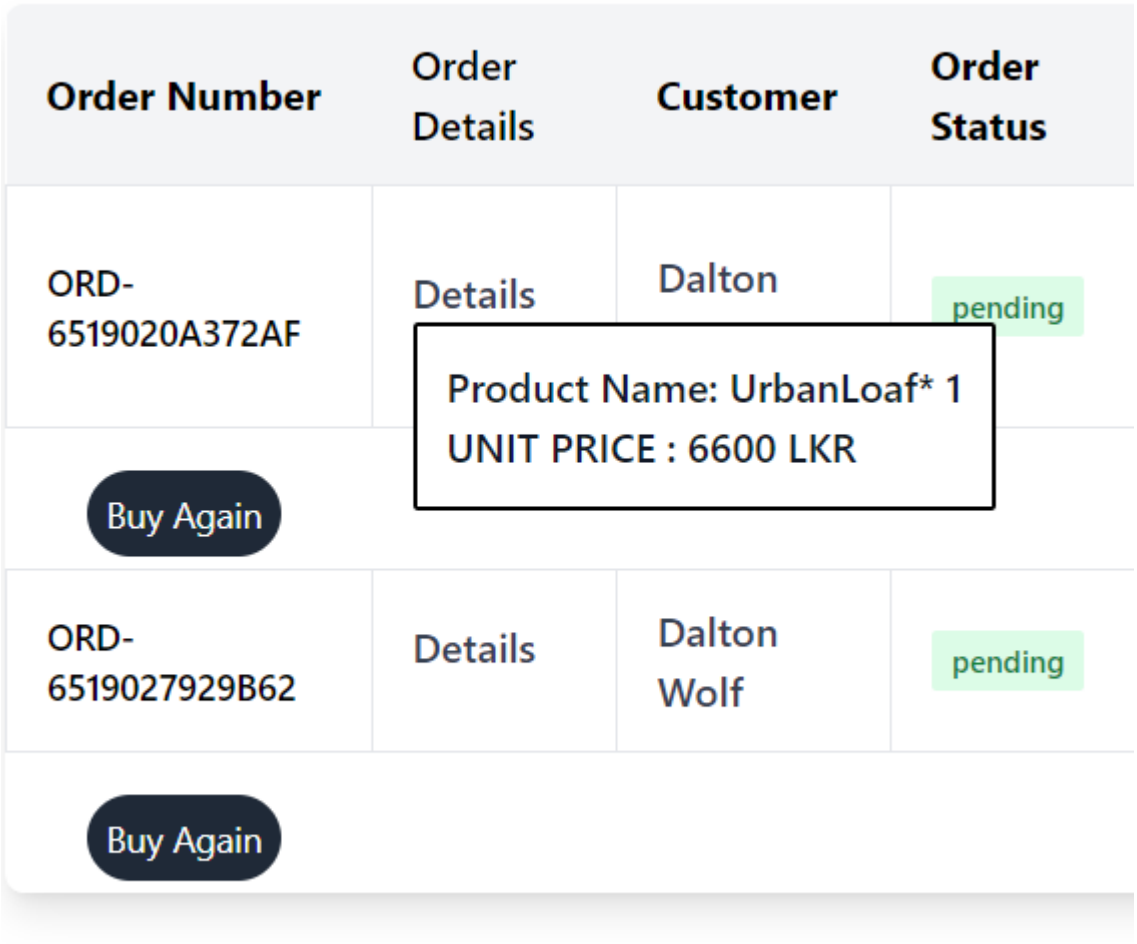


Figure 55 show

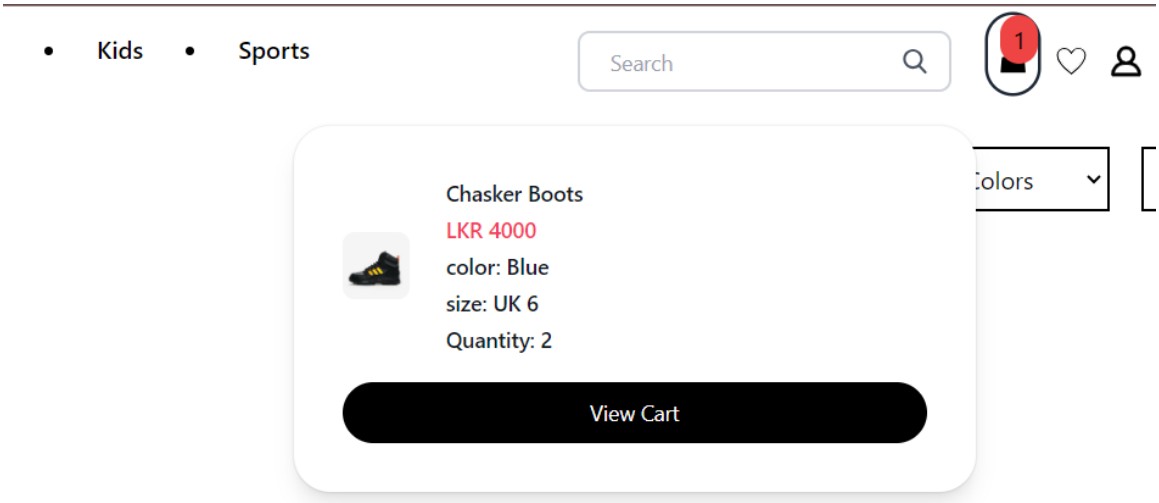


Figure 56 Dropdown

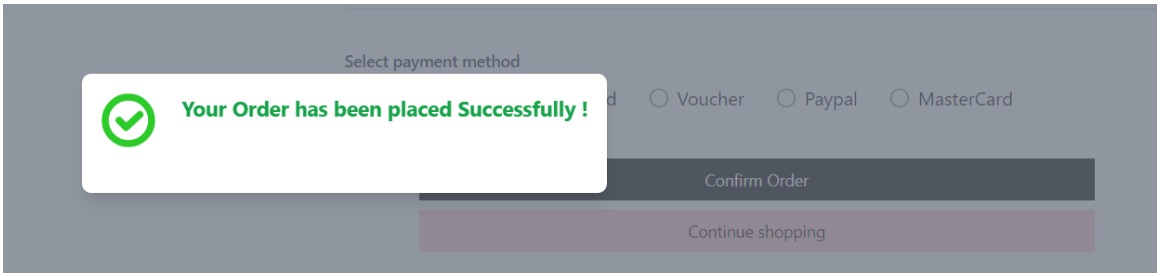


Figure 57 Pop up 2

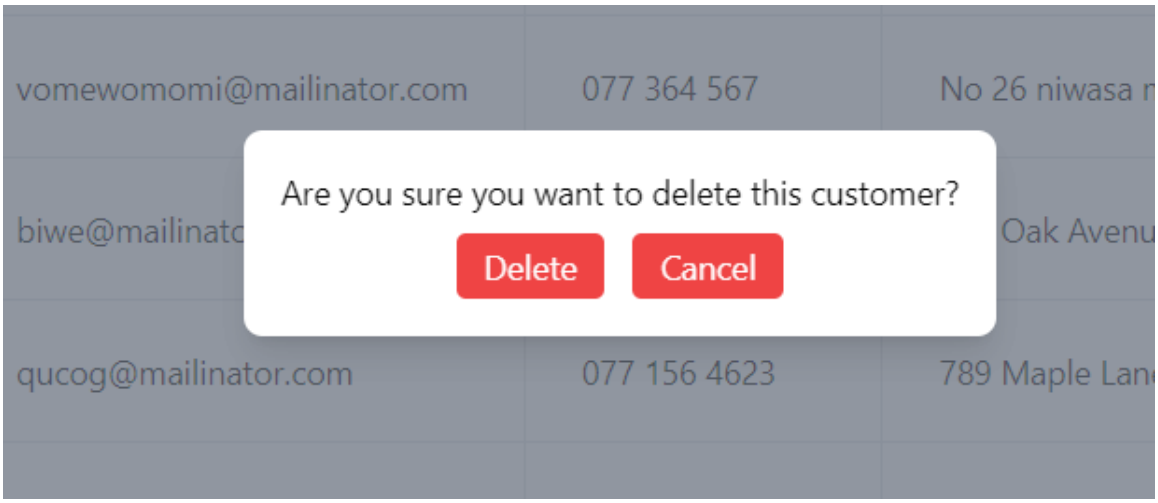


Figure 58 Modal

Chart Js

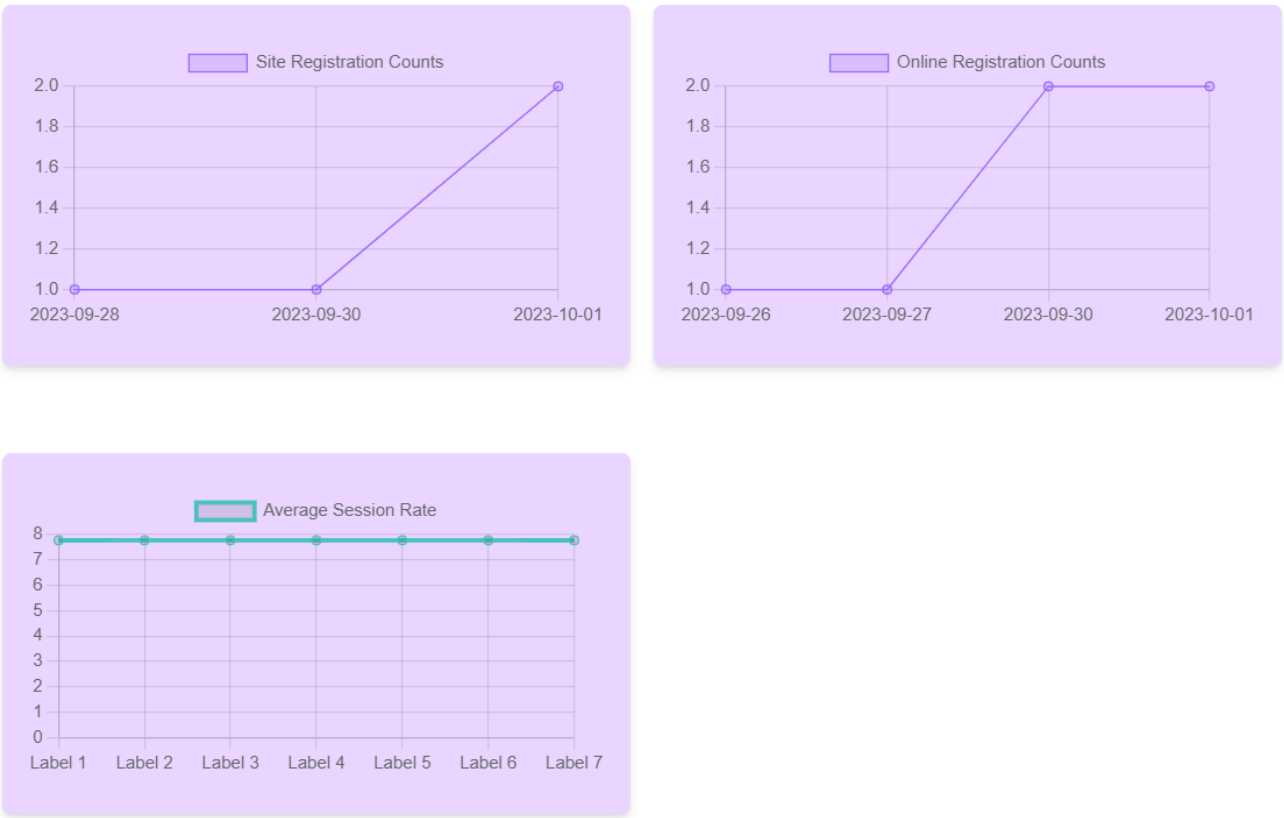


Figure 59 Chart Js

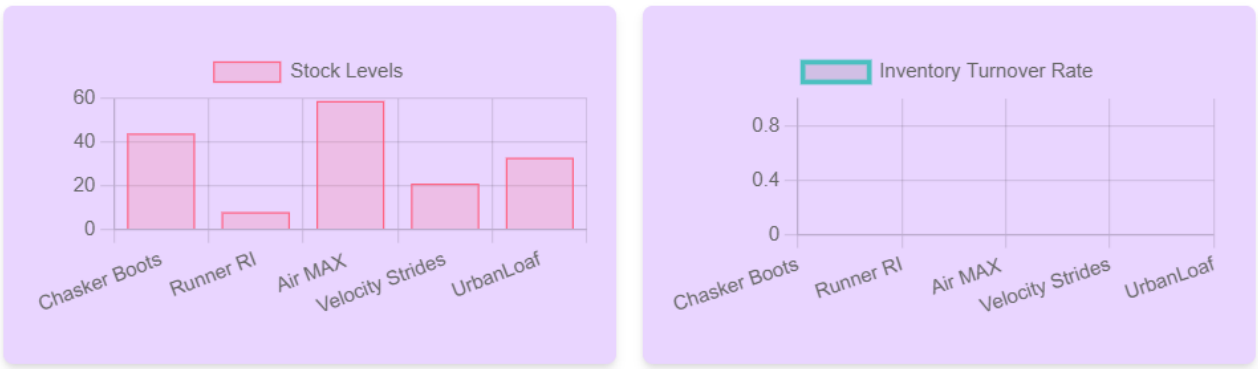


Figure 60 Chart js 2

Livewire

Livewire for product CRUD

Figure 61 Livewire product crud

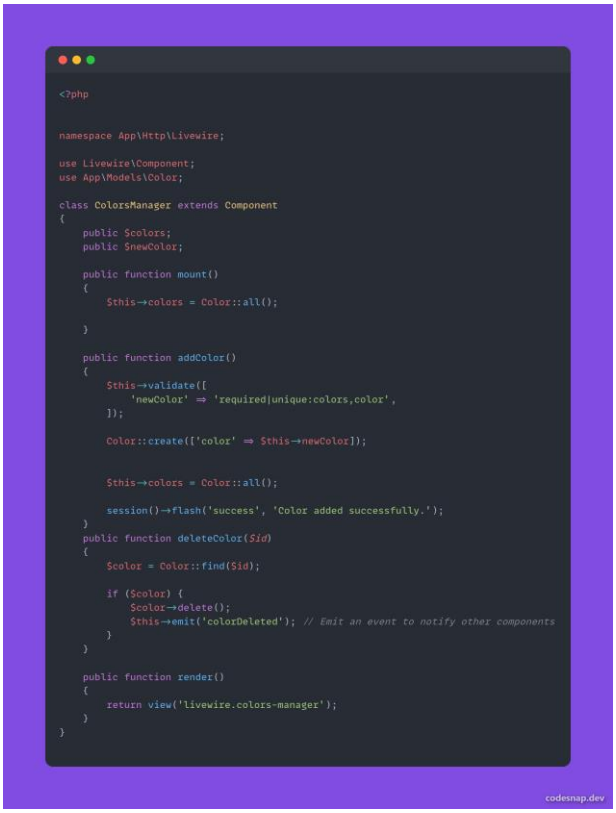
Livewire size CRUD

```php
<?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

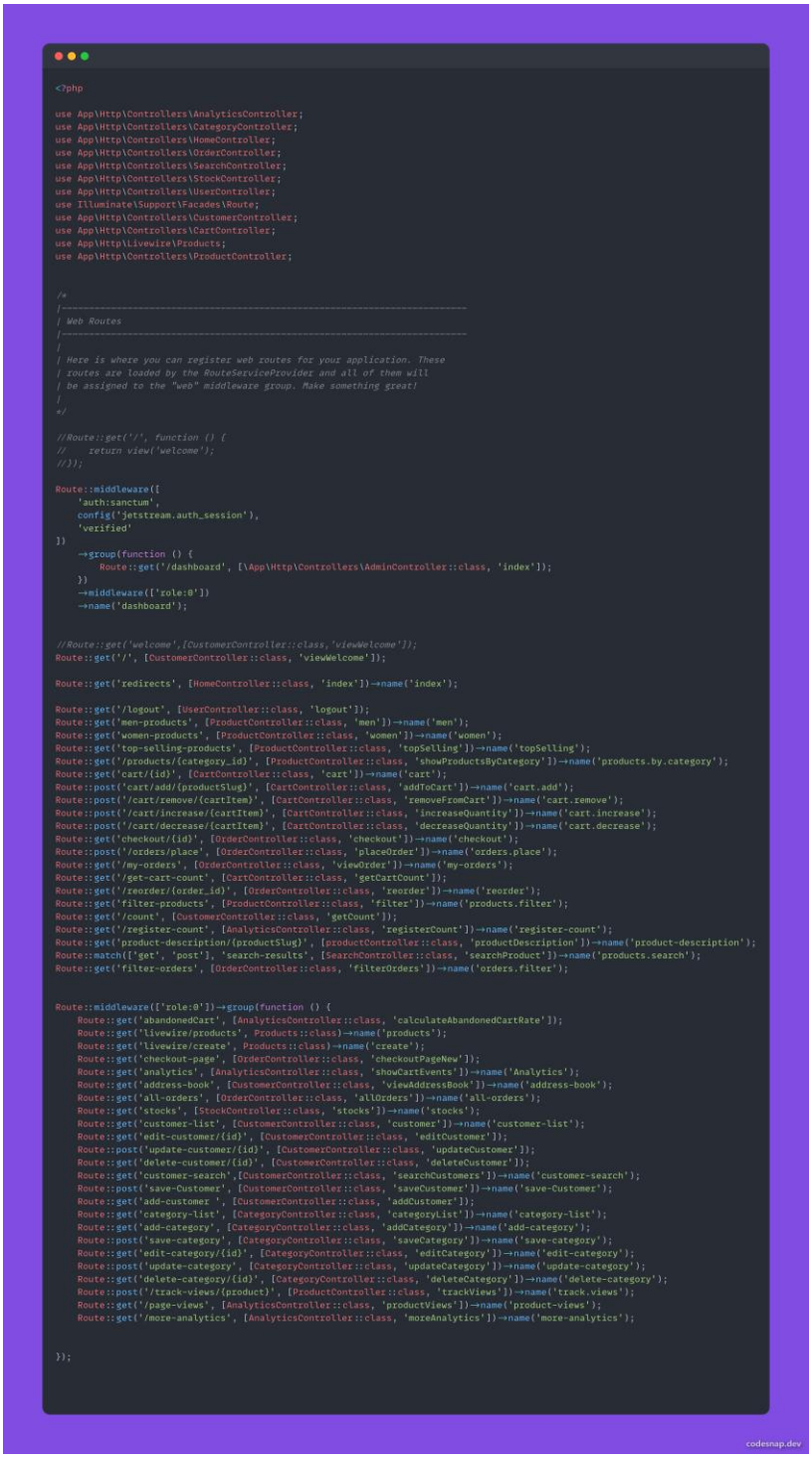
Livewire Color CRUD



```php
<?php


namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Colorz;

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');
    }
}
```


Figure 63 livewire color crud

Routes



```php
<?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\Http\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/{cartItem}', [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'])->name('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('delete-customer/{id}', [CustomerController::class, 'deleteCustomer']);
    Route::get('customer-search',[CustomerController::class, 'searchCustomers'])->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

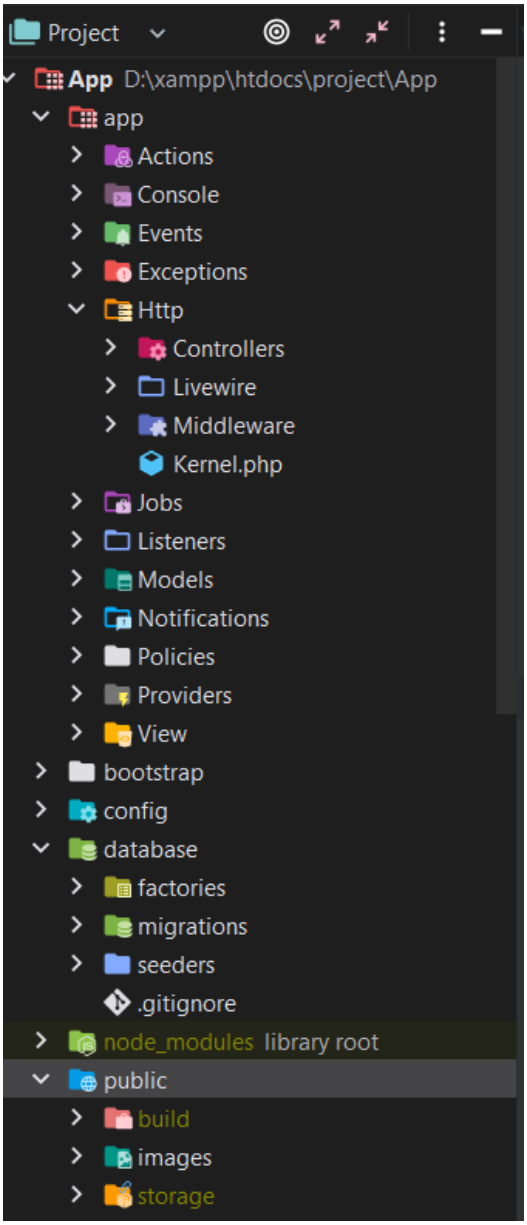


Figure 65 Code structure part 1

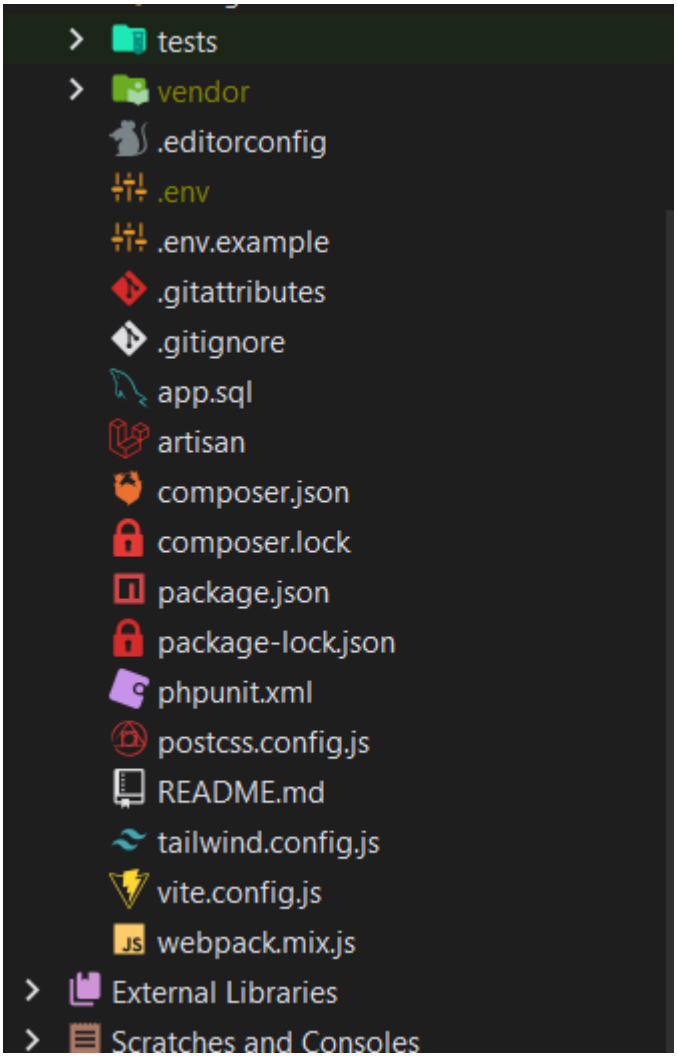


Figure 66 Code structure part 2

Migrations

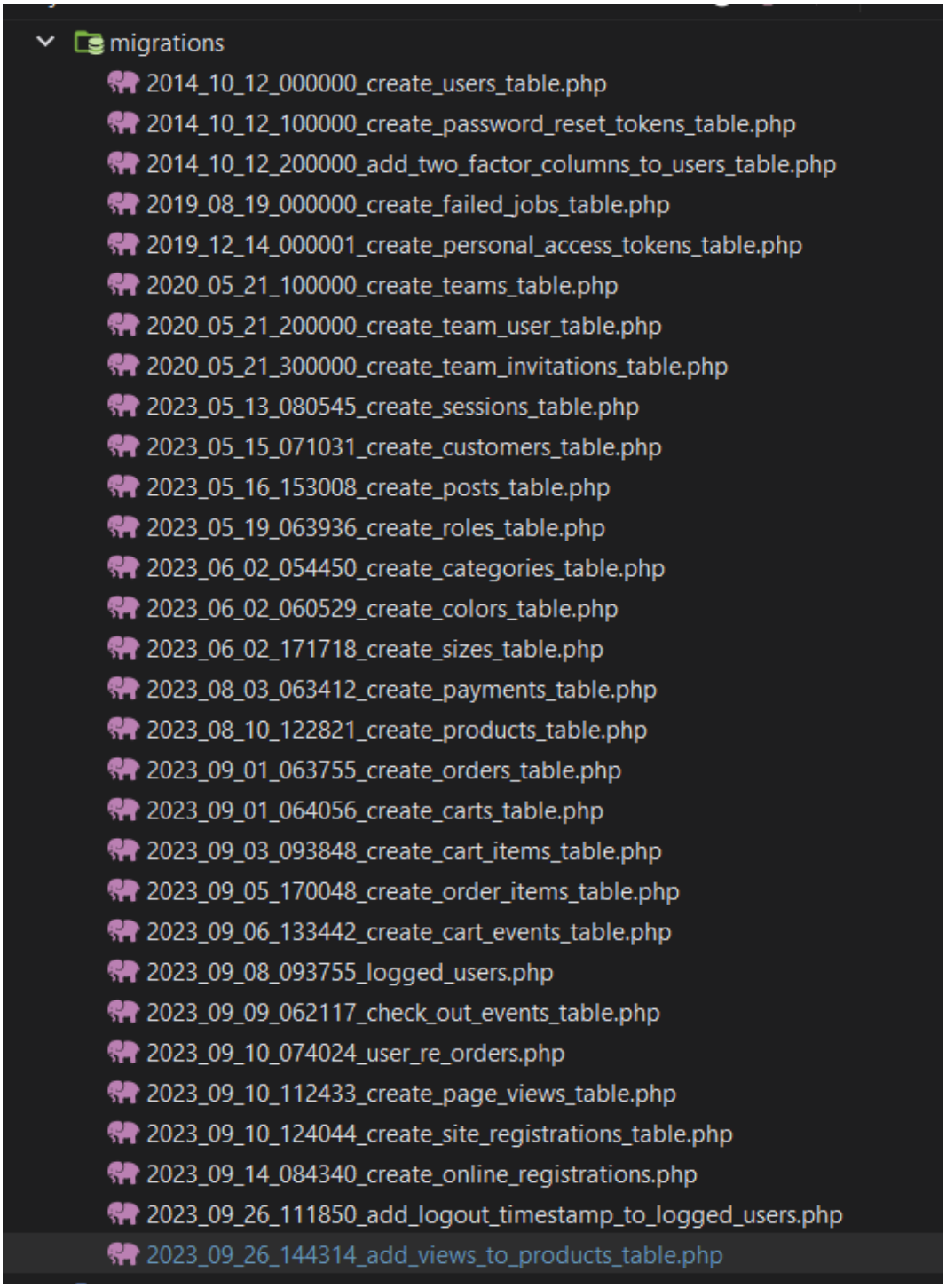


Figure 67 Migrations

MySQL database snapshot

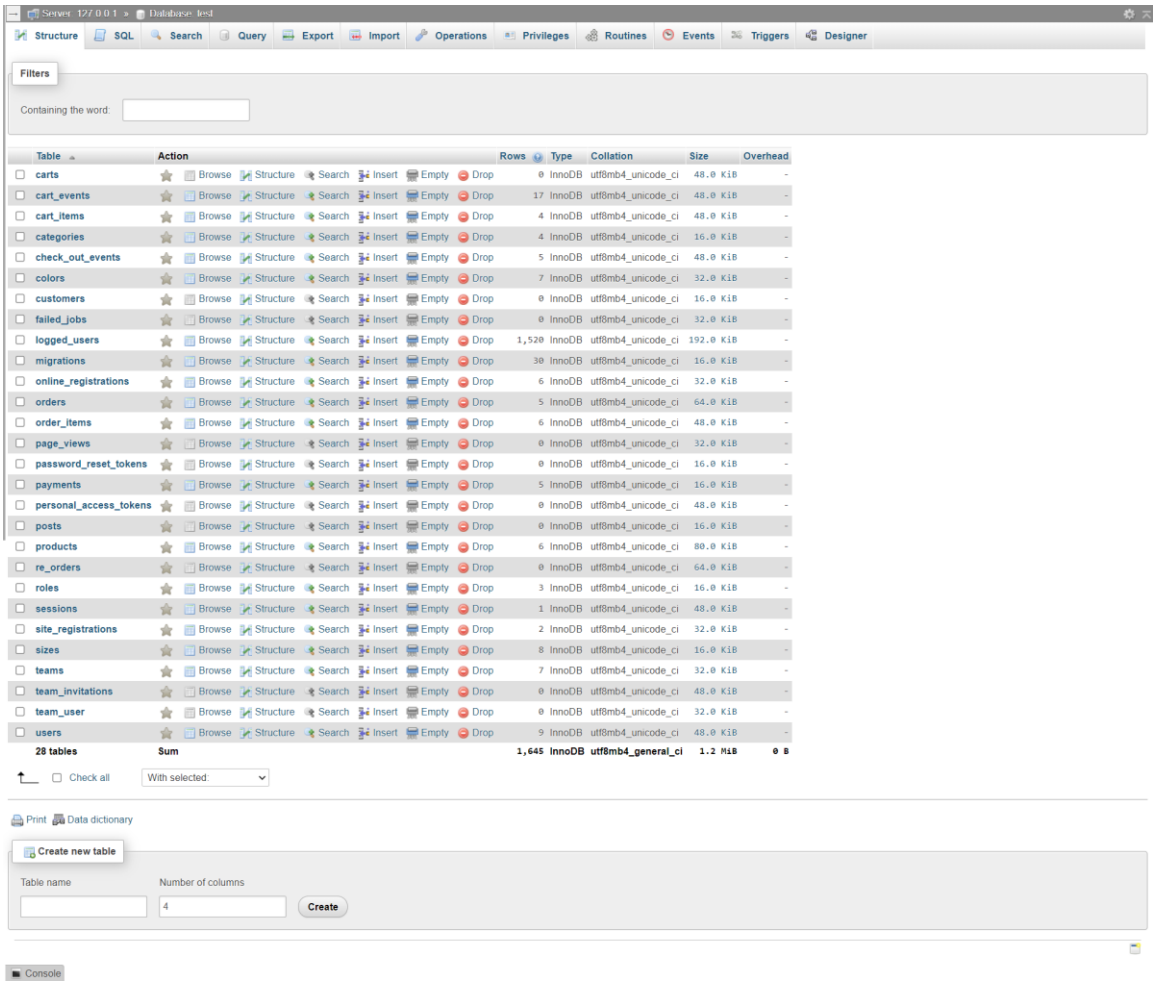


Figure 68 Database

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	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).