Web Programming Project

Project Proposal Template

	Full Name	ID	Section
Student 1	Nour Merhi	101220034	B
Student 2	none	none	

Course: CSC 400 – Web Programming
Instructor: Rayane El Sibai, Ph.D

Table of Contents

Chapter One: Proposal

1.1 Project Title

SmartShop: An E-Commerce Platform for Al-Amleyeh Hypermarket

1.2 Project Idea

This project focuses on the concept of **online shopping for Al- Amleyeh Hypermarket**, aiming to enhance the shopping experience for customers through a user-friendly digital platform. The application allows users to browse a wide range of products, add items to a shopping cart, and proceed to checkout, all from the convenience of their devices whether a laptop or phone.

The main problem suggested by this application is the inconvenience and time consumption of physically visiting supermarkets and searching for products. By offering a virtual alternative, the platform saves time, reduces effort, and adds flexibility to the shopping process.

One of the real-world motivations behind this project is the busy lifestyle of working parents. Managing both responsibilities and household tasks, especially while caring for children, leaves little time for traditional shopping. This application is designed to ease that burden by enabling them to complete grocery and household shopping quickly and efficiently from home.

1.3 Roles

1) Product manager(Admin): that manages the backend and overall operations. Can add, edit, or delete products, customers and delivery men information. Create accounts for both customer and delivery man.
2) Costumers : primary users of the app. Can browse for products in the shopping page or through home page, add items to cart, use coupons and then checkout.
3) Delivery personnel: that handles the delivery a shipping process. Can update delivery status, and communicate with costumers regarding delivery through their phone number that is displayed in the delivery dashboard.

1.4 Main Features

1.4.1 Guest:

- User Login and Registration: Allows new users to register an account and return users to log in page, or can logout if already logged in.
- Browse Products: Allowed to explore products and items available. No account needed.
- Product Details: See description, images, category, and price of the product through available items without needing an account.
- Cart: guests can add to cart products but have to login for purchases.

1.4.2 Registered User (Customer):

- Browse Products: navigate through shop page and see products, or directly buy featured or best sellers products displayed in home page.
- View Product Information: Explore detail information about products description, images, category and price.
- Add Items to Cart: Select product and add it into shopping cart for purchase directly from home and shop page. Or through the product's description page.
- Checkout: can proceed directly to checkout and enter his location for shipment.
- Email: Can receive a confirmation email about his order and the price.

1.4.3 Product Manager:

- Product Management: Add, edit, or remove products in his dashboard. Manage product details like price, description, and availability.
- Manage Customer Account: Add, view, and update customer accounts like email, phone number and account state.
- Manage Delivery Account: Add, edit, delete and view delivery accounts. Manage delivery details like vehicle type, state while delivering and phone number.

1.4.4 Delivery Personnel:

- Update Delivery Status: Refresh the status of orders.
- Manage Deliveries: Look at and handle the list of orders that have been given to them for delivering. Take order based on his choice and reserve this order.

- Customer Communication: Talk to customers about the delivery status or any problems with the delivery process through the phone number displayed in there order info.

Chapter Two: Design Phase

1.5 Frameworks used:

1.5.1 HTML (Hypertext Markup Language):

Used to build the structure and content of the web pages. Each HTML file defines the layout and elements present on the screen for the user to interact with. The following pages were developed:

i. **Home.html**: Includes a navigation bar with the SmartShop title, links to Home, Shop, and Contact pages, as well as buttons for Sign In (registration/login) and a cart icon. It also contains sections for the application's features, featured products, and best-selling items, along with a footer listing contact information.

ii. **Shop.html**: Displays all available products in the application, allowing customers to view details and add products to the cart.

iii. **Contact.html:** Contains the hypermarket's contact details including location, phone number, email, and working hours. It also includes a contact form for customer inquiries.

iv. **Register.html:** Provides a form for new user registration, including fields for name, email, password, and password confirmation.

v. **Login.html:** Includes a form for users to log in with their email and password.

vi. **productDescription.html**: Shows detailed information about a selected product including images, description, category, price, and quantity, with an option to add it to the cart.

vii. **Cart.html:** Displays all products added to the cart in a table format (image, name, price, quantity, total). Includes options to remove items, apply a coupon, and proceed to checkout.

viii. **Checkout.html:** Contains a form to collect shipping information required to place the order.

ix. **oderConfirm.html:** Displays an order confirmation and thank-you message after successful checkout.

x. **AdminDashboard.html:** Provides admin tools including modals for viewing products, customers, and delivery info. It includes forms to add, update, or delete products and user accounts, and control buttons for management.

xi. **DeliveryDashboard.html:** Displays the delivery profile (ID and name), and a table listing all customer orders with details such as order ID, customer information, shipping info, total price, delivery man ID, and order state. Includes buttons to take and update orders.

1.5.2 CSS (Cascading Style Sheets):

Was used to design and style the interface of the web application, ensuring visual consistency and a user-friendly experience.

i. **Style.css:** contains styling rules for the Home, Shop, Contact, Cart, Checkout, and OrderConfirm pages.
ii. **Register.css**: Provides styles for the Register and Login pages.
iii. **Dashboard.css:** Contains all styling for the Admin and Delivery Dashboard pages.

1.5.3 Js (JavaScript):

Used to make the application interactive and dynamic, enabling real-time changes in the UI based on user actions.

A single script.js file includes:

- Functions to manage toggling between tables and forms in the Admin Dashboard.
- Password validation (e.g., checking length) in registration and account creation forms.
- Image toggling functionality in the Product Description page.

1.6 Responsiveness:

Website application fits the requirements for responsiveness. Where the designing of the pages is responsiveness and fits variable screen sizes like phones and tablets. The CSS techniques used such as flexible layouts and media queries to adjust how the content looks. This helps users have a smooth and easy experience when they shop using their mobile devices, without needing to zoom in or scroll too much.

Chapter Three: Implantation

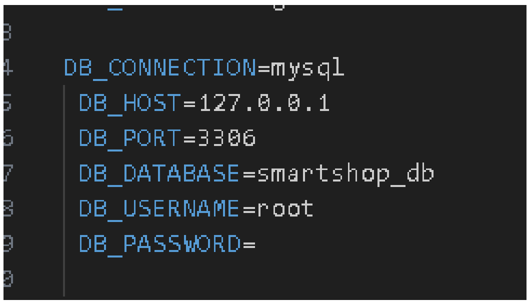
1.7 Laravel Setup:

1.7.1 Installing prerequisites:

- Xampp server for providing the environment needed to run PHP-based web applications locally on my computer.
- Composer: for managing and updating libraries and packages required for my Laravel project, and automatically set up all needed fodders for the project
- Noje.js & NPM: installed to manage frontend libraries, manage and compile all frontend assets needed like CSS and JavaScript.
- VS Code: installed for its built in extensions, integrated terminal and support for features like syntax highlighting, debugging and more important its ability to open live server.

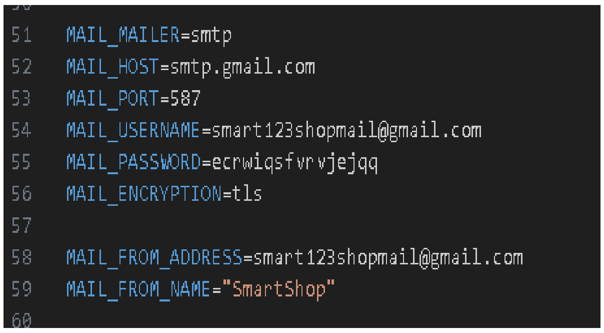
1.7.2 Creating Laravel project smartShop:

- Opening the folder web_projects to save Laravel project in it, then running the following command to setup Laravel project using composer: `composer create-project Laravel/Laravel SmartShop`
- Setting the environment file by running the following command first to create a copy of the .env.example folder: `copy .env.example .env`, then configuring my database connection within the following:



```
DB_CONNECTION=mysql
 DB_HOST=127.0.0.1
 DB_PORT=3306
 DB_DATABASE=smartshop_db
 DB_USERNAME=root
 DB_PASSWORD=
```

- After that configuring my mail connection with the Gmail server for the project to be able to send emails for users who purchase an order using the website’s mail **smart123shopmail@gmail.com**:



```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=smart123shopmail@gmail.com
MAIL_PASSWORD=ecrwiqsfvrvjejqq
MAIL_ENCRYPTION=tls

MAIL_FROM_ADDRESS=smart123shopmail@gmail.com
MAIL_FROM_NAME="SmartShop"
```

1.7.3 Setting up Authentication (Laravel Breeze):

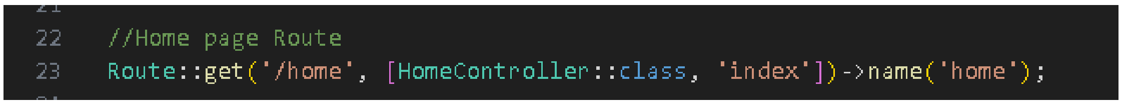
- Installing Laravel breeze package via composer using this command: `composer require Laravel/breeze --dev`
- Installing breeze’s authentication into my Laravel project to generate the necessary files for my views, controllers and routes needed for authentication: `php artisan breeze/install`
- Installing Node.js dependencies for my project: `npm install`
- Compiling frontend assets (JavaScript, CSS) for development: `npm run dev`
- Running database migration to create all necessary tables for authentication: `php artisan migrate`

1.8 Routing:

Defining all application routes in `web.php` file to connect each URL to a specific controller method:

1.8.1 Home page routes:

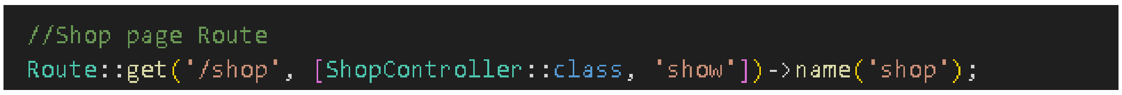
one route responsible to display the home page of the website



```
//Home page Route
Route::get('/home', [HomeController::class, 'index'])->name('home');
```

1.8.2 Shop page route:

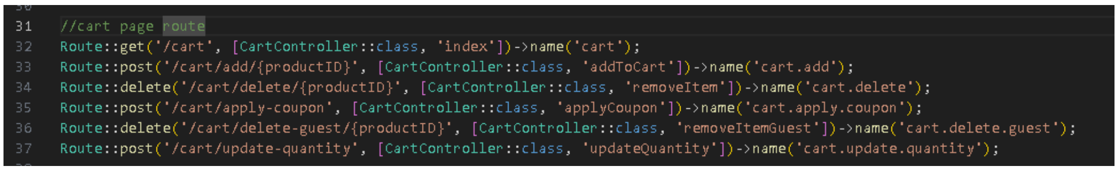
responsible for displaying shop page and fetching all necessary products to be displayed also in the page using show method defined in the route



```php
//Shop page Route
Route::get('/shop', [ShopController::class, 'show'])->name('shop');
```

1.8.3 Cart page routes:

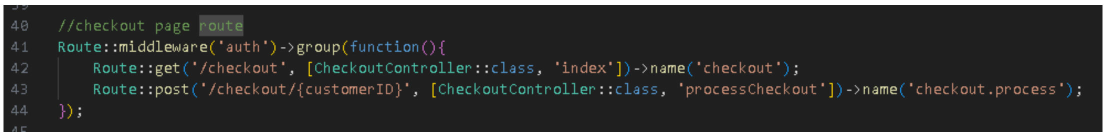
six routers each responsible for a specific functionality like displaying cart page, adding/ deleting items to the cart for authenticated customers, applying coupon, adding/ deleting items to cart for guest users.



```php
//cart page route
Route::get('/cart', [CartController::class, 'index'])->name('cart');
Route::post('/cart/add/{productID}', [CartController::class, 'addToCart'])->name('cart.add');
Route::delete('/cart/delete/{productID}', [CartController::class, 'removeItem'])->name('cart.delete');
Route::post('/cart/apply-coupon', [CartController::class, 'applyCoupon'])->name('cart.apply.coupon');
Route::delete('/cart/delete-guest/{productID}', [CartController::class, 'removeItemGuest'])->name('cart.delete.guest');
Route::post('/cart/update-quantity', [CartController::class, 'updateQuantity'])->name('cart.update.quantity');
```

1.8.4 Checkout page routes:

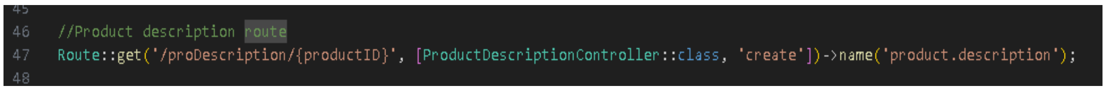
two routes get and post, one responsible for displaying the checkout page with its form and the other to submit the information filled in the form. Both wrapped in a middleware route to check if the user is logged in or not before loading checkout page.



```php
//checkout page route
Route::middleware('auth')->group(function(){
    Route::get('/checkout', [CheckoutController::class, 'index'])->name('checkout');
    Route::post('/checkout/{customerID}', [CheckoutController::class, 'processCheckout'])->name('checkout.process');
});
```

1.8.5 Product description page:

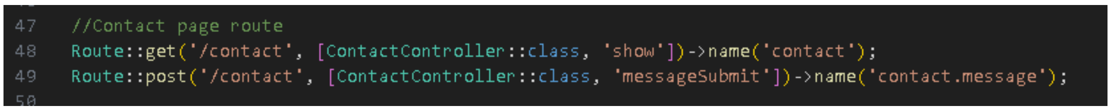
responsible for displaying product information fetched through method create.



```php
//Product description route
Route::get('/proDescription/{productID}', [ProductDescriptionController::class, 'create'])->name('product.description');
```

1.8.6 Contact page routes:

two routes one get route for displaying contact page and one post route for submitting info filled in the contact message form.

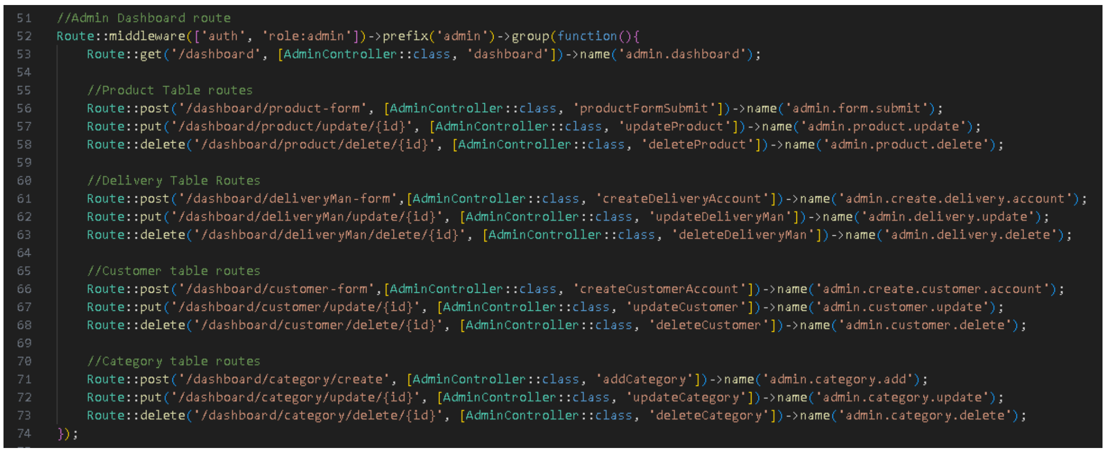


```php
//Contact page route
Route::get('/contact', [ContactController::class, 'show'])->name('contact');
Route::post('/contact', [ContactController::class, 'messageSubmit'])->name('contact.message');
```

1.8.7 Admin dashboard routes:

13 routes each has a specific functionality:

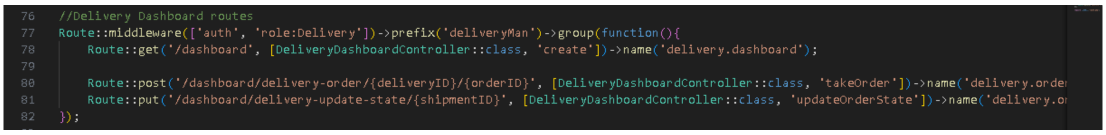
- One route for displaying dashboard for admin
- Three routes for product table in admin dashboard, post, put and delete to submit form info, update product info and delete any product
- Three routes for delivery table post, put , and delete to create an account for delivery man, update his info and delete delivery man record
- Three routes for customer table post, put and delete to create account for users, updating account info and delete an account
- Three routes for category table post, put and delete to add a category, update category type and delete a category.



```php
//Admin Dashboard route
Route::middleware(['auth', 'role:admin'])->prefix('admin')->group(function(){
    Route::get('/dashboard', [AdminController::class, 'dashboard'])->name('admin.dashboard');

    //Product Table routes
    Route::post('/dashboard/product-form', [AdminController::class, 'productFormSubmit'])->name('admin.form.submit');
    Route::put('/dashboard/product/update/{id}', [AdminController::class, 'updateProduct'])->name('admin.product.update');
    Route::delete('/dashboard/product/delete/{id}', [AdminController::class, 'deleteProduct'])->name('admin.product.delete');

    //Delivery Table Routes
    Route::post('/dashboard/deliveryMan-form',[AdminController::class, 'createDeliveryAccount'])->name('admin.create.delivery.account');
    Route::put('/dashboard/deliveryMan/update/{id}', [AdminController::class, 'updateDeliveryMan'])->name('admin.delivery.update');
    Route::delete('/dashboard/deliveryMan/delete/{id}', [AdminController::class, 'deleteDeliveryMan'])->name('admin.delivery.delete');

    //Customer table routes
    Route::post('/dashboard/customer-form',[AdminController::class, 'createCustomerAccount'])->name('admin.create.customer.account');
    Route::put('/dashboard/customer/update/{id}', [AdminController::class, 'updateCustomer'])->name('admin.customer.update');
    Route::delete('/dashboard/customer/delete/{id}', [AdminController::class, 'deleteCustomer'])->name('admin.customer.delete');

    //Category table routes
    Route::post('/dashboard/category/create', [AdminController::class, 'addCategory'])->name('admin.category.add');
    Route::put('/dashboard/category/update/{id}', [AdminController::class, 'updateCategory'])->name('admin.category.update');
    Route::delete('/dashboard/category/delete/{id}', [AdminController::class, 'deleteCategory'])->name('admin.category.delete');
});
```

1.8.8 Delivery dashboard routes:

Three routes get, post and put for displaying delivery dashboard and fetch all data needed, submit info for take order and updating the order state

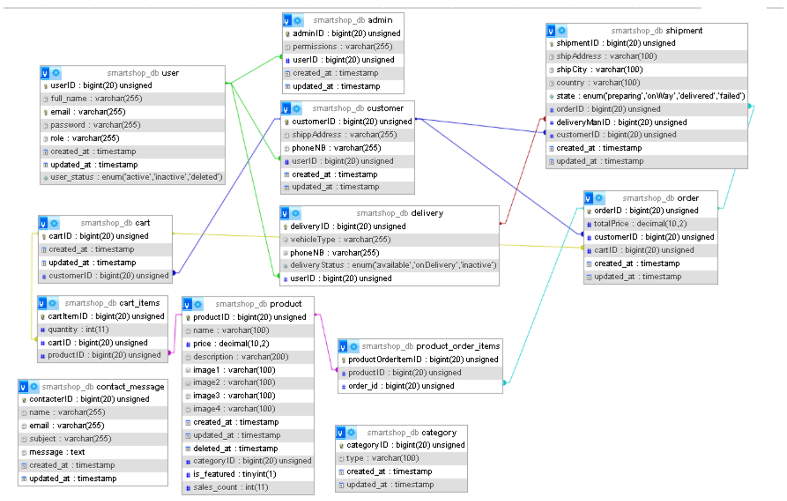


```
//Delivery Dashboard routes
Route::middleware(['auth', 'role:Delivery'])->prefix('deliveryMan')->group(function(){
    Route::get('/dashboard', [DeliveryDashboardController::class, 'create'])->name('delivery.dashboard');

    Route::post('/dashboard/delivery-order/{deliveryID}/{orderID}', [DeliveryDashboardController::class, 'takeOrder'])->name('delivery.order
    Route::put('/dashboard/delivery-update-state/{shipmentID}', [DeliveryDashboardController::class, 'updateOrderState'])->name('delivery.or
});
```

1.9 Database:

Used by MySQL to create the projects database and laravel migrations to define and manage tables.

1.9.1 Er diagram:



smartshop_db admin
adminID : bigint(20) unsigned
permissions : varchar(255)
userID : bigint(20) unsigned
created_at : timestamp
updated_at : timestamp
smartshop_db shipment
shipmentID : bigint(20) unsigned
shipAddress : varchar(100)
shipCity : varchar(100)
country : varchar(100)
state : enum('preparing','onWay','delivered','failed')
orderID : bigint(20) unsigned
deliveryManID : bigint(20) unsigned
customerID : bigint(20) unsigned
created_at : timestamp
updated_at : timestamp
smartshop_db user
userID : bigint(20) unsigned
full_name : varchar(255)
email : varchar(255)
password : varchar(255)
role : varchar(255)
created_at : timestamp
updated_at : timestamp
user_status : enum('active','inactive','deleted')
smartshop_db customer
customerID : bigint(20) unsigned
shippAddress : varchar(255)
phoneNB : varchar(255)
userID : bigint(20) unsigned
created_at : timestamp
updated_at : timestamp
smartshop_db order
orderID : bigint(20) unsigned
totalPrice : decimal(10,2)
customerID : bigint(20) unsigned
cartID : bigint(20) unsigned
created_at : timestamp
updated_at : timestamp
smartshop_db cart
cartID : bigint(20) unsigned
created_at : timestamp
updated_at : timestamp
customerID : bigint(20) unsigned
smartshop_db delivery
deliveryID : bigint(20) unsigned
vehicleType : varchar(255)
phoneNB : varchar(255)
deliveryStatus : enum('available','onDelivery','inactive')
userID : bigint(20) unsigned
smartshop_db cart_items
cartItemID : bigint(20) unsigned
quantity : int(11)
cartID : bigint(20) unsigned
productID : bigint(20) unsigned
smartshop_db product
productID : bigint(20) unsigned
name : varchar(100)
price : decimal(10,2)
description : varchar(200)
image1 : varchar(100)
image2 : varchar(100)
image3 : varchar(100)
image4 : varchar(100)
created_at : timestamp
updated_at : timestamp
deleted_at : timestamp
categoryID : bigint(20) unsigned
is_featured : tinyint(1)
sales_count : int(11)
smartshop_db product_order_items
productOrderItemID : bigint(20) unsigned
productID : bigint(20) unsigned
order_id : bigint(20) unsigned
smartshop_db contact_message
contacterID : bigint(20) unsigned
name : varchar(255)
email : varchar(255)
subject : varchar(255)
message : text
created_at : timestamp
updated_at : timestamp
smartshop_db category
categoryID : bigint(20) unsigned
type : varchar(100)
created_at : timestamp
updated_at : timestamp

1.9.2 Creating smartshop_db database in phpMyAdmin:

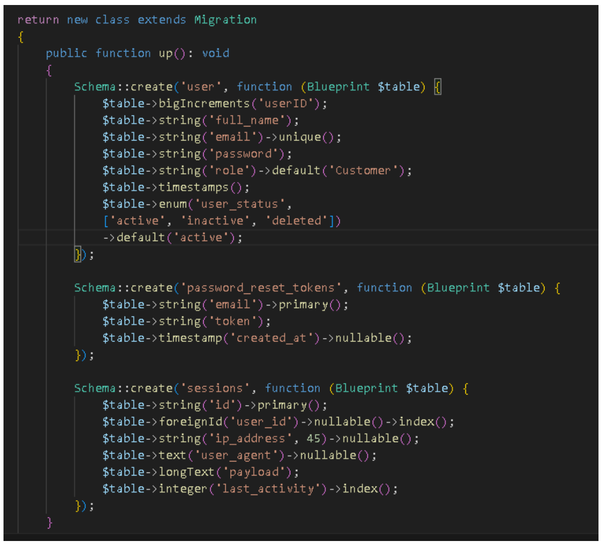
i. Open localhost/phpMyAdmin
ii. Create a new database **smartshop_db**
iii. Generate application key to use for various security-related tasks and store it in .env file as APP_KEY**:** `php artisan key:generate`

1.10 Generating tables in smartshop_db database:

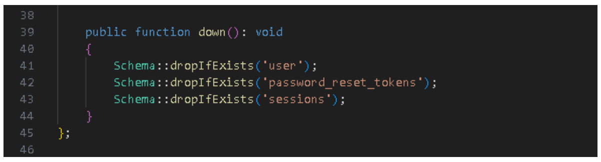
1.10.1 User table:

Creating userID, full_name, email, password, role, created_at, updated_at and user_status fields needed for all users

0001_01_01_000000_create_user_table.php:



```php
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('user', function (Blueprint $table) {
            $table->bigIncrements('userID');
            $table->string('full_name');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('role')->default('Customer');
            $table->timestamps();
            $table->enum('user_status',
            ['active', 'inactive', 'deleted'])
            ->default('active');
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }
```

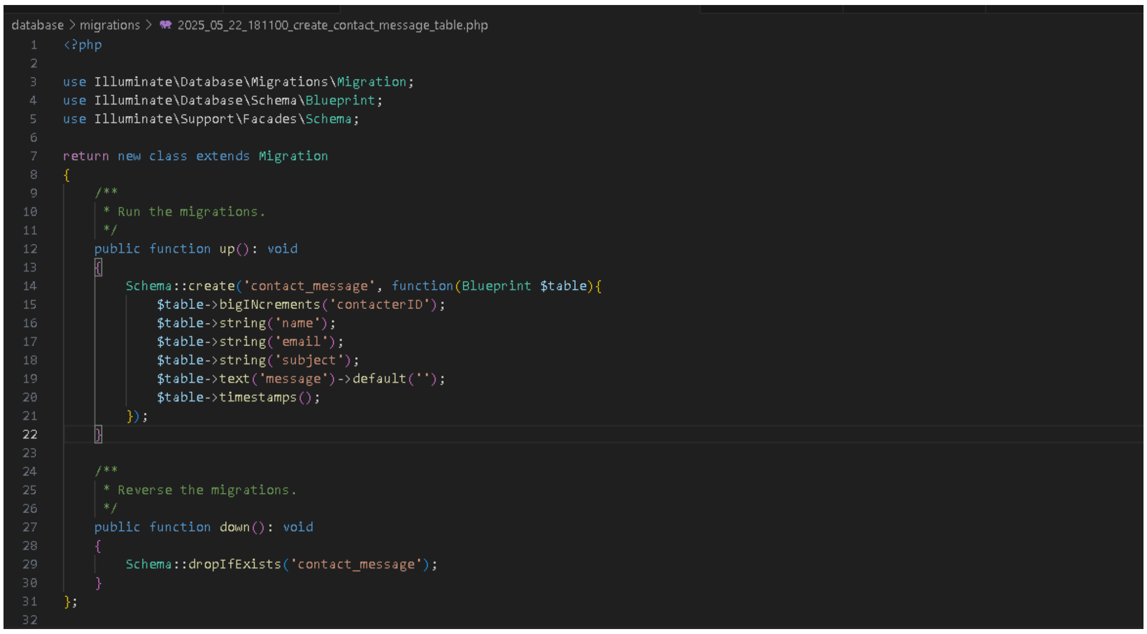


```php

    public function down(): void
    {
        Schema::dropIfExists('user');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};

```

1.10.2 Contact message table:

Creating fileds contacterID, name, email, subject, message and createa_at, updated_at using timestamp()

2025_05_22_181100_create_contact_message_table.php



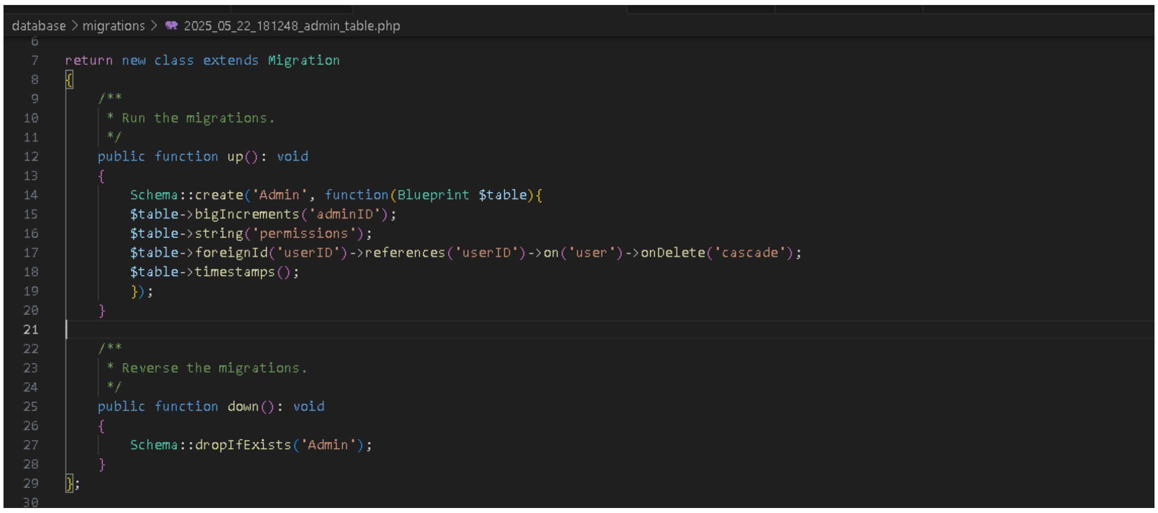
database > migrations > 2025_05_22_181100_create_contact_message_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('contact_message', function(Blueprint $table){
            $table->bigINcrements('contacterID');
            $table->string('name');
            $table->string('email');
            $table->string('subject');
            $table->text('message')->default('');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('contact_message');
    }
};

```

1.10.3 Admin table:

Created necessary fileds for an admin such as adminID, permissions, foreign key userID with user table, created_at and updated_at timestamps

2025_05_22_181248_admin_table.php

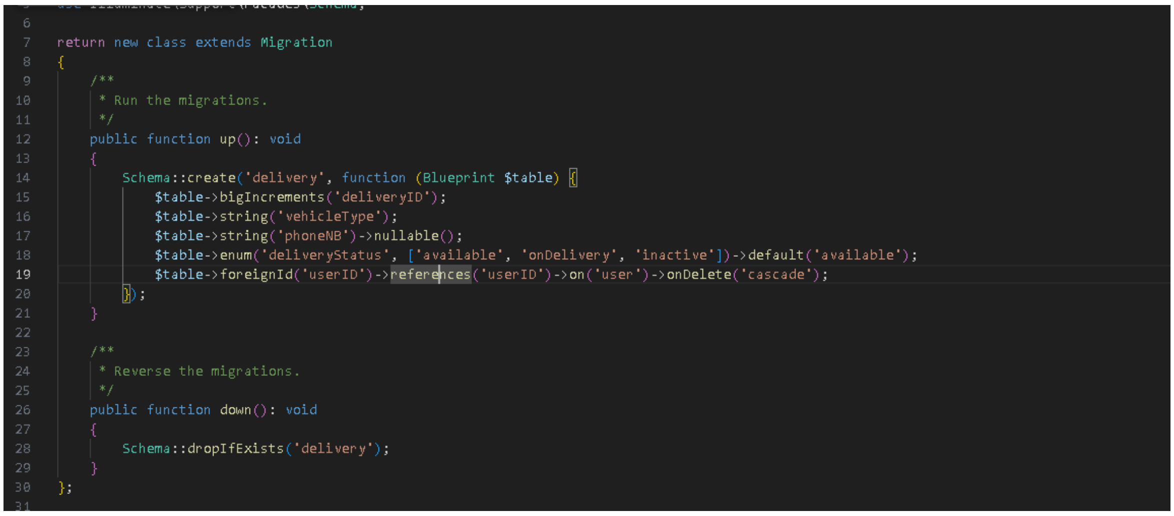


```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('Admin', function(Blueprint $table){
        $table->bigIncrements('adminID');
        $table->string('permissions');
        $table->foreignId('userID')->references('userID')->on('user')->onDelete('cascade');
        $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('Admin');
    }
};
```

1.10.4 Delivery table:

Created fields for delivery men like deliveryID, vehicleType, phoneNB, deliveryStatus and foreign key userID with user table.

2025_05_22_181326_create_delivery_table.php

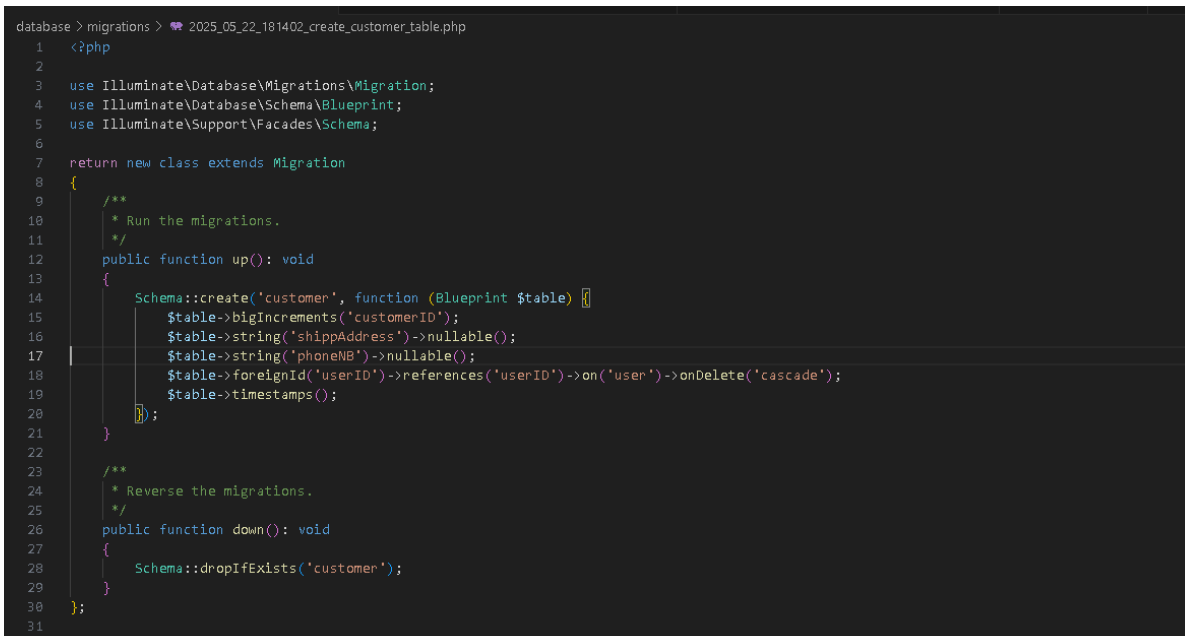


```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('delivery', function (Blueprint $table) {
            $table->bigIncrements('deliveryID');
            $table->string('vehicleType');
            $table->string('phoneNB')->nullable();
            $table->enum('deliveryStatus', ['available', 'onDelivery', 'inactive'])->default('available');
            $table->foreignId('userID')->references('userID')->on('user')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('delivery');
    }
};
```

1.10.5 Customer table:

Created fields for delivery men like customerID, shipAddress, phoneNB, foreign key userID with user table and created_at and updated_at timestamps.

2025_05_22_181402_create_customer_table.php



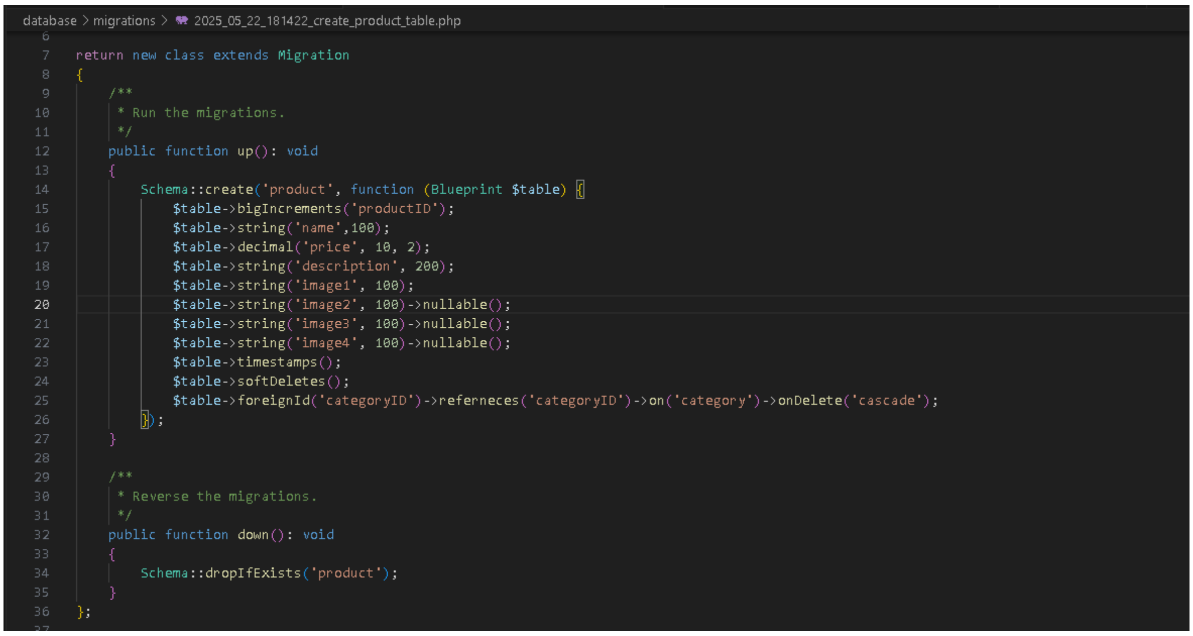
database > migrations > 2025_05_22_181402_create_customer_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('customer', function (Blueprint $table) {
            $table->bigIncrements('customerID');
            $table->string('shippAddress')->nullable();
            $table->string('phoneNB')->nullable();
            $table->foreignId('userID')->references('userID')->on('user')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('customer');
    }
};
```

1.10.6 Product table:

Creates fields for products: productid, name, description, image 1/2/3/4, timestamps ,foreign key categoryID with category table, is_featured for featured products and sales_count for products.

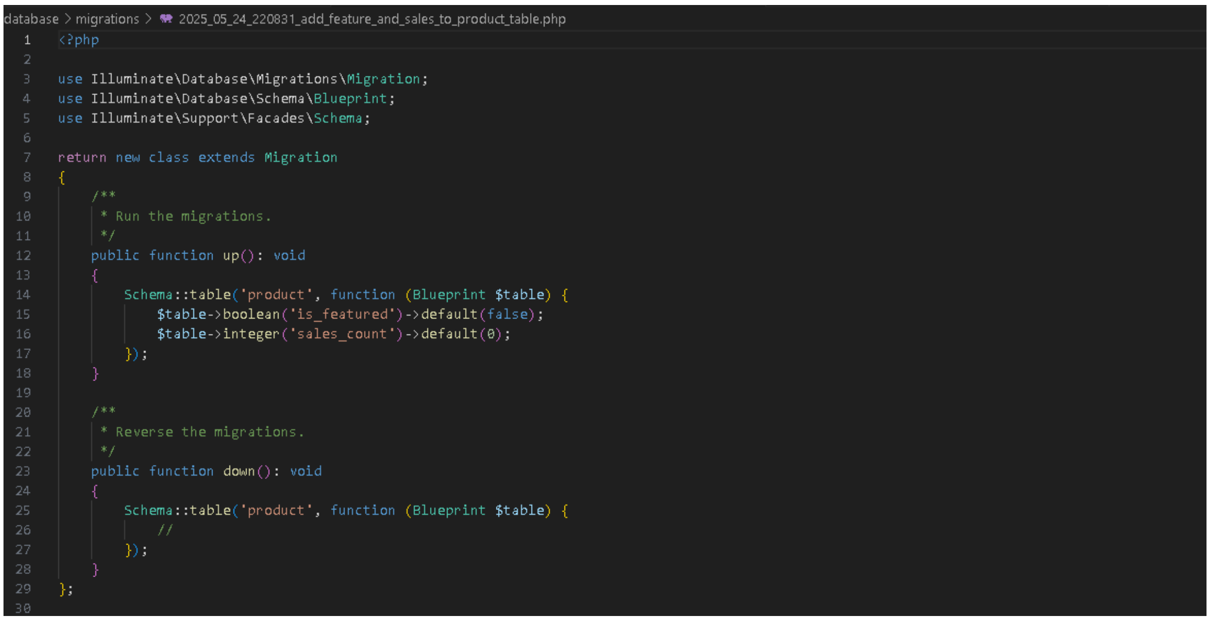
2025_05_22_181422_create_product_table.php



database > migrations > 2025_05_22_181422_create_product_table.php

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('product', function (Blueprint $table) {
            $table->bigIncrements('productID');
            $table->string('name',100);
            $table->decimal('price', 10, 2);
            $table->string('description', 200);
            $table->string('image1', 100);
            $table->string('image2', 100)->nullable();
            $table->string('image3', 100)->nullable();
            $table->string('image4', 100)->nullable();
            $table->timestamps();
            $table->softDeletes();
            $table->foreignId('categoryID')->referneces('categoryID')->on('category')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('product');
    }
};
```

2025_05_24_220831_add_feature_and_sales_to_product_table.php



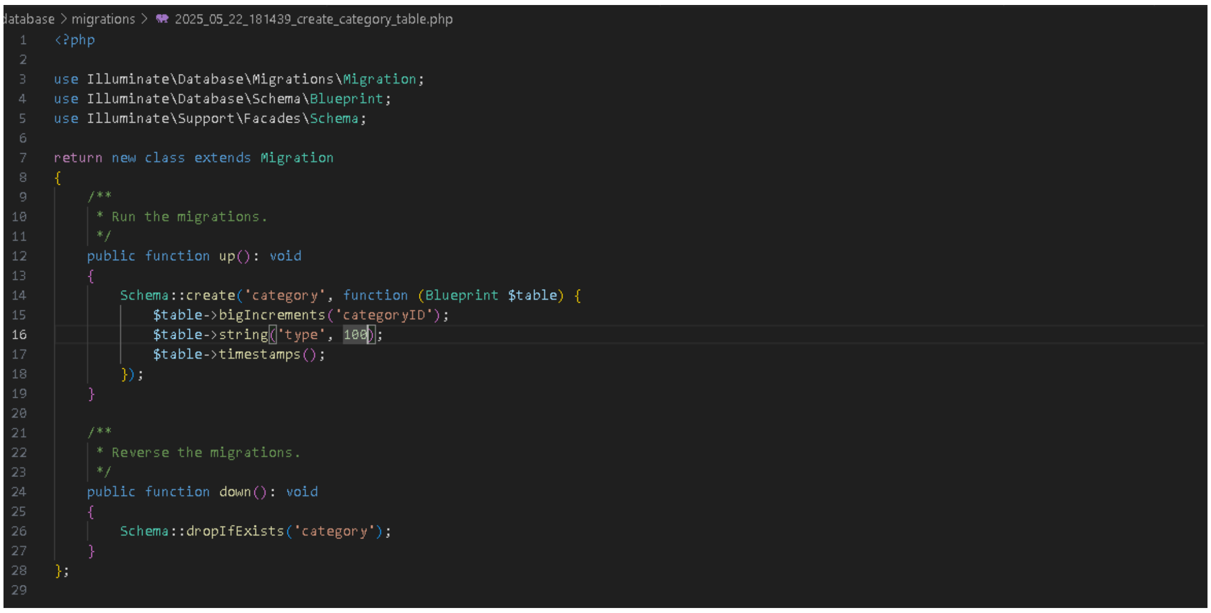
database > migrations > 2025_05_24_220831_add_feature_and_sales_to_product_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('product', function (Blueprint $table) {
            $table->boolean('is_featured')->default(false);
            $table->integer('sales_count')->default(0);
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('product', function (Blueprint $table) {
            //
        });
    }
};
```

1.10.7 Category table:

Creates fileds for product category has categoryID, type and timestamps

2025_05_22_181439_create_category_table.php



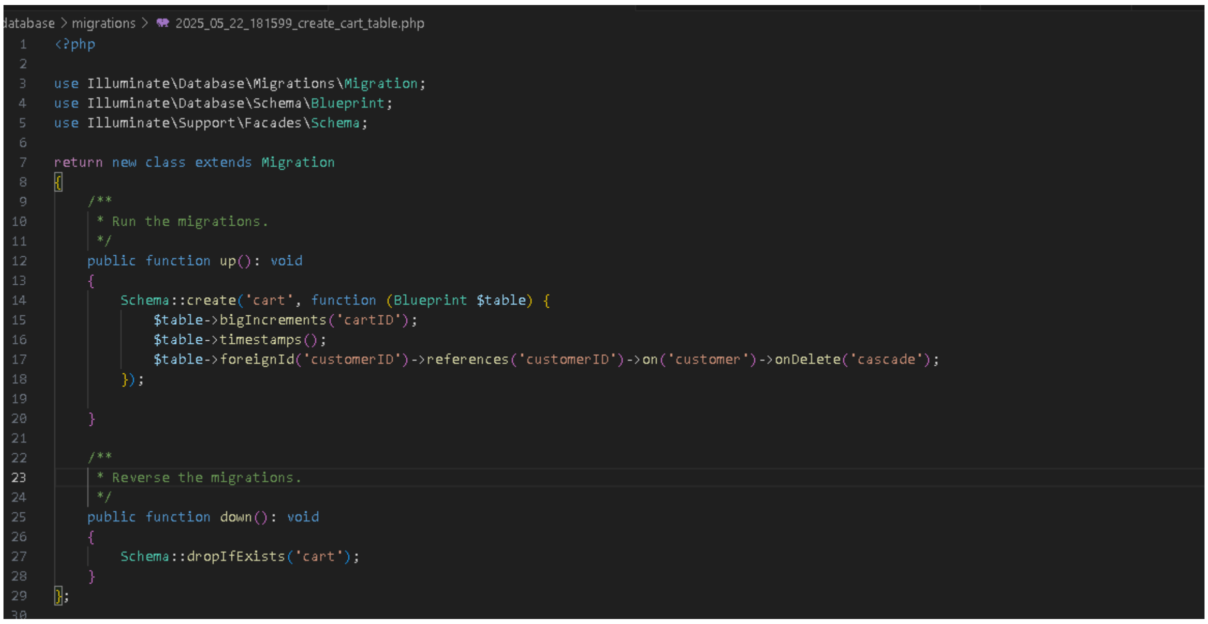
database > migrations > 2025_05_22_181439_create_category_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('category', function (Blueprint $table) {
            $table->bigIncrements('categoryID');
            $table->string('type', 100);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('category');
    }
};
```

1.10.8 Cart table:

Creates filed for products added to the cart like cartID, timestamps and foreign key customerID with customer table

2025_05_22_181599_create_cart_table.php

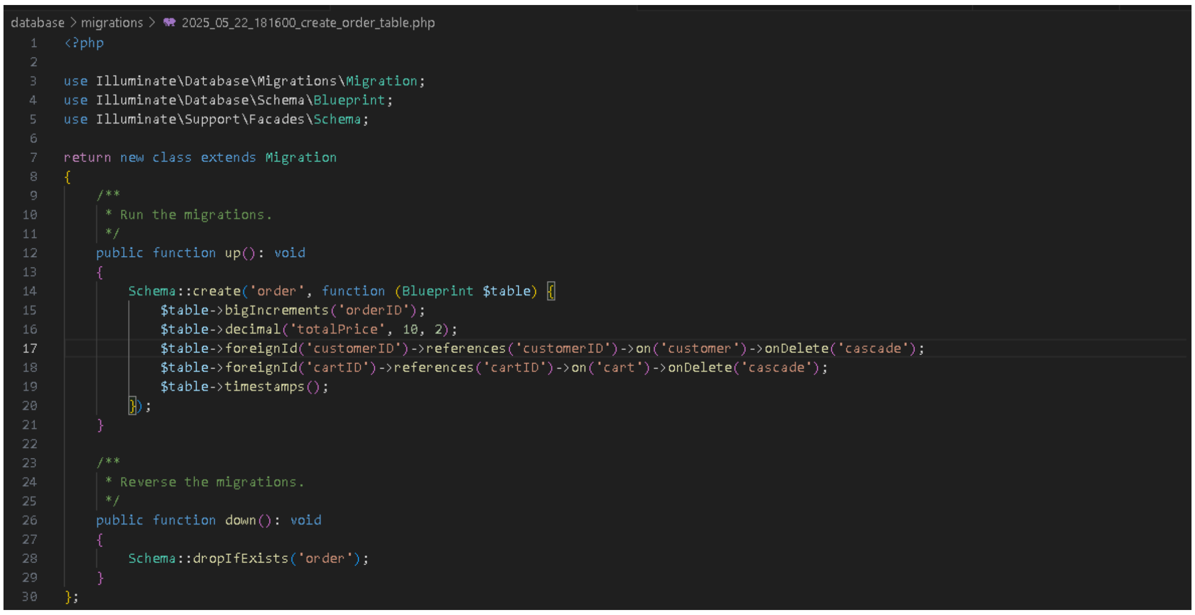


```
database > migrations > 2025_05_22_181599_create_cart_table.php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('cart', function (Blueprint $table) {
            $table->bigIncrements('cartID');
            $table->timestamps();
            $table->foreignId('customerID')->references('customerID')->on('customer')->onDelete('cascade');
        });

    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('cart');
    }
};
```

1.10.9 Order table:

Creates fields for customer orders like orderID, total price of the order, foreign key customerID with customer table and cartID with cart table

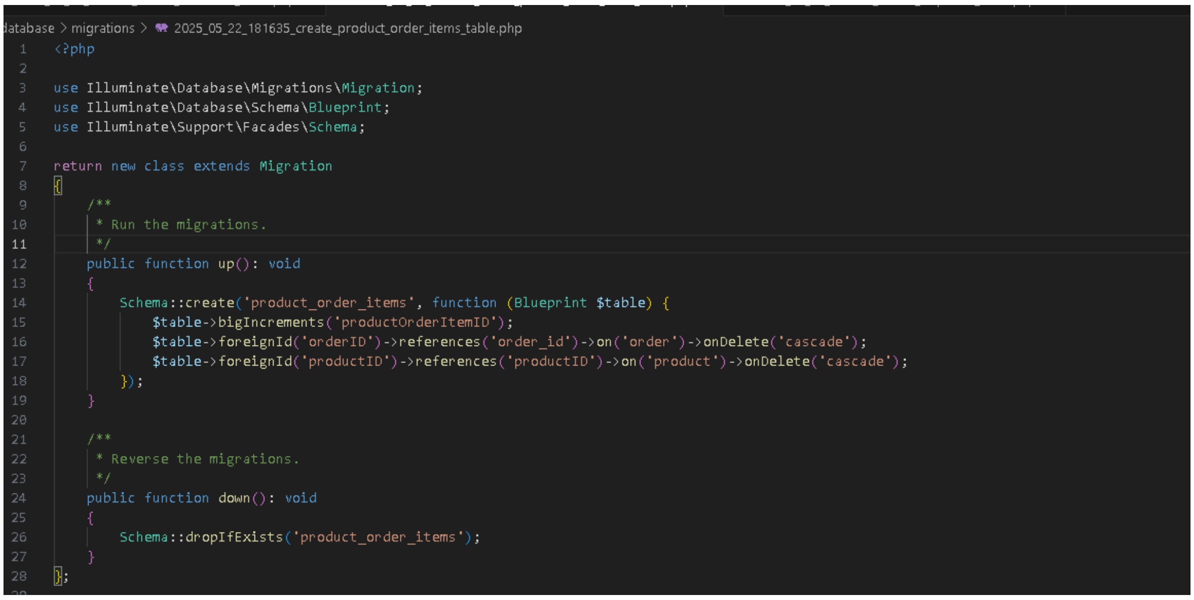
2025_05_22_181600_create_order_table.php



```
database > migrations > 2025_05_22_181600_create_order_table.php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('order', function (Blueprint $table) {
            $table->bigIncrements('orderID');
            $table->decimal('totalPrice', 10, 2);
            $table->foreignId('customerID')->references('customerID')->on('customer')->onDelete('cascade');
            $table->foreignId('cartID')->references('cartID')->on('cart')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('order');
    }
};
```

1.10.10 Product Order items pivot table:

Creates all necessary fields to link order and product tables due to many-to-many relationship. (productOrderItemID, foreign keys orderID with order table and productID with product table)



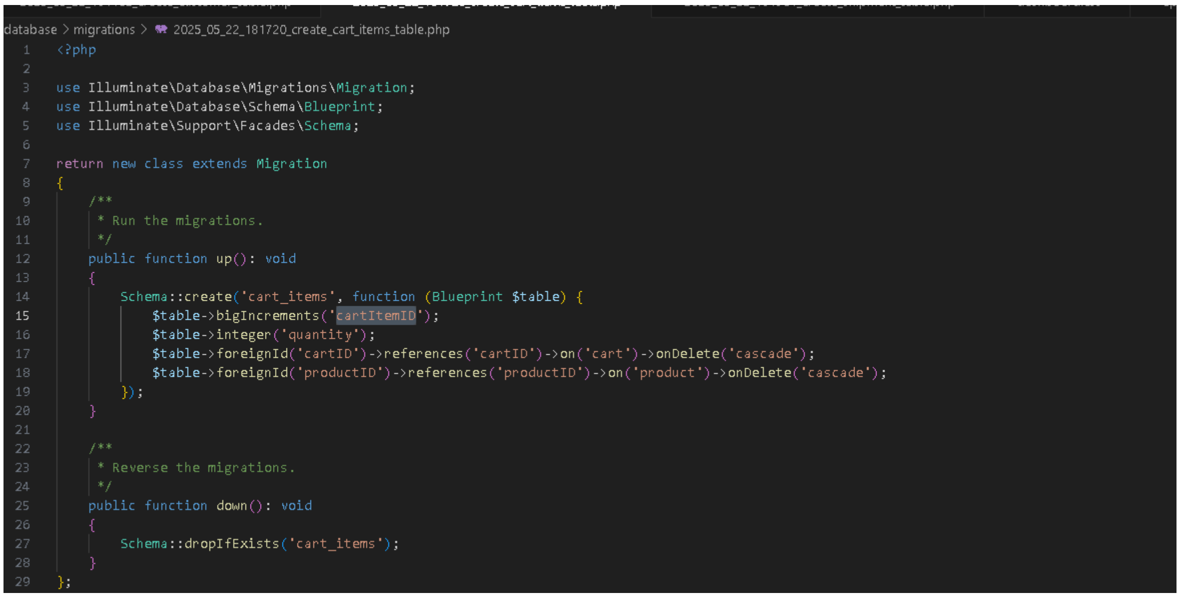
```
database > migrations > 2025_05_22_181635_create_product_order_items_table.php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('product_order_items', function (Blueprint $table) {
            $table->bigIncrements('productOrderItemID');
            $table->foreignId('orderID')->references('order_id')->on('order')->onDelete('cascade');
            $table->foreignId('productID')->references('productID')->on('product')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('product_order_items');
    }
};
```

Figure 1. 2025_05_22_181635_create_product_order_items_table.php

1.10.11 Cart items table:

Creates fields for items info -> cartItemID, quantity for how much from each item a customer wants to buy , and foreign keys cartID with cart table and productID with product table.

2025_05_22_181720_create_cart_items_table.php

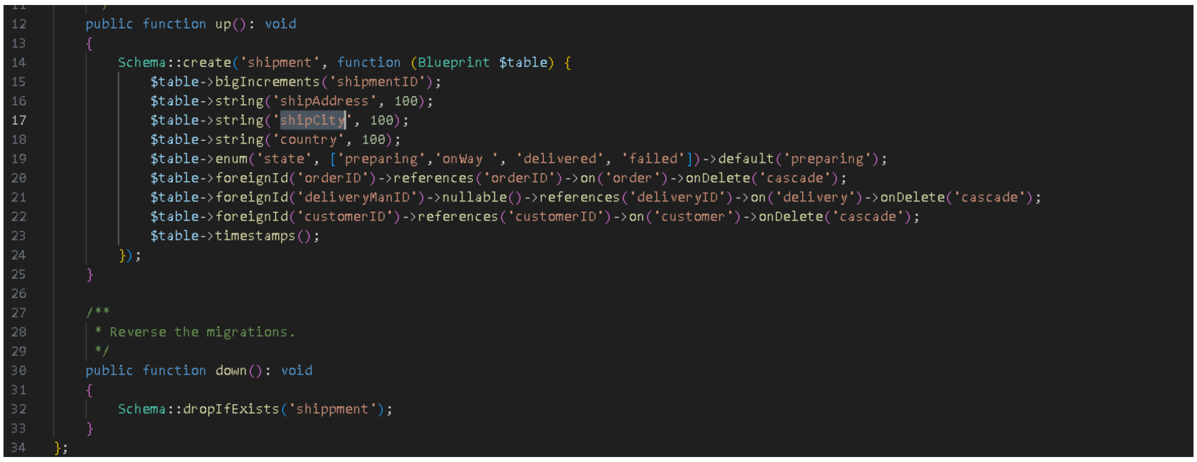


```
database > migrations > 2025_05_22_181720_create_cart_items_table.php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('cart_items', function (Blueprint $table) {
            $table->bigIncrements('cartItemID');
            $table->integer('quantity');
            $table->foreignId('cartID')->references('cartID')->on('cart')->onDelete('cascade');
            $table->foreignId('productID')->references('productID')->on('product')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('cart_items');
    }
};
```

1.10.12 Shipment table:

Creates fields for shipmen location such as shipmentID, shipAddress, shipCity, country, state for orders state, foreign keys orderID with order table, deliveryManID with delivery table and customerID with customerTable, and timestamps.

2025_05_22_181901_create_shipment_table.php



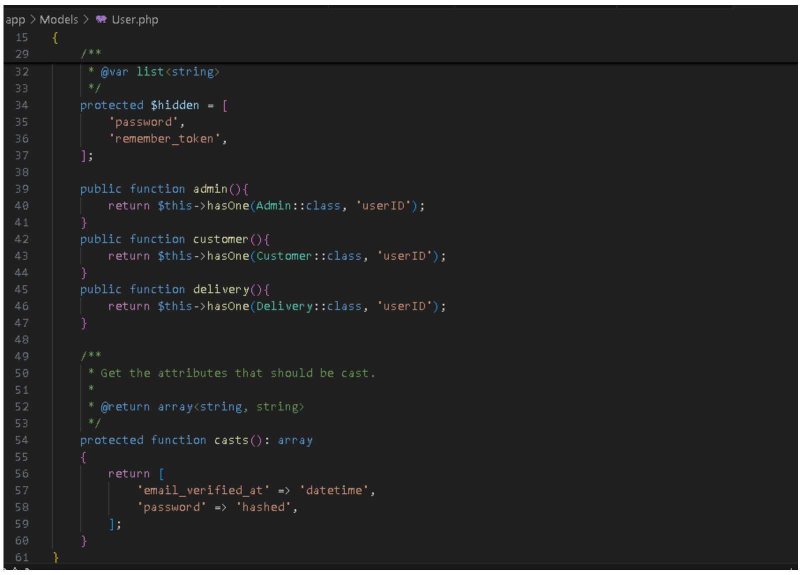
```php
    public function up(): void
    {
        Schema::create('shipment', function (Blueprint $table) {
            $table->bigIncrements('shipmentID');
            $table->string('shipAddress', 100);
            $table->string('shipCity', 100);
            $table->string('country', 100);
            $table->enum('state', ['preparing','onWay ', 'delivered', 'failed'])->default('preparing');
            $table->foreignId('orderID')->references('orderID')->on('order')->onDelete('cascade');
            $table->foreignId('deliveryManID')->nullable()->references('deliveryID')->on('delivery')->onDelete('cascade');
            $table->foreignId('customerID')->references('customerID')->on('customer')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('shippment');
    }
};
```

1.11 Models:

Created Eloquent models for each database table to represent the data and define relationships

1.11.1 User.php:

Each user has one admin or customer or delivery:

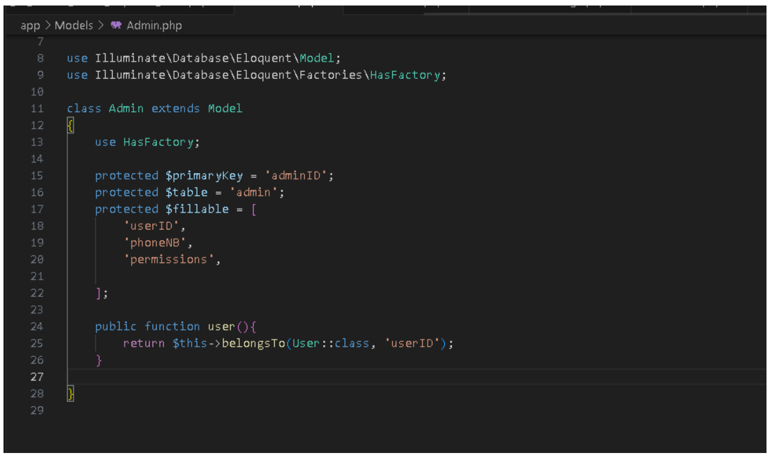


app > Models > User.php

```php
{
    /**
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    public function admin(){
        return $this->hasOne(Admin::class, 'userID');
    }
    public function customer(){
        return $this->hasOne(Customer::class, 'userID');
    }
    public function delivery(){
        return $this->hasOne(Delivery::class, 'userID');
    }

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
}
```

1.11.2 Admin.php:

Each admin is a user (belong to user table)



app > Models > Admin.php

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

class Admin extends Model
{
    use HasFactory;

    protected $primaryKey = 'adminID';
    protected $table = 'admin';
    protected $fillable = [
        'userID',
        'phoneNB',
        'permissions',

    ];

    public function user(){
        return $this->belongsTo(User::class, 'userID');
    }

}
```

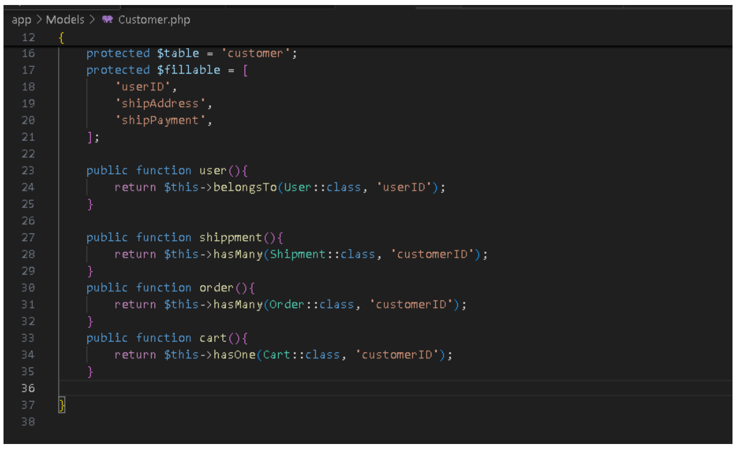
1.11.3 Customer.php:

Each customer is a user (belong to user table)

Each customer can have many shipment orders

Each customer can order many orders

Each customer can have one cart



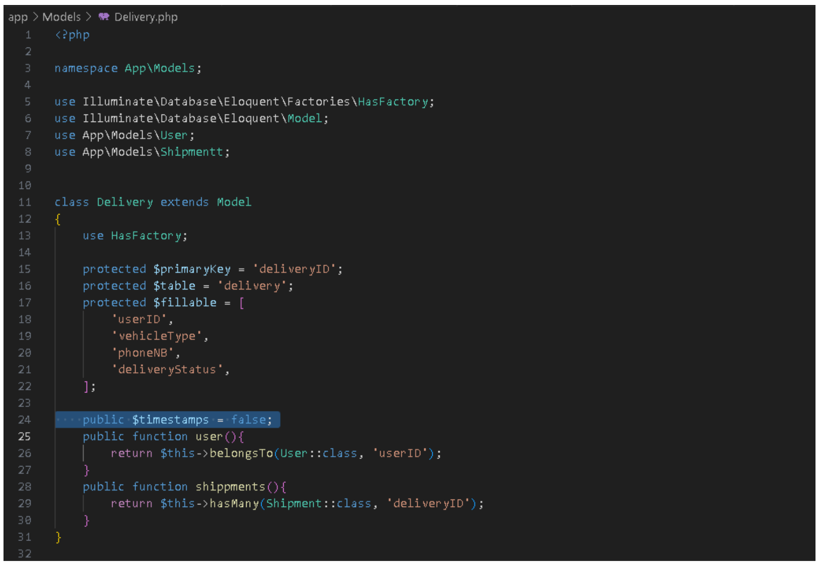
app > Models > Customer.php

```php
{
    protected $table = 'customer';
    protected $fillable = [
        'userID',
        'shipAddress',
        'shipPayment',
    ];

    public function user(){
        return $this->belongsTo(User::class, 'userID');
    }

    public function shippment(){
        return $this->hasMany(Shipment::class, 'customerID');
    }
    public function order(){
        return $this->hasMany(Order::class, 'customerID');
    }
    public function cart(){
        return $this->hasOne(Cart::class, 'customerID');
    }

}
```

1.11.4 Delivery.php:

Each delivery is a user(belong to user table)

Each delivery man has many shipment orders to take


app > Models > Delivery.php

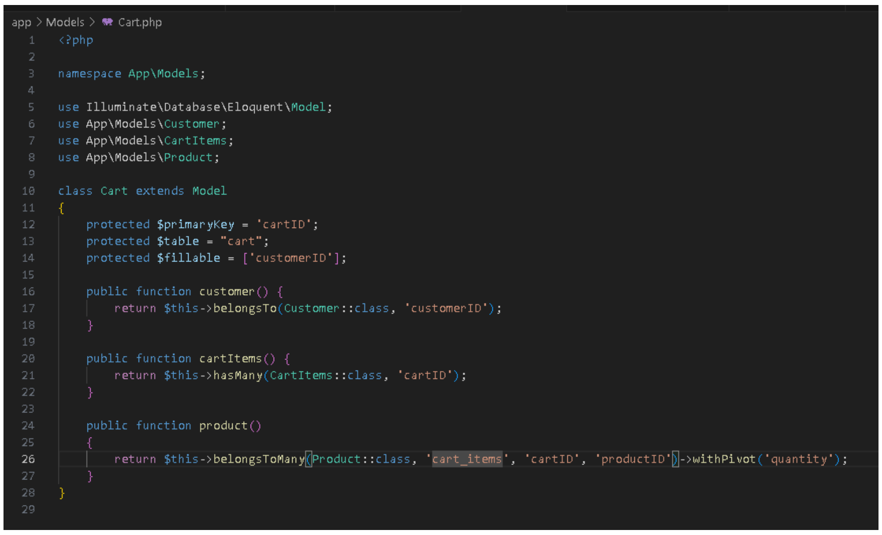
```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\User;
use App\Models\Shipmentt;


class Delivery extends Model
{
    use HasFactory;

    protected $primaryKey = 'deliveryID';
    protected $table = 'delivery';
    protected $fillable = [
        'userID',
        'vehicleType',
        'phoneNB',
        'deliveryStatus',
    ];

    public $timestamps = false;
    public function user(){
        return $this->belongsTo(User::class, 'userID');
    }
    public function shippments(){
        return $this->hasMany(Shipment::class, 'deliveryID');
    }
}
```

1.11.5 Cart.php:

Each cat belongs to a unique customer

Each cart can have many items in it

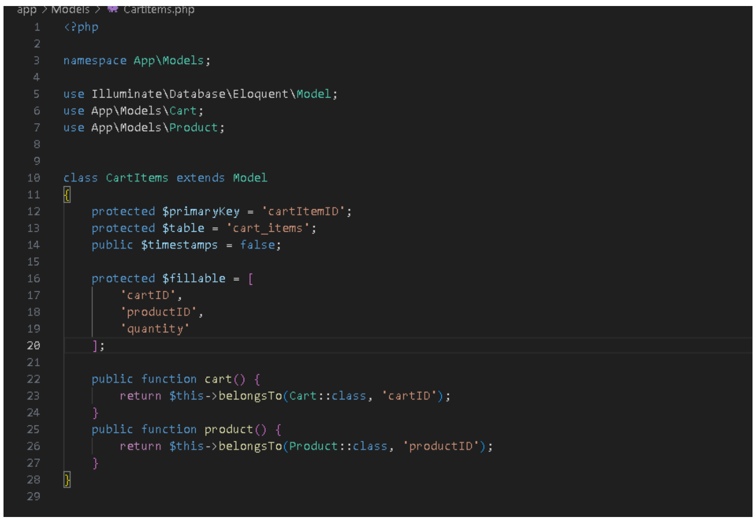
Each cart belongs to many products


app > Models > Cart.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use App\Models\Customer;
use App\Models\CartItems;
use App\Models\Product;

class Cart extends Model
{
    protected $primaryKey = 'cartID';
    protected $table = "cart";
    protected $fillable = ['customerID'];

    public function customer() {
        return $this->belongsTo(Customer::class, 'customerID');
    }

    public function cartItems() {
        return $this->hasMany(CartItems::class, 'cartID');
    }

    public function product()
    {
        return $this->belongsToMany(Product::class, 'cart_items', 'cartID', 'productID')->withPivot('quantity');
    }
}
```

1.11.6 CartItems.php:

Pivot table many-to-many relationship between cart and product table:

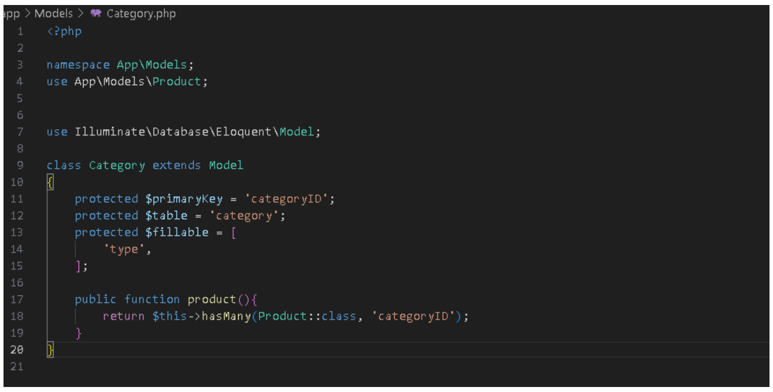


app > Models > CartItems.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use App\Models\Cart;
use App\Models\Product;


class CartItems extends Model
{
    protected $primaryKey = 'cartItemID';
    protected $table = 'cart_items';
    public $timestamps = false;

    protected $fillable = [
        'cartID',
        'productID',
        'quantity'
    ];

    public function cart() {
        return $this->belongsTo(Cart::class, 'cartID');
    }
    public function product() {
        return $this->belongsTo(Product::class, 'productID');
    }
}
```

1.11.7 Category.php:

E3ach category can be assigned to many products

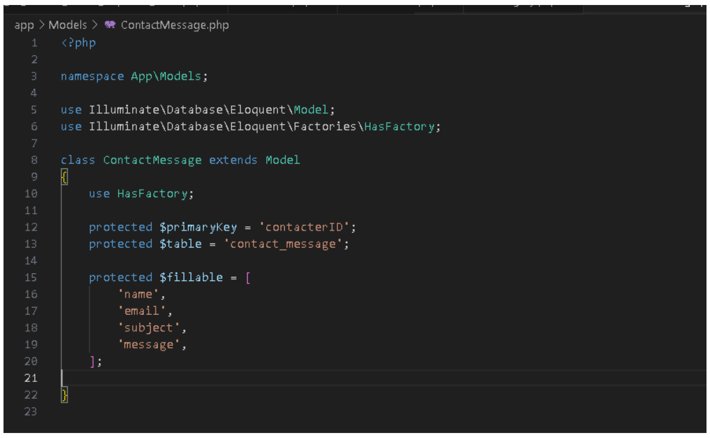


pp > Models > Category.php

```php
<?php

namespace App\Models;
use App\Models\Product;


use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    protected $primaryKey = 'categoryID';
    protected $table = 'category';
    protected $fillable = [
        'type',
    ];

    public function product(){
        return $this->hasMany(Product::class, 'categoryID');
    }
}
```

1.11.8 ContactMessage.php:

No relationship with any table for its functionality



app > Models > ContactMessage.php

```php
<?php

namespace App\Models;

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

class ContactMessage extends Model
{
    use HasFactory;

    protected $primaryKey = 'contacterID';
    protected $table = 'contact_message';

    protected $fillable = [
        'name',
        'email',
        'subject',
        'message',
    ];

}
```

1.11.9 Order.php:

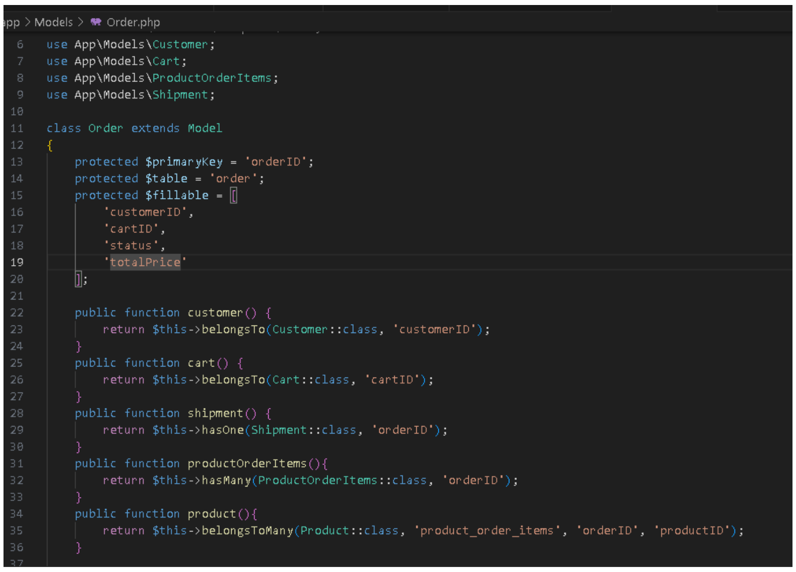
Each order belongs to one customer (belongs to customer table)

Each order purchased belongs to one cart (belong to cart table)

Each order is related to one shipment

Each order can have many order items in it (pivot table product order items)

Many orders can have many products (many-to-many relationship with products)



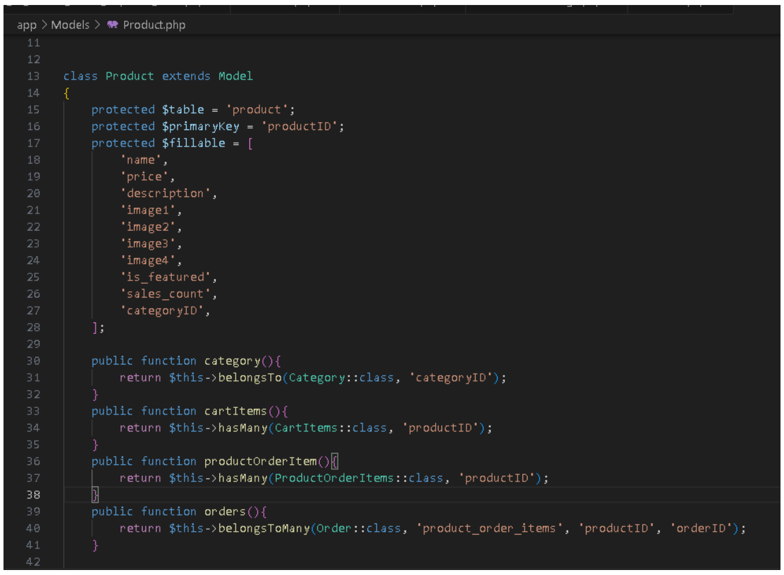
```php
use App\Models\Customer;
use App\Models\Cart;
use App\Models\ProductOrderItems;
use App\Models\Shipment;

class Order extends Model
{
    protected $primaryKey = 'orderID';
    protected $table = 'order';
    protected $fillable = [
        'customerID',
        'cartID',
        'status',
        'totalPrice'
    ];

    public function customer() {
        return $this->belongsTo(Customer::class, 'customerID');
    }
    public function cart() {
        return $this->belongsTo(Cart::class, 'cartID');
    }
    public function shipment() {
        return $this->hasOne(Shipment::class, 'orderID');
    }
    public function productOrderItems(){
        return $this->hasMany(ProductOrderItems::class, 'orderID');
    }
    public function product(){
        return $this->belongsToMany(Product::class, 'product_order_items', 'orderID', 'productID');
    }
```

1.11.10 Product.php:

Each product belongs to a category

Each product can be in many cart items

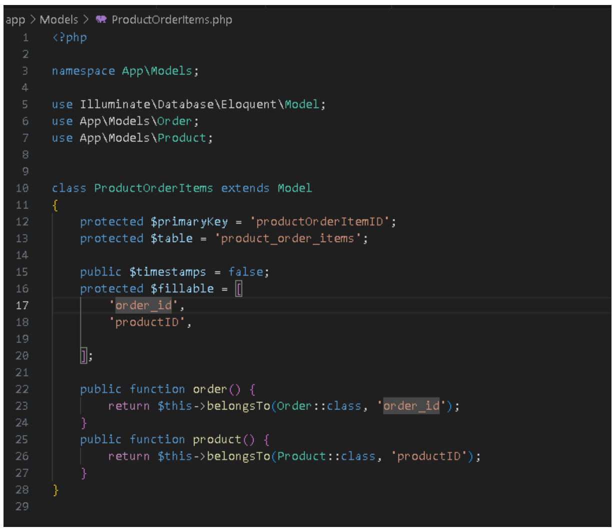
Each product belongs to many orders



```php


class Product extends Model
{
    protected $table = 'product';
    protected $primaryKey = 'productID';
    protected $fillable = [
        'name',
        'price',
        'description',
        'image1',
        'image2',
        'image3',
        'image4',
        'is_featured',
        'sales_count',
        'categoryID',
    ];

    public function category(){
        return $this->belongsTo(Category::class, 'categoryID');
    }
    public function cartItems(){
        return $this->hasMany(CartItems::class, 'productID');
    }
    public function productOrderItem(){
        return $this->hasMany(ProductOrderItems::class, 'productID');
    }
    public function orders(){
        return $this->belongsToMany(Order::class, 'product_order_items', 'productID', 'orderID');
    }

```

1.11.11 ProductOrderItems.php:

Pivot table many-to-many relationship between order and product table:



app > Models > ProductOrderItems.php

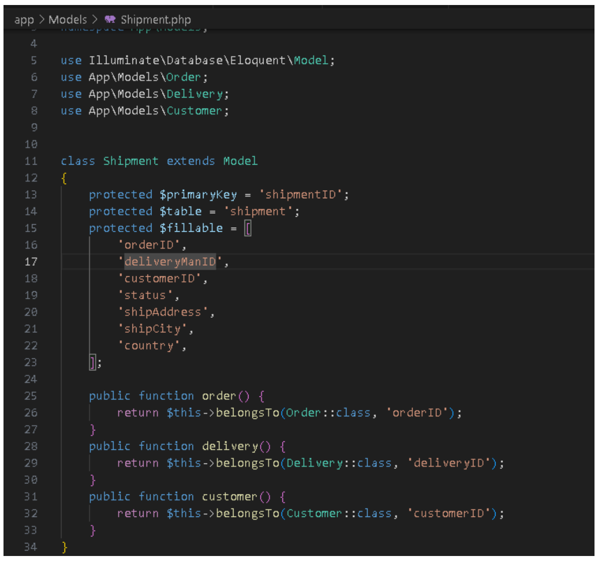
```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use App\Models\Order;
use App\Models\Product;


class ProductOrderItems extends Model
{
    protected $primaryKey = 'productOrderItemID';
    protected $table = 'product_order_items';

    public $timestamps = false;
    protected $fillable = [
        'order_id',
        'productID',

    ];

    public function order() {
        return $this->belongsTo(Order::class, 'order_id');
    }
    public function product() {
        return $this->belongsTo(Product::class, 'productID');
    }
}

```

1.11.12 Shipment.php:

Each shipment belongs to an order

Each shipment belongs to a customer

Each shipment belongs to a delivery man



app > Models > Shipment.php

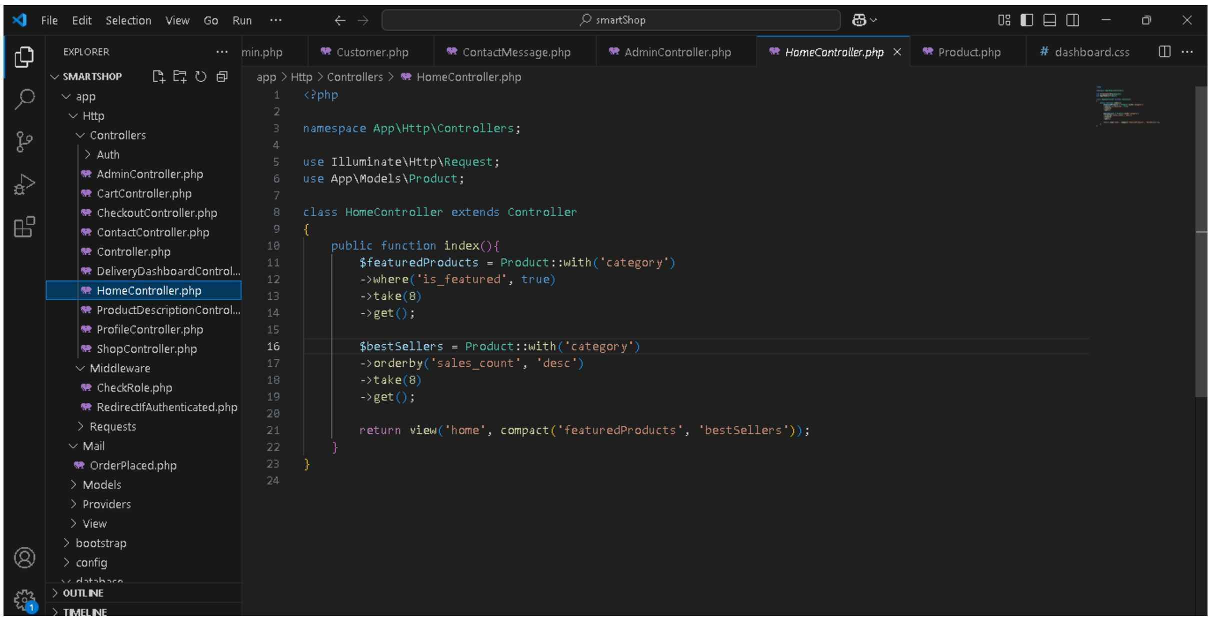
```php

use Illuminate\Database\Eloquent\Model;
use App\Models\Order;
use App\Models\Delivery;
use App\Models\Customer;


class Shipment extends Model
{
    protected $primaryKey = 'shipmentID';
    protected $table = 'shipment';
    protected $fillable = [
        'orderID',
        'deliveryManID',
        'customerID',
        'status',
        'shipAddress',
        'shipCity',
        'country',
    ];

    public function order() {
        return $this->belongsTo(Order::class, 'orderID');
    }
    public function delivery() {
        return $this->belongsTo(Delivery::class, 'deliveryID');
    }
    public function customer() {
        return $this->belongsTo(Customer::class, 'customerID');
    }
}
```

1.12 Controllers:

Built to handle application logic such as displaying, managing and handling authentications

1.12.1 Home controller:

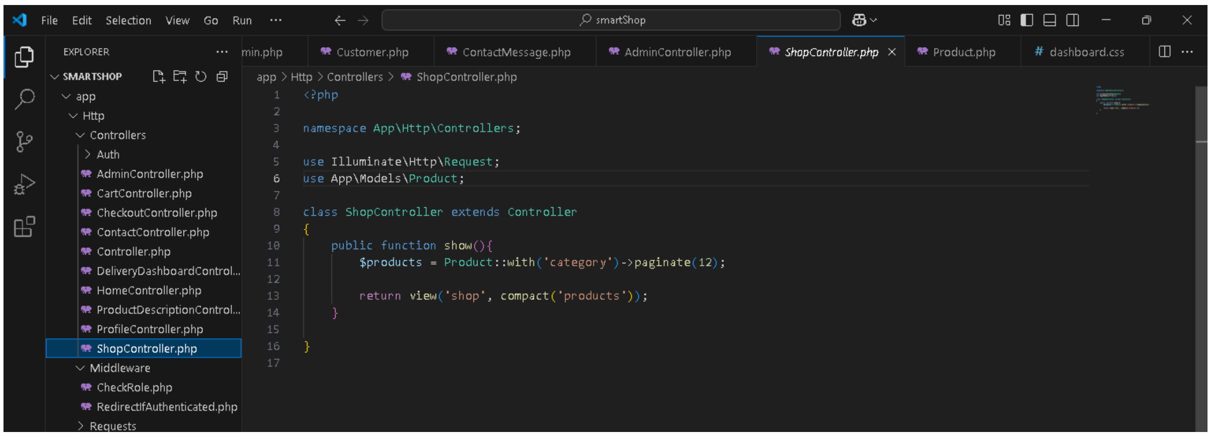
- index(): this method retrieves first 8 records of featured products and the top 8 bought products based on ordering the products by their sales counts and return home view with the passed data.



```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;

class HomeController extends Controller
{
    public function index(){
        $featuredProducts = Product::with('category')
        ->where('is_featured', true)
        ->take(8)
        ->get();

        $bestSellers = Product::with('category')
        ->orderby('sales_count', 'desc')
        ->take(8)
        ->get();

        return view('home', compact('featuredProducts', 'bestSellers'));
    }
}
```

1.12.2 Shop controller:

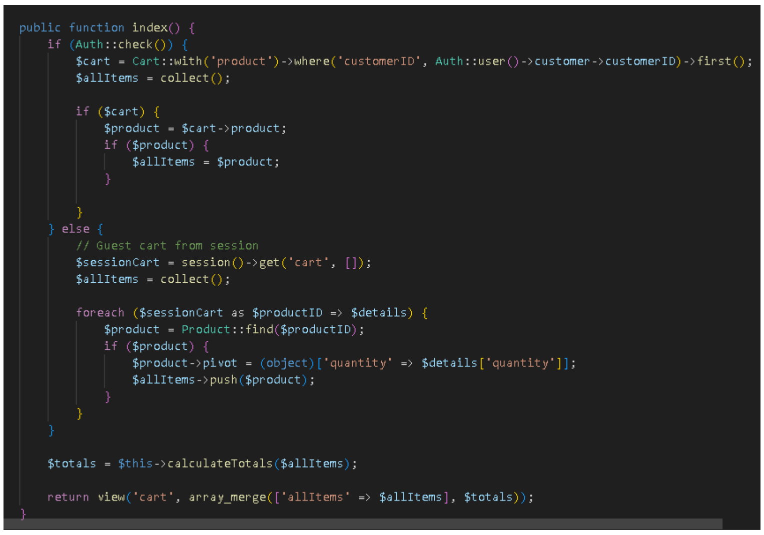
- show(): this method retrieves all products with their categories and return shop view with the retrieved data.



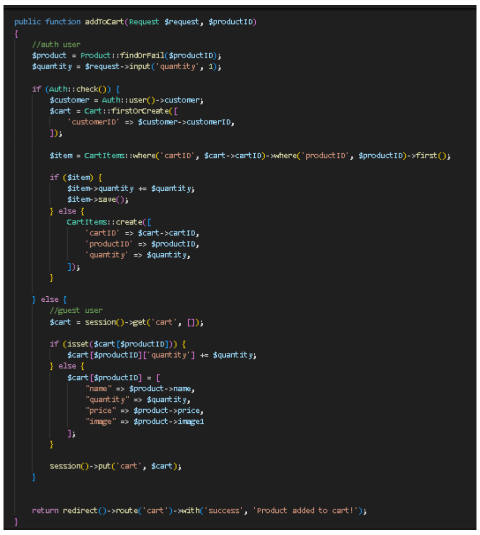
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;

class ShopController extends Controller
{
    public function show(){
        $products = Product::with('category')->paginate(12);

        return view('shop', compact('products'));
    }

}
```

1.12.3 Cart controller:

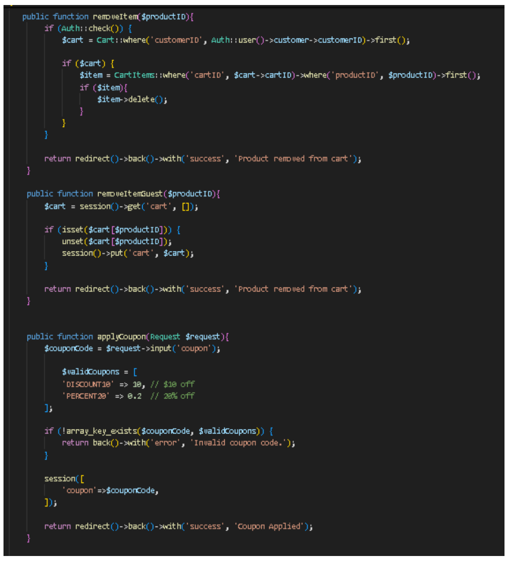
- index(): checks if the user is authenticated then it fetches his cart products and assign it in allItems collection, else if it is a guest, it creates a session and assign all products in it. Also, it gets the price of eachitems then it merges all data fetched in an array and pass them to the view
- addToCart(): finds products based on passed id in the method parameter, then checks if the user is authenticated it creates a cart for this user and checks for the quantity amount submitted for each product then creates cart items. While for the unlogged in users it creates a session under name cart, and fill in cart items with the products submitted using their ids.
- removeItem(): it removes items from the authenticated users' cart based on comparing product's id with the cart Item table
- removeItemGuest(): retrieves the sessions, compare the cart with product id and then drops the session created for each product based on their id.
- applyCoupon(): handles the request for applying two coupons DISCOUNT10 and PERCENT20, creates the array that holds coupon codes, then compare the submit one to array.
- updateQuantity(): handles updating quantity for each user (auth & guest) enter a number in the cart page.
- calculatesTotals(): this method calculates the total price for the order based on multiplying product price by quantity number, and subtracts money if coupon is applied, and sends the data calculated for view usage.



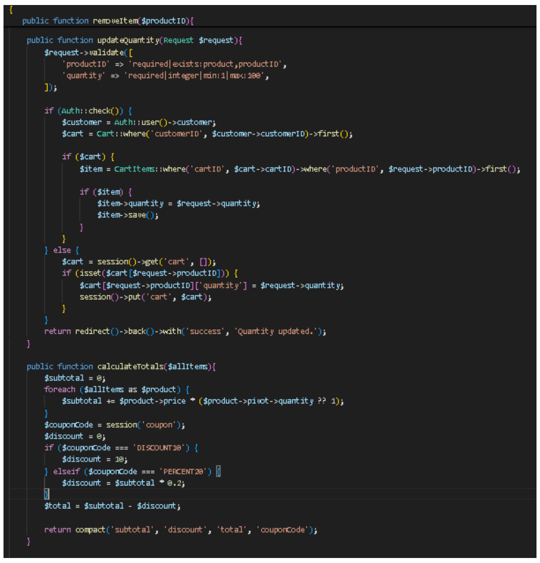
```php
public function index() {
    if (Auth::check()) {
        $cart = Cart::with('product')->where('customerID', Auth::user()->customer->customerID)->first();
        $allItems = collect();

        if ($cart) {
            $product = $cart->product;
            if ($product) {
                $allItems = $product;
            }

        }
    } else {
        // Guest cart from session
        $sessionCart = session()->get('cart', []);
        $allItems = collect();

        foreach ($sessionCart as $productID => $details) {
            $product = Product::find($productID);
            if ($product) {
                $product->pivot = (object)['quantity' => $details['quantity']];
                $allItems->push($product);
            }
        }
    }

    $totals = $this->calculateTotals($allItems);

    return view('cart', array_merge(['allItems' => $allItems], $totals));
}
```



```php
public function addToCart(Request $request, $productID)
{
    //auth user
    $product = Product::findOrFail($productID);
    $quantity = $request->input('quantity', 1);

    if (Auth::check()) {
        $customer = Auth::user()->customer;
        $cart = Cart::firstOrCreate([
            'customerID' => $customer->customerID,
        ]);

        $item = CartItems::where('cartID', $cart->cartID)->where('productID', $productID)->first();

        if ($item) {
            $item->quantity += $quantity;
            $item->save();
        } else {
            CartItems::create([
                'cartID' => $cart->cartID,
                'productID' => $productID,
                'quantity' => $quantity,
            ]);
        }

    } else {
        //guest user
        $cart = session()->get('cart', []);

        if (isset($cart[$productID])) {
            $cart[$productID]['quantity'] += $quantity;
        } else {
            $cart[$productID] = [
                "name" => $product->name,
                "quantity" => $quantity,
                "price" => $product->price,
                "image" => $product->image1
            ];
        }

        session()->put('cart', $cart);
    }


    return redirect()->route('cart')->with('success', 'Product added to cart!');
}
```



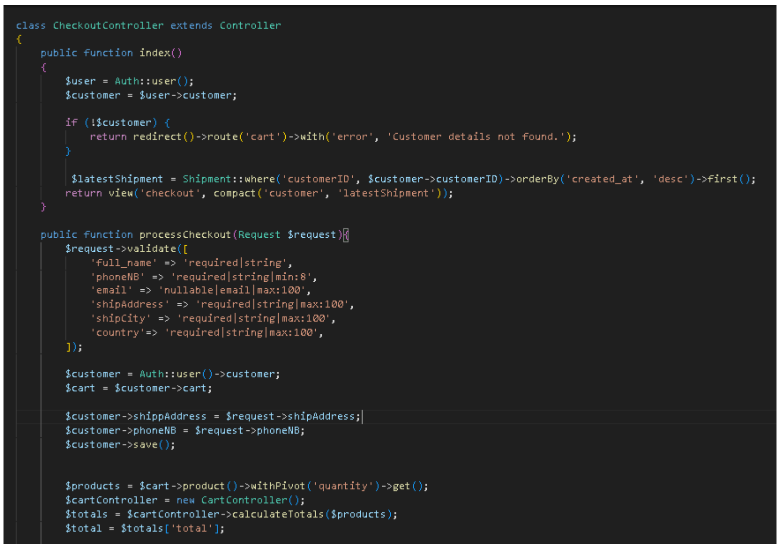
```php
    public function removeItem($productID){
        if (Auth::check()) {
            $cart = Cart::where('customerID', Auth::user()->customer->customerID)->first();

            if ($cart) {
                $item = CartItems::where('cartID', $cart->cartID)->where('productID', $productID)->first();
                if ($item){
                    $item->delete();
                }
            }
        }

        return redirect()->back()->with('success', 'Product removed from cart');
    }

    public function removeItemGuest($productID){
        $cart = session()->get('cart', []);

        if (isset($cart[$productID])) {
            unset($cart[$productID]);
            session()->put('cart', $cart);
        }

        return redirect()->back()->with('success', 'Product removed from cart');
    }


    public function applyCoupon(Request $request){
        $couponCode = $request->input('coupon');

            $validCoupons = [
            'DISCOUNT10' => 10, // $10 off
            'PERCENT20' => 0.2  // 20% off
        ];

        if (!array_key_exists($couponCode, $validCoupons)) {
            return back()->with('error', 'Invalid coupon code.');
        }

        session([
            'coupon'=>$couponCode,
        ]);

        return redirect()->back()->with('success', 'Coupon Applied');
    }
```



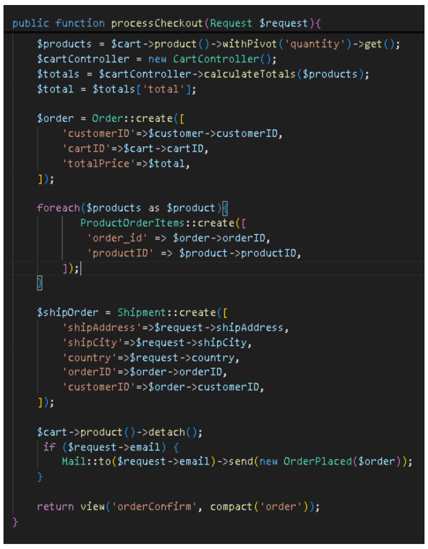
```php
{
    public function removeItem($productID){

    public function updateQuantity(Request $request){
        $request->validate([
            'productID' => 'required|exists:product,productID',
            'quantity' => 'required|integer|min:1|max:100',
        ]);

        if (Auth::check()) {
            $customer = Auth::user()->customer;
            $cart = Cart::where('customerID', $customer->customerID)->first();

            if ($cart) {
                $item = CartItems::where('cartID', $cart->cartID)->where('productID', $request->productID)->first();

                if ($item) {
                    $item->quantity = $request->quantity;
                    $item->save();
                }
            }
        } else {
            $cart = session()->get('cart', []);
            if (isset($cart[$request->productID])) {
                $cart[$request->productID]['quantity'] = $request->quantity;
                session()->put('cart', $cart);
            }
        }
        return redirect()->back()->with('success', 'Quantity updated.');
    }

    public function calculateTotals($allItems){
        $subtotal = 0;
        foreach ($allItems as $product) {
            $subtotal += $product->price * ($product->pivot->quantity ?? 1);
        }
        $couponCode = session('coupon');
        $discount = 0;
        if ($couponCode === 'DISCOUNT10') {
            $discount = 10;
        } elseif ($couponCode === 'PERCENT20') {
            $discount = $subtotal * 0.2;
        }
        $total = $subtotal - $discount;

        return compact('subtotal', 'discount', 'total', 'couponCode');
    }
```

1.12.4 Checkout controller:

- index(): this method retrieves the authenticated customer's info and their most recent shipping address (if available), then returns the checkout view with this data.
- processCheckout(): this method processes all logics needed for the checkout page where it validates user input, updates customer info, calculates the total price, creates an order, links products to that order, creates a shipment record, clears the cart, and sends a confirmation email if provided.



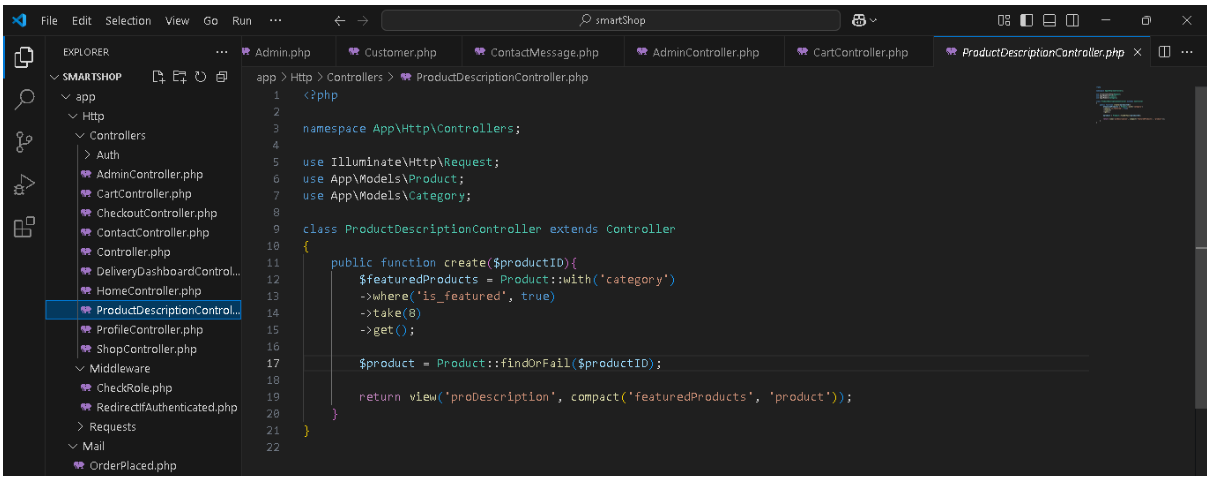
```php
class CheckoutController extends Controller
{
    public function index()
    {
        $user = Auth::user();
        $customer = $user->customer;

        if (!$customer) {
            return redirect()->route('cart')->with('error', 'Customer details not found.');
        }

         $latestShipment = Shipment::where('customerID', $customer->customerID)->orderBy('created_at', 'desc')->first();
        return view('checkout', compact('customer', 'latestShipment'));
    }

    public function processCheckout(Request $request){
        $request->validate([
            'full_name' => 'required|string',
            'phoneNB' => 'required|string|min:8',
            'email' => 'nullable|email|max:100',
            'shipAddress' => 'required|string|max:100',
            'shipCity' => 'required|string|max:100',
            'country'=> 'required|string|max:100',
        ]);

        $customer = Auth::user()->customer;
        $cart = $customer->cart;

        $customer->shippAddress = $request->shipAddress;
        $customer->phoneNB = $request->phoneNB;
        $customer->save();


        $products = $cart->product()->withPivot('quantity')->get();
        $cartController = new CartController();
        $totals = $cartController->calculateTotals($products);
        $total = $totals['total'];
```



```php
public function processCheckout(Request $request){
    $products = $cart->product()->withPivot('quantity')->get();
    $cartController = new CartController();
    $totals = $cartController->calculateTotals($products);
    $total = $totals['total'];

    $order = Order::create([
        'customerID'=>$customer->customerID,
        'cartID'=>$cart->cartID,
        'totalPrice'=>$total,
    ]);

    foreach($products as $product){
            ProductOrderItems::create([
             'order_id' => $order->orderID,
             'productID' => $product->productID,
        ]);
    }

    $shipOrder = Shipment::create([
        'shipAddress'=>$request->shipAddress,
        'shipCity'=>$request->shipCity,
        'country'=>$request->country,
        'orderID'=>$order->orderID,
        'customerID'=>$order->customerID,
    ]);

    $cart->product()->detach();
     if ($request->email) {
        Mail::to($request->email)->send(new OrderPlaced($order));
    }

    return view('orderConfirm', compact('order'));
}
```

1.12.5 Contact controller:

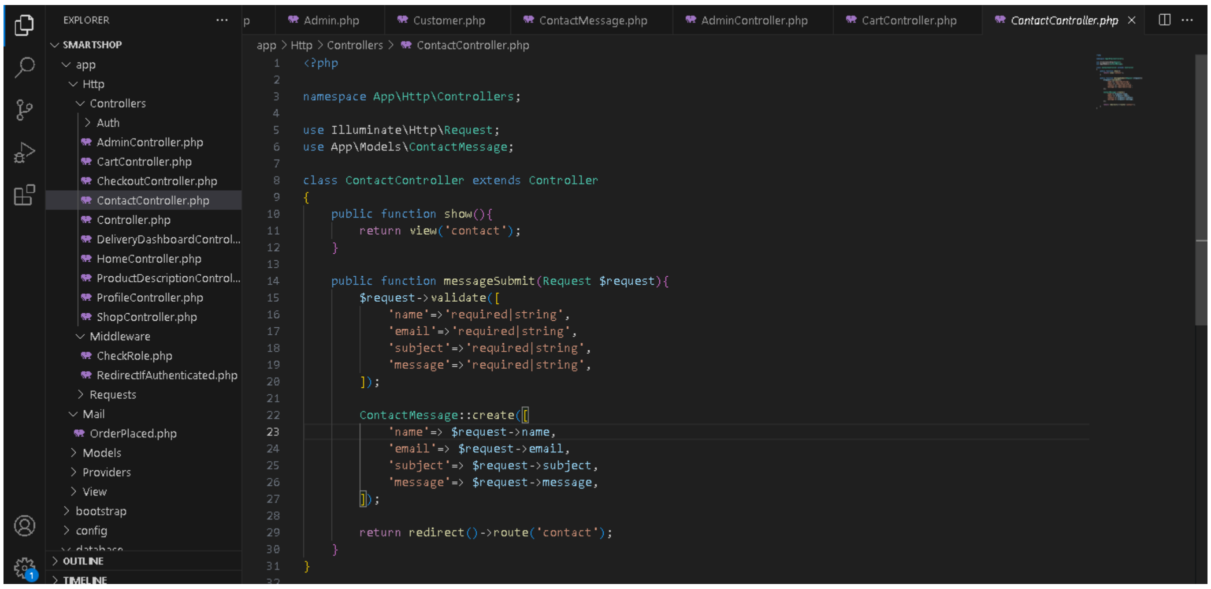
- show(): this method returns contact view.
- messageSubmit(): validates fields submitted in the contact form then creates a record in the contact message table.



```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;
use App\Models\Category;

class ProductDescriptionController extends Controller
{
    public function create($productID){
        $featuredProducts = Product::with('category')
        ->where('is_featured', true)
        ->take(8)
        ->get();

        $product = Product::findOrFail($productID);

        return view('proDescription', compact('featuredProducts', 'product'));
    }
}
```

1.12.6 Product description controller:

- create(): this method gets the first 8 featured products to be displayed under the product description, and finds all products based on their ids, then return product description view with the passed featured products.

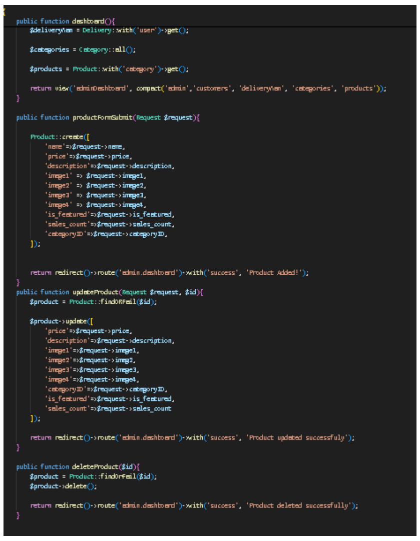


```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\ContactMessage;

class ContactController extends Controller
{
    public function show(){
        return view('contact');
    }

    public function messageSubmit(Request $request){
        $request->validate([
            'name'=>'required|string',
            'email'=>'required|string',
            'subject'=>'required|string',
            'message'=>'required|string',
        ]);

        ContactMessage::create([
            'name'=> $request->name,
            'email'=> $request->email,
            'subject'=> $request->subject,
            'message'=> $request->message,
        ]);

        return redirect()->route('contact');
    }
}
```

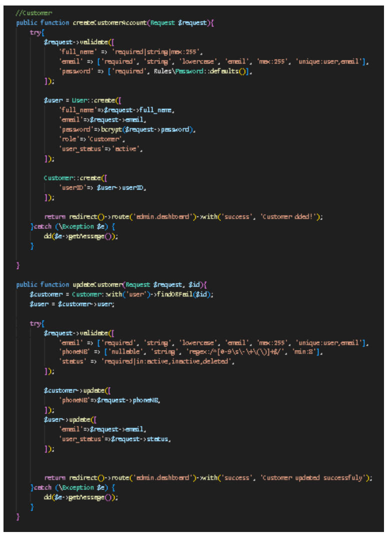
1.12.7 Admin controller:

- Dashboard(): handles the logic behind fetching all data needed for displaying customers, delivery men, categories, products and gets the first auth admin user. Then passes them in the method compact in the return view
- productFormSubmit(): retrieves all filled in fields in the create product form and add then to the product table in the database, then return the admin to the admin dashboard.
- UpdateProduct(): handles the logic behind updating any field based on the submitted form such as updating price, description, images, etc., and redirect to the admin dashboard
- deleteProduct(): gets the product id and compare it in the product table and get the first matching entry then delete the whole row of this id.
- CreateCustomerAccount(): validates filled in area of the create account form for customers and add them to the user and customer table.
- UpdateCustomer(): it gets the customer id, compare it with ids in the table, gets the first matching or show error if not found, then update the record in the user and the customer tables
- deleteCustomer(): get the first matching id in the customer table and delete his whole record in both user and customer tables.

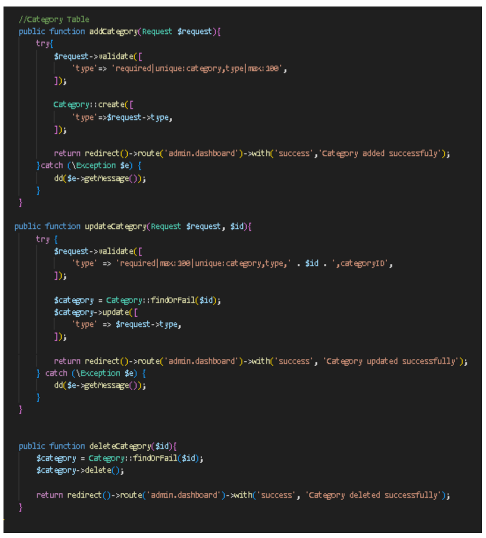
- createDeliveryAccount():validates the filled in fields, then create a new record in both user and delivery tables.
- updateDeliveryMan():validates the required fields for update and update them in delivery table.
- deleteDeliveryMan(): gets the first matching delivery id and delete the whole record in both user and delivery tables, or show error if id not found.
- addCategory(): adds a new category in the category table after validating the filled in fields.
- updateCategory(): it updates type field in category table based on the its matched id.
- deleteCategory(): searches for the first matching record based on the given id and delete its record in the category table.



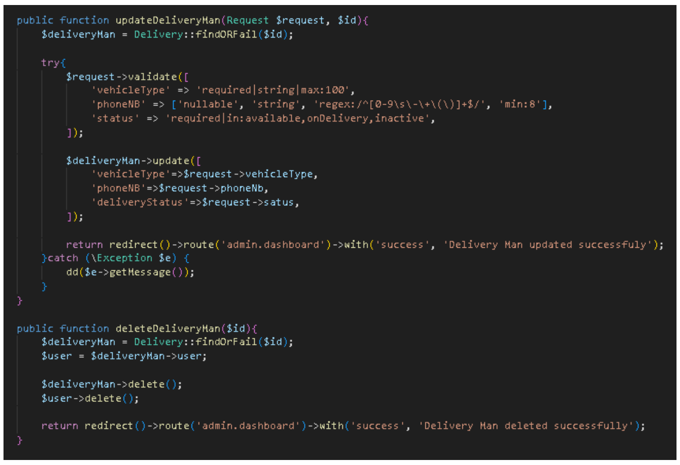
```php
{
    public function dashboard(){
        $deliveryMen = Delivery::with('user')->get();

        $categories = Category::all();

        $products = Product::with('category')->get();

        return view('adminDashboard', compact('admin','customers', 'deliveryMen', 'categories', 'products'));
    }

    public function productFormSubmit(Request $request){

        Product::create([
            'name'=>$request->name,
            'price'=>$request->price,
            'description'=>$request->description,
            'image1' => $request->image1,
            'image2' => $request->image2,
            'image3' => $request->image3,
            'image4' => $request->image4,
            'is_featured'=>$request->is_featured,
            'sales_count'=>$request->sales_count,
            'categoryID'=>$request->categoryID,
        ]);


        return redirect()->route('admin.dashboard')->with('success', 'Product Added!');
    }
    public function updateProduct(Request $request, $id){
        $product = Product::findORFail($id);

        $product->update([
            'price'=>$request->price,
            'description'=>$request->description,
            'image1'=>$request->image1,
            'image2'=>$request->img2,
            'image3'=>$request->image3,
            'image4'=>$request->image4,
            'categoryID'=>$request->categoryID,
            'is_featured'=>$request->is_featured,
            'sales_count'=>$request->sales_count
        ]);

        return redirect()->route('admin.dashboard')->with('success', 'Product updated successfuly');
    }

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

        return redirect()->route('admin.dashboard')->with('success', 'Product deleted successfully');
    }
```



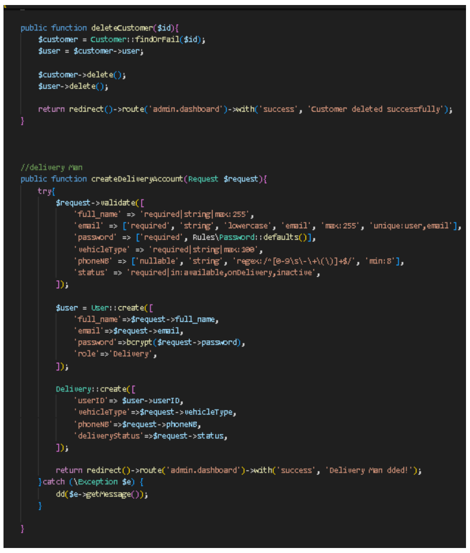
```php
    //Customer
    public function createCustomerAccount(Request $request){
        try{
            $request->validate([
                'full_name' => 'required|string|max:255',
                'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:user,email'],
                'password' => ['required', Rules\Password::defaults()],
            ]);

            $user = User::create([
                'full_name'=>$request->full_name,
                'email'=>$request->email,
                'password'=>bcrypt($request->password),
                'role'=>'Customer',
                'user_status'=>'active',
            ]);

            Customer::create([
                'userID'=> $user->userID,
            ]);

            return redirect()->route('admin.dashboard')->with('success', 'Customer dded!');
        }catch (\Exception $e) {
            dd($e->getMessage());
        }

    }

    public function updateCustomer(Request $request, $id){
        $customer = Customer::with('user')->findORFail($id);
        $user = $customer->user;

        try{
            $request->validate([
                'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:user,email'],
                'phoneNB' => ['nullable', 'string', 'regex:/^[0-9\s\-\+\(\)]+$/', 'min:8'],
                'status' => 'required|in:active,inactive,deleted',
            ]);

            $customer->update([
                'phoneNB'=>$request->phoneNB,
            ]);
            $user->update([
                'email'=>$request->email,
                'user_status'=>$request->status,
            ]);


            return redirect()->route('admin.dashboard')->with('success', 'Customer updated successfuly');
        }catch (\Exception $e) {
            dd($e->getMessage());
        }
    }
```



```php
    //Category Table
    public function addCategory(Request $request){
        try{
            $request->validate([
                'type'=> 'required|unique:category,type|max:100',
            ]);

            Category::create([
                'type'=>$request->type,
            ]);

            return redirect()->route('admin.dashboard')->with('success','Category added successfuly');
        }catch (\Exception $e) {
            dd($e->getMessage());
        }
    }

   public function updateCategory(Request $request, $id){
        try {
            $request->validate([
                'type' => 'required|max:100|unique:category,type,' . $id . ',categoryID',
            ]);

            $category = Category::findOrFail($id);
            $category->update([
                'type' => $request->type,
            ]);

            return redirect()->route('admin.dashboard')->with('success', 'Category updated successfully');
        } catch (\Exception $e) {
            dd($e->getMessage());
        }
    }


    public function deleteCategory($id){
        $category = Category::findOrFail($id);
        $category->delete();

        return redirect()->route('admin.dashboard')->with('success', 'Category deleted successfully');
    }
```



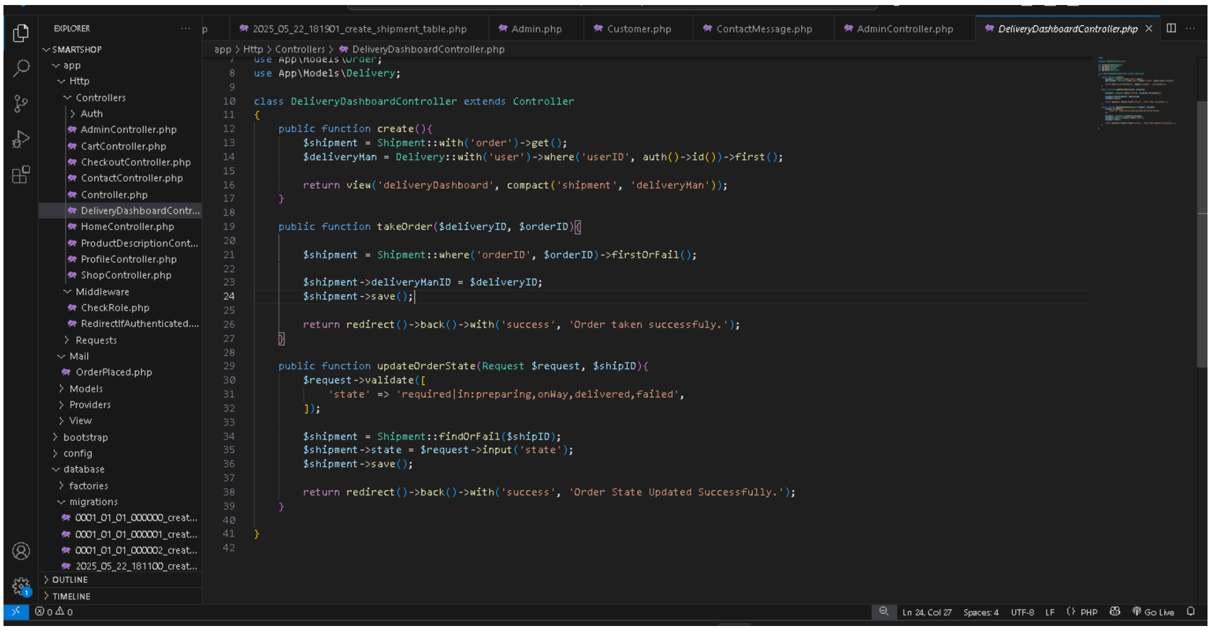
```php
public function updateDeliveryMan(Request $request, $id){
    $deliveryMan = Delivery::findORFail($id);

    try{
        $request->validate([
            'vehicleType' => 'required|string|max:100',
            'phoneNB' => ['nullable', 'string', 'regex:/^[0-9\s\-\+\(\)]+$/', 'min:8'],
            'status' => 'required|in:available,onDelivery,inactive',
        ]);

        $deliveryMan->update([
            'vehicleType'=>$request->vehicleType,
            'phoneNB'=>$request->phoneNb,
            'deliveryStatus'=>$request->satus,
        ]);

        return redirect()->route('admin.dashboard')->with('success', 'Delivery Man updated successfuly');
    }catch (\Exception $e) {
        dd($e->getMessage());
    }
}

public function deleteDeliveryMan($id){
    $deliveryMan = Delivery::findORFail($id);
    $user = $deliveryMan->user;

    $deliveryMan->delete();
    $user->delete();

    return redirect()->route('admin.dashboard')->with('success', 'Delivery Man deleted successfully');
}
```



```php
    public function deleteCustomer($id){
        $customer = Customer::findOrFail($id);
        $user = $customer->user;

        $customer->delete();
        $user->delete();

        return redirect()->route('admin.dashboard')->with('success', 'Customer deleted successfully');
    }



    //delivery Man
    public function createDeliveryAccount(Request $request){
        try{
            $request->validate([
                'full_name' => 'required|string|max:255',
                'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:user,email'],
                'password' => ['required', Rules\Password::defaults()],
                'vehicleType' => 'required|string|max:100',
                'phoneNB' => ['nullable', 'string', 'regex:/^[0-9\s\-\+\(\)]+$/', 'min:8'],
                'status' => 'required|in:available,onDelivery,inactive',
            ]);

            $user = User::create([
                'full_name'=>$request->full_name,
                'email'=>$request->email,
                'password'=>bcrypt($request->password),
                'role'=>'Delivery',
            ]);

            Delivery::create([
                'userID'=> $user->userID,
                'vehicleType'=>$request->vehicleType,
                'phoneNB'=>$request->phoneNB,
                'deliveryStatus'=>$request->status,
            ]);

            return redirect()->route('admin.dashboard')->with('success', 'Delivery Man dded!');
        }catch (\Exception $e) {
            dd($e->getMessage());
        }

    }
```

1.12.8 Delivery dashboard controller:

- create(): retrieves all shipments with their related orders and the logged in delivery man's information, then returns the delivery dashboard view with the retrieved data.
- takeOrder(): assigns the given delivery man to the specified order by updating the deliveryManID in the shipment record.
- updateOrderState(): this method validates and updates the shipment's delivery state (preparing, onWay, delivered, failed), then redirects back with a success message.

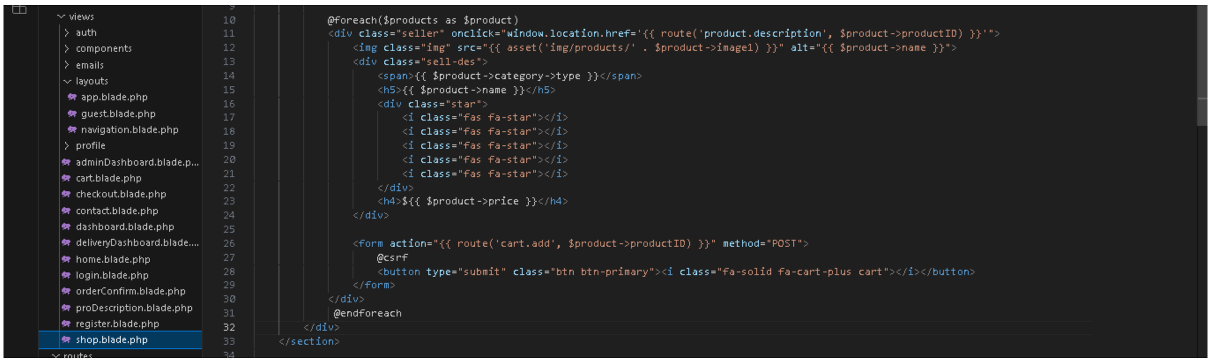


```php
use App\Models\Order;
use App\Models\Delivery;

class DeliveryDashboardController extends Controller
{
    public function create(){
        $shipment = Shipment::with('order')->get();
        $deliveryMan = Delivery::with('user')->where('userID', auth()->id())->first();

        return view('deliveryDashboard', compact('shipment', 'deliveryMan'));
    }

    public function takeOrder($deliveryID, $orderID){

        $shipment = Shipment::where('orderID', $orderID)->firstOrFail();

        $shipment->deliveryManID = $deliveryID;
        $shipment->save();

        return redirect()->back()->with('success', 'Order taken successfuly.');
    }

    public function updateOrderState(Request $request, $shipID){
        $request->validate([
            'state' => 'required|in:preparing,onWay,delivered,failed',
        ]);

        $shipment = Shipment::findOrFail($shipID);
        $shipment->state = $request->input('state');
        $shipment->save();

        return redirect()->back()->with('success', 'Order State Updated Successfully.');
    }

}
```

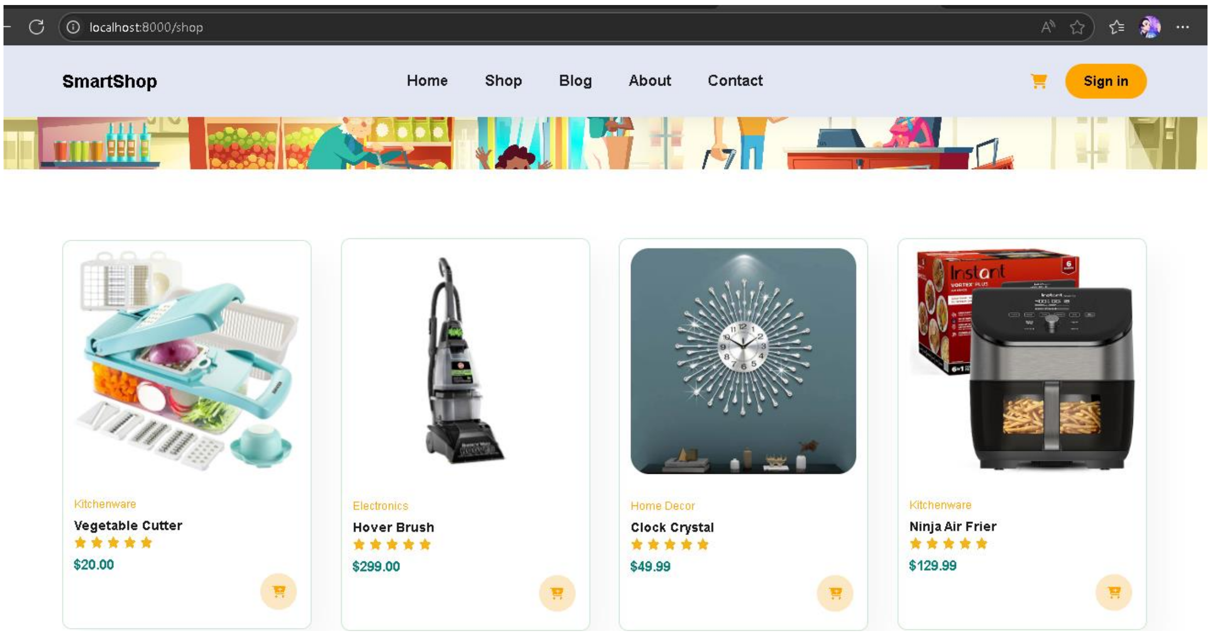
Chapter Four: Integration

Linking frontend with backend for dynamic and interactive user experience, the following several steps taken to achieve website interaction:

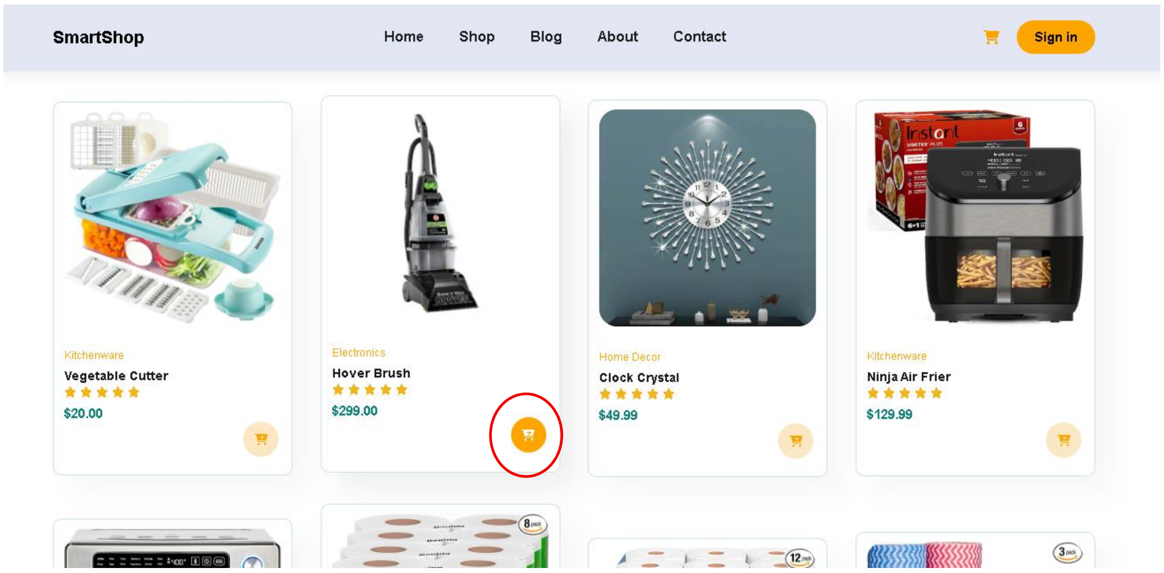
- First, created the forms in the frontend pages like register, login, checkout, contact message and those in admin and delivery dashboards. When users filled them and clicked the submit button, the data is sent to their corresponding controllers using the POST routes.
- Used Laravel blade templates to show data from the database in the frontend. For example, in the shop page, there is a for each loop that loops through all products passed in the method controller and displays then in a grid.



```
            @foreach($products as $product)
            <div class="seller" onclick="window.location.href='{{ route('product.description', $product->productID) }}'">
                <img class="img" src="{{ asset('img/products/' . $product->image1) }}" alt="{{ $product->name }}">
                <div class="sell-des">
                    <span>{{ $product->category->type }}</span>
                    <h5>{{ $product->name }}</h5>
                    <div class="star">
                        <i class="fas fa-star"></i>
                        <i class="fas fa-star"></i>
                        <i class="fas fa-star"></i>
                        <i class="fas fa-star"></i>
                        <i class="fas fa-star"></i>
                    </div>
                    <h4>${{ $product->price }}</h4>
                </div>

                <form action="{{ route('cart.add', $product->productID) }}" method="POST">
                    @csrf
                    <button type="submit" class="btn btn-primary"><i class="fa-solid fa-cart-plus cart"></i></button>
                </form>
            </div>
             @endforeach
        </div>
    </section>
```



- When users click the buttons like “Add to Cart” or “Signin”, these actions call backend methods created in the controllers, which handles the logic like saving orders or updating shipment status.

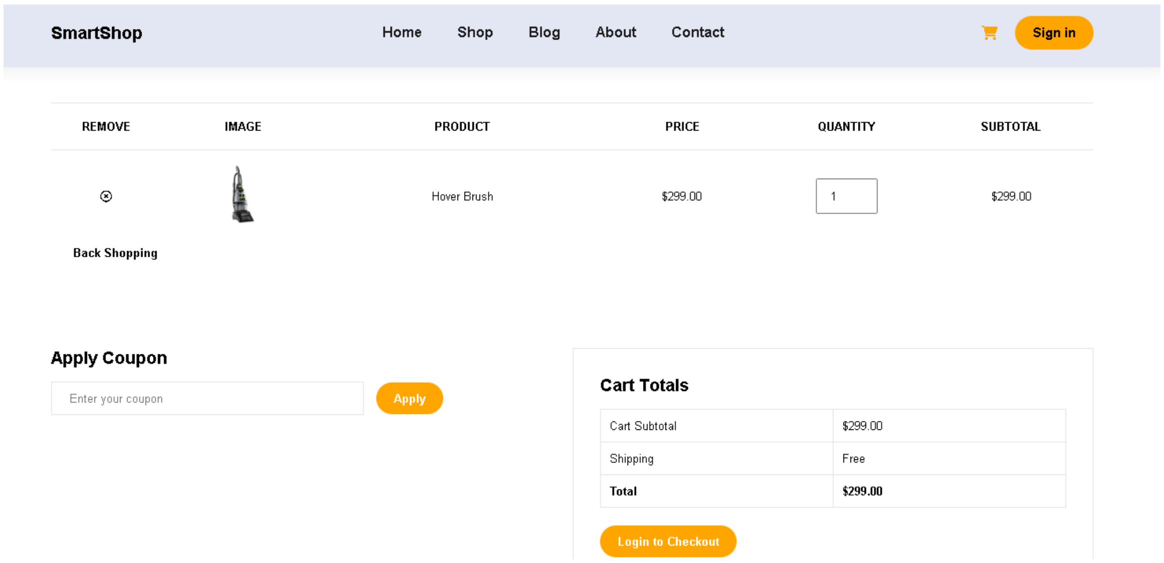


Kitchenware
Vegetable Cutter
$20.00

Electronics
Hover Brush
$299.00

Home Decor
Clock Crystal
$49.99

Kitchenware
Ninja Air Frier
$129.99



REMOVE	IMAGE	PRODUCT	PRICE	QUANTITY	SUBTOTAL
⊗		Hover Brush	$299.00	1	$299.00

Apply Coupon

Enter your coupon

Cart Totals

Cart Subtotal	$299.00
Shipping	Free
Total	**$299.00**

- Finally, I pass the data back from the controller to the blade views using compact() method in the return statements, in order to show the latest updates to the users in real time.

These steps help in connecting both sides of the project and make it fully functional for customers, admins and delivery men.

Chapter Five: Testing & Polish

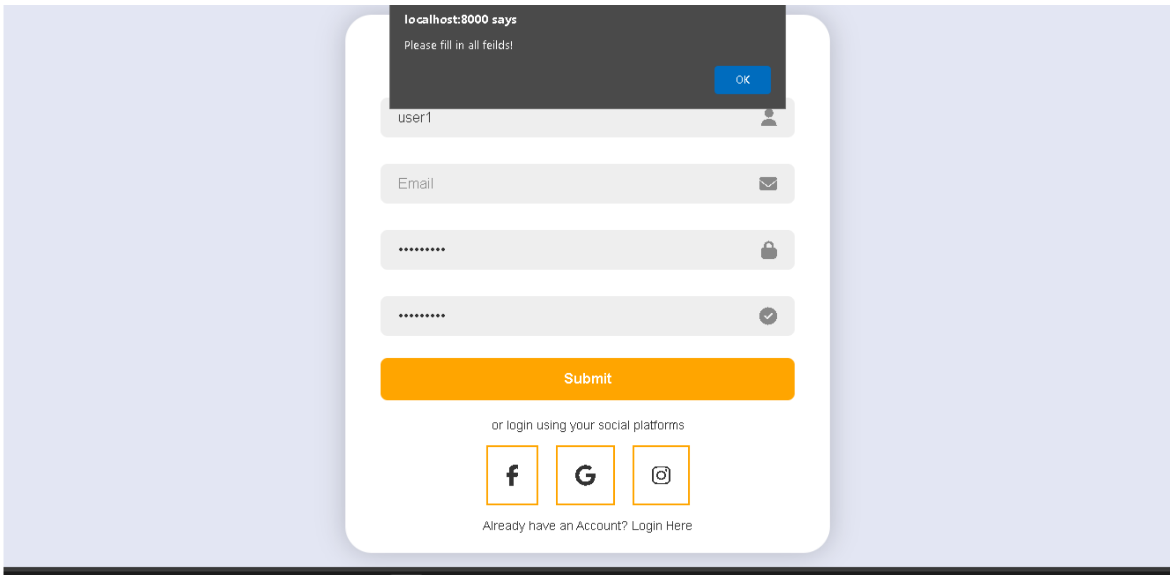
1.13 Fix bugs:

While working on this project, I faced several bugs that helps me learn how to debug and improve the system:

- One major bug happened when I tried to use custome middleware for checking user roles. Laravel kept giving an error saying: ”Class ‘role’ does not exist”. After searching and asking my friend, we discovered that Laravel no longer uses Kernel.php for middleware registration in new versions. So. I fixed it by registering the alias in bootstrap/app.php file inside the withMiddleware() method section like this: $middleware->alias([‘role’=> \App\Http\Middleware\CheckRole::class,]);
- Another problem was data not saving correctly to the database during checkout. I realized I forgot to pass the right IDs when creating the order and shipment records. I used add() (dump and die) to debug and trace where the data stopped being passed. Once I found the mistake. I fixed the logic inside CheckoutController.
- In some views, products weren’t displaying correctly. I figured out the problem was due to not using withPivot(‘quantity’) when getting products from the cart, which made quantities not appear.
- Another bug I faced was when I implemented the checkout process; after submitting the form, the cart was not clearing and users could accidently resubmit the same order. I realized I needed to use $cart->product()->detach(); after placing the order, to remove all products from the cart. Adding this fixed the problem and ensured the cart resets after a successful order.

1.14 validate forms:

To make sure users enter the correct information in the forms, I used laravel’s built in form validation. This helped prevent errors like missing fields or entering invalid data. For example, in the checkout form, I used this $request->validate to check that the name, phone number, shipping address, city and country are filled in correctly. If any required data was missing or in wrong format (like an invalid email), Laravel automatically show an error message. I did the same for all fors like registration, login, contact and other ones. This helped in making the app more reliable and user-friendly.


localhost:8000 says

Please fill in all feilds!

OK

user1

Email

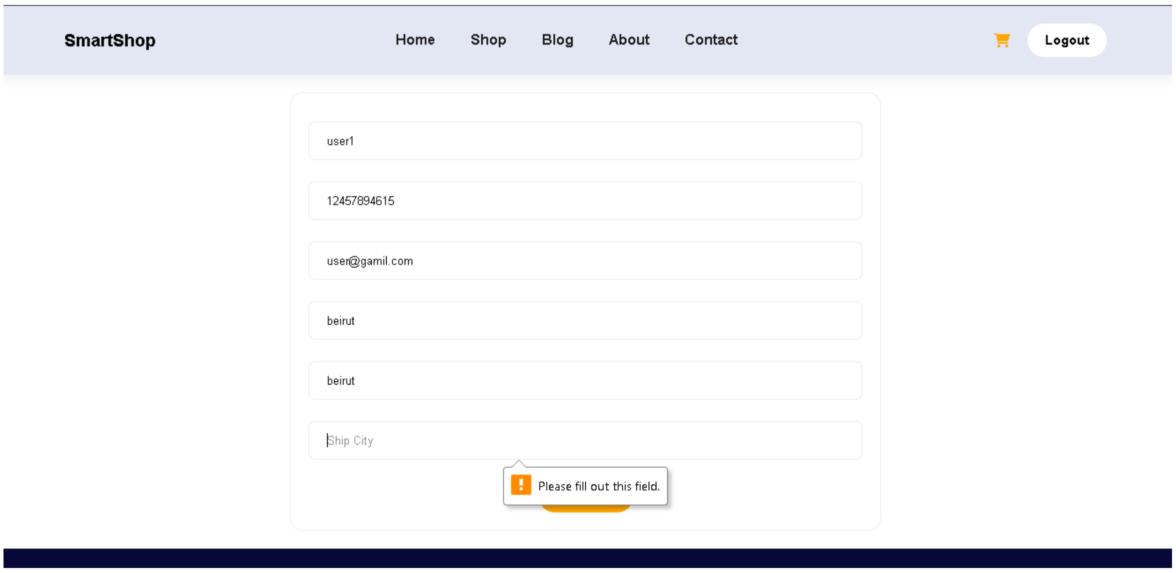
•••••••••

•••••••••

Submit

or login using your social platforms

Already have an Account? Login Here



user1

12457894615

user@gamil.com

beirut

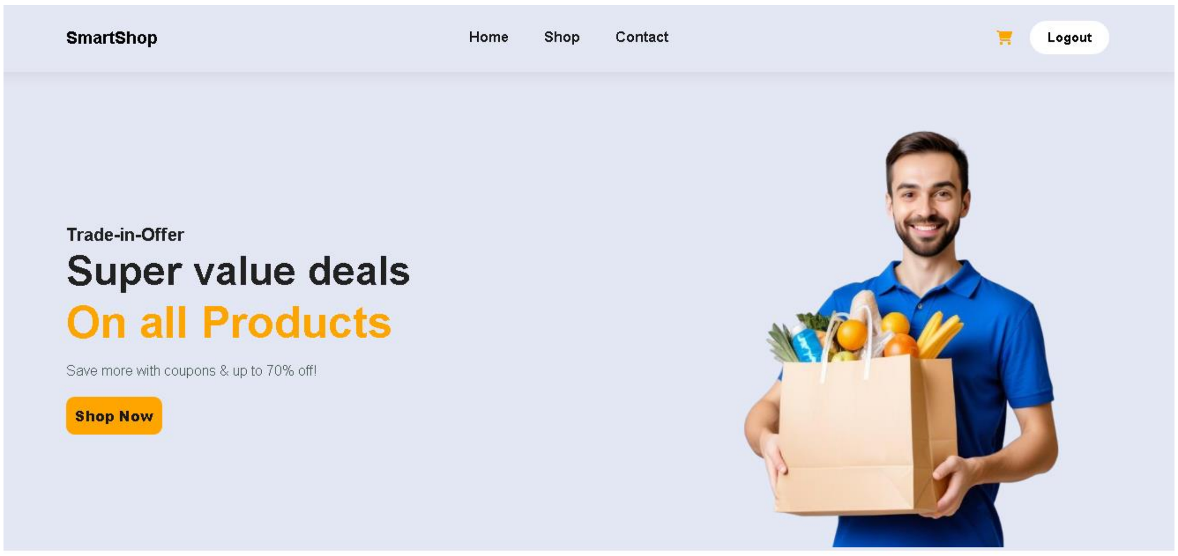
beirut

Ship City

Please fill out this field.

1.15 style pages:

used CSS to style pages as mentioned above, three css files one for dashboards, one for the login and register pages and one for the rest of the websites pages, the following are few screenshots of how the websites looks like after styling:


SmartShop

Home Shop Contact

Logout

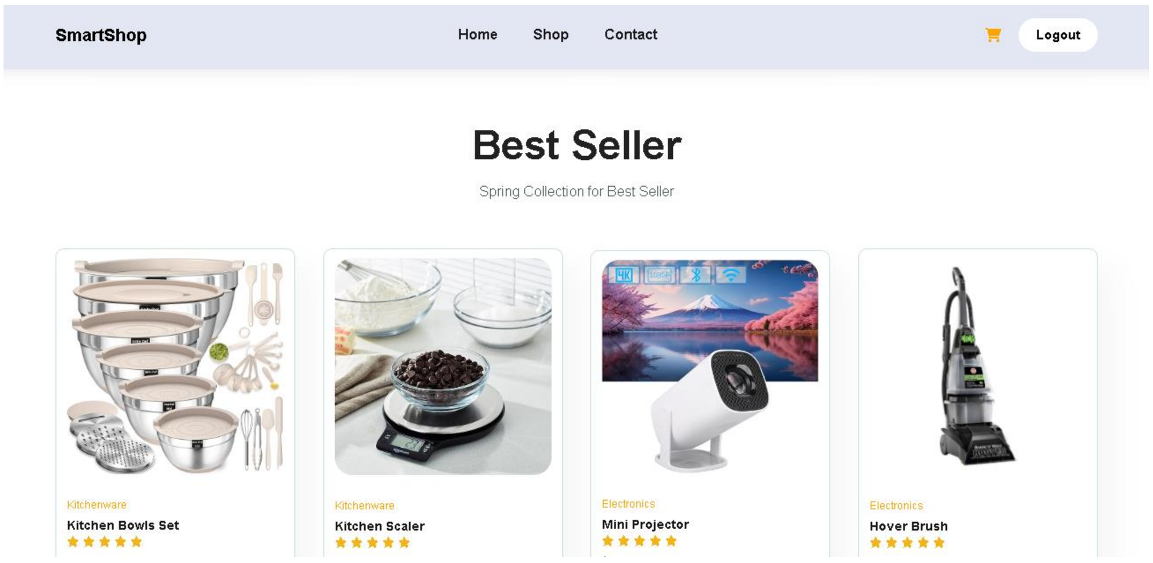
Trade-in-Offer

Super value deals

On all Products

Save more with coupons & up to 70% off!

Shop Now


SmartShop

Home Shop Contact

Logout

Best Seller

Spring Collection for Best Seller

Kitchenware

Kitchen Bowls Set

Kitchenware

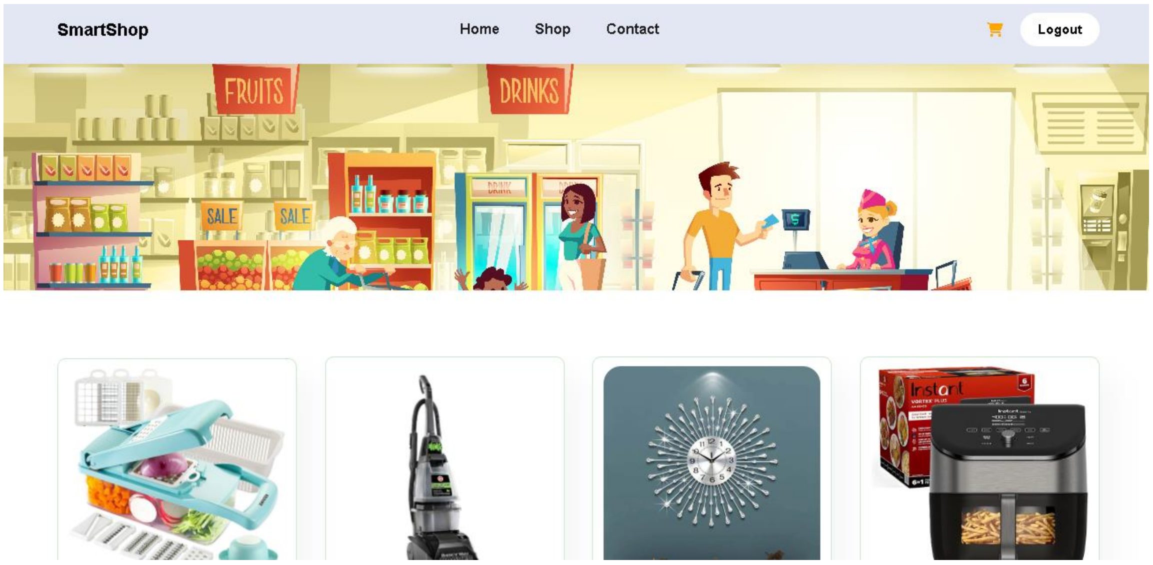
Kitchen Scaler

Electronics

Mini Projector

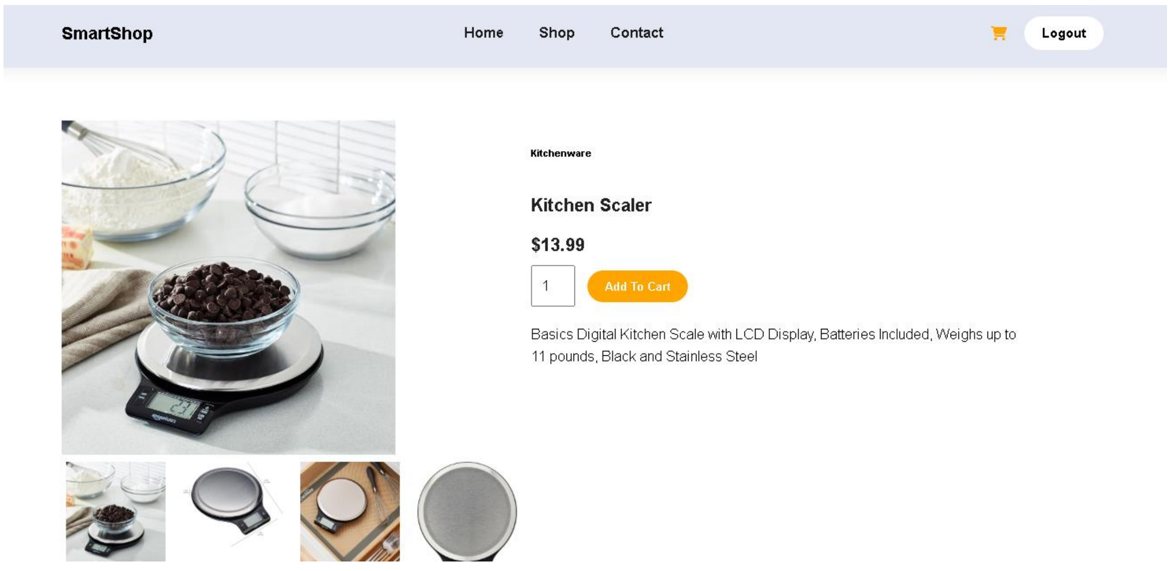
Electronics

Hover Brush


SmartShop

Home Shop Contact

Logout


SmartShop Home Shop Contact Logout

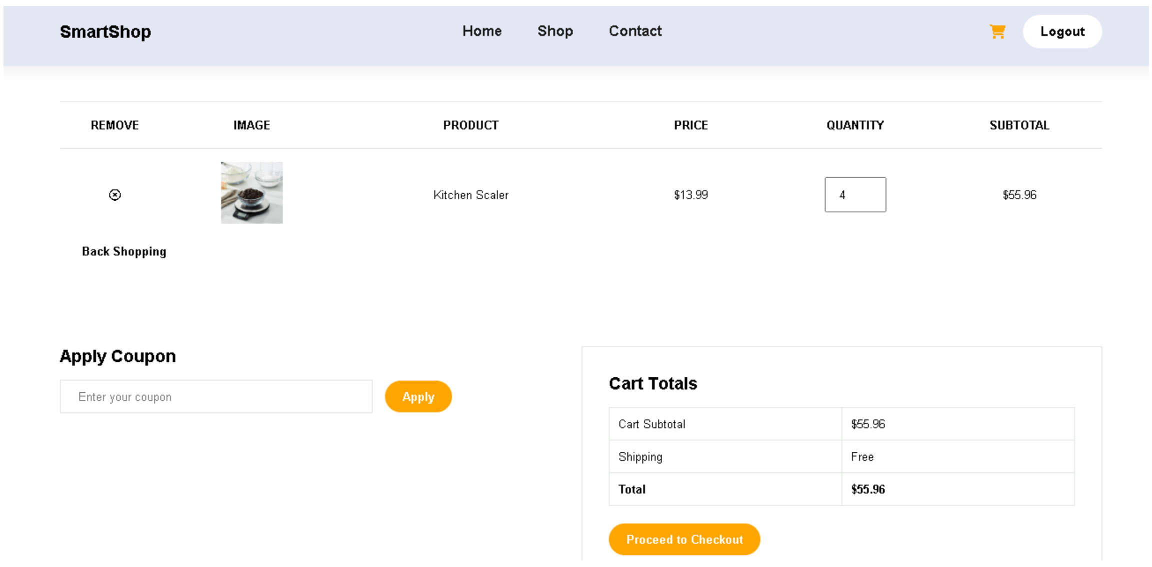
Kitchenware

Kitchen Scaler

$13.99

1 Add To Cart

Basics Digital Kitchen Scale with LCD Display, Batteries Included, Weighs up to 11 pounds, Black and Stainless Steel


SmartShop Home Shop Contact Logout

REMOVE	IMAGE	PRODUCT	PRICE	QUANTITY	SUBTOTAL
⊗		Kitchen Scaler	$13.99	4	$55.96

Back Shopping

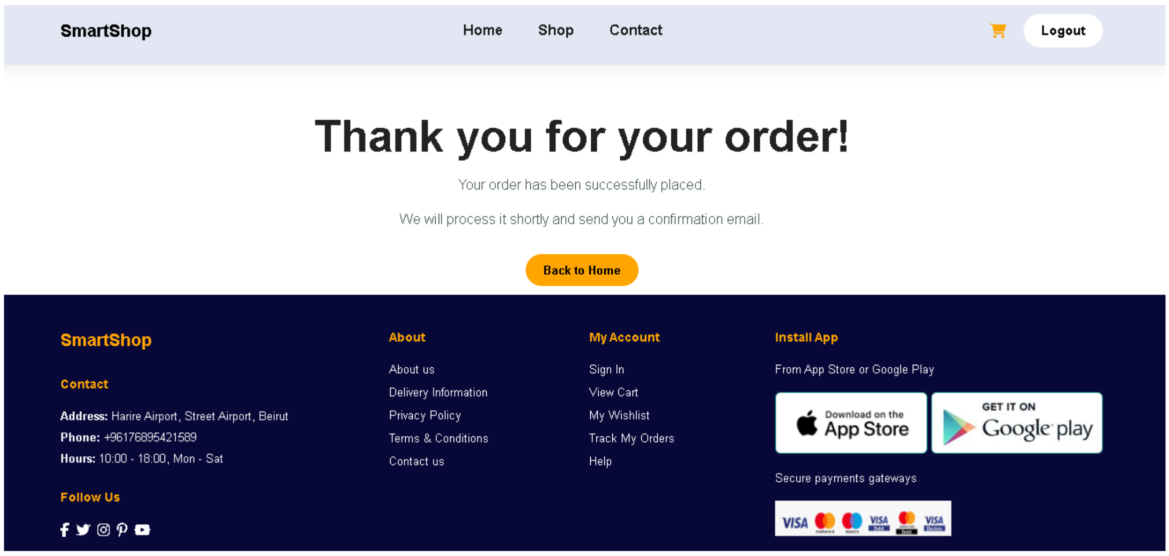
Apply Coupon

Enter your coupon Apply

Cart Totals

Cart Subtotal	$55.96
Shipping	Free
Total	**$55.96**

Proceed to Checkout


SmartShop Home Shop Contact Logout

Thank you for your order!

Your order has been successfully placed.

We will process it shortly and send you a confirmation email.

Back to Home

SmartShop

Contact

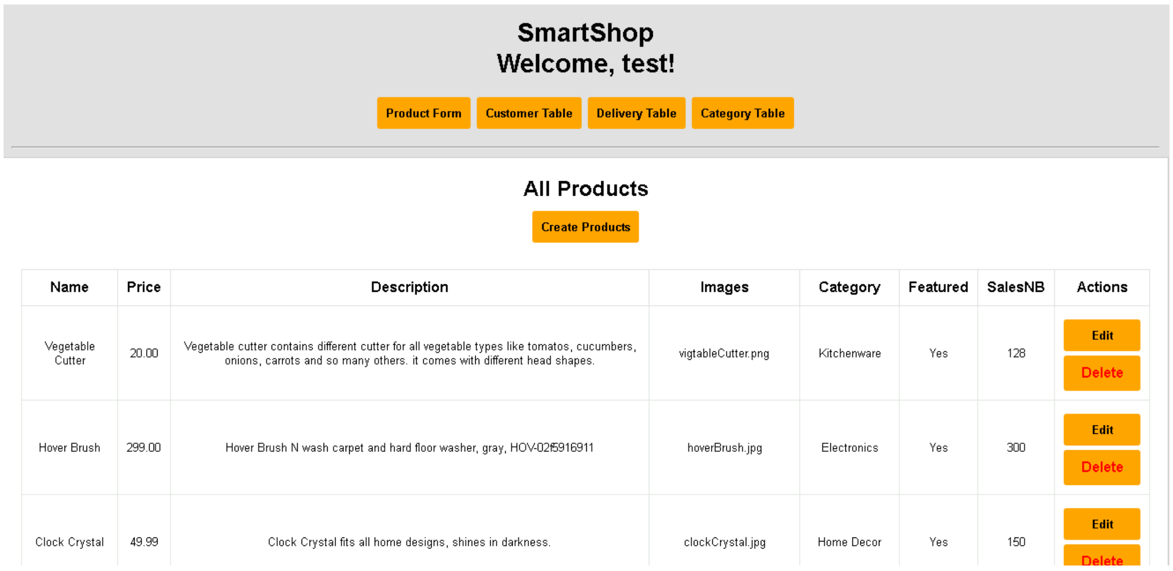
Address: Harire Airport, Street Airport, Beirut
Phone: +96176895421589
Hours: 10:00 - 18:00, Mon - Sat

Follow Us

About
About us
Delivery Information
Privacy Policy
Terms & Conditions
Contact us

My Account
Sign In
View Cart
My Wishlist
Track My Orders
Help

Install App
From App Store or Google Play
Download on the App Store
GET IT ON Google play
Secure payments gateways



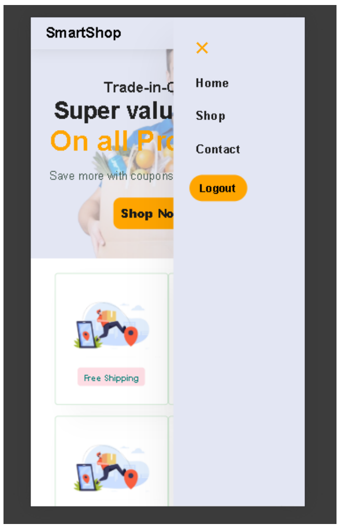
SmartShop
Welcome, test!

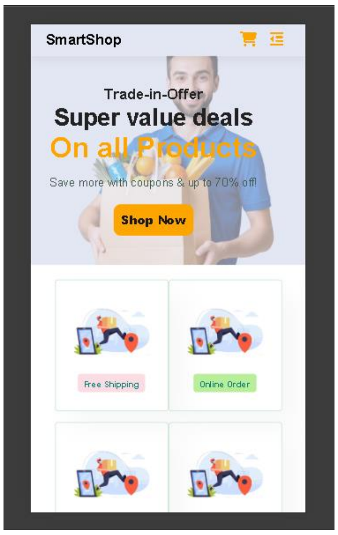
Product Form | Customer Table | Delivery Table | Category Table

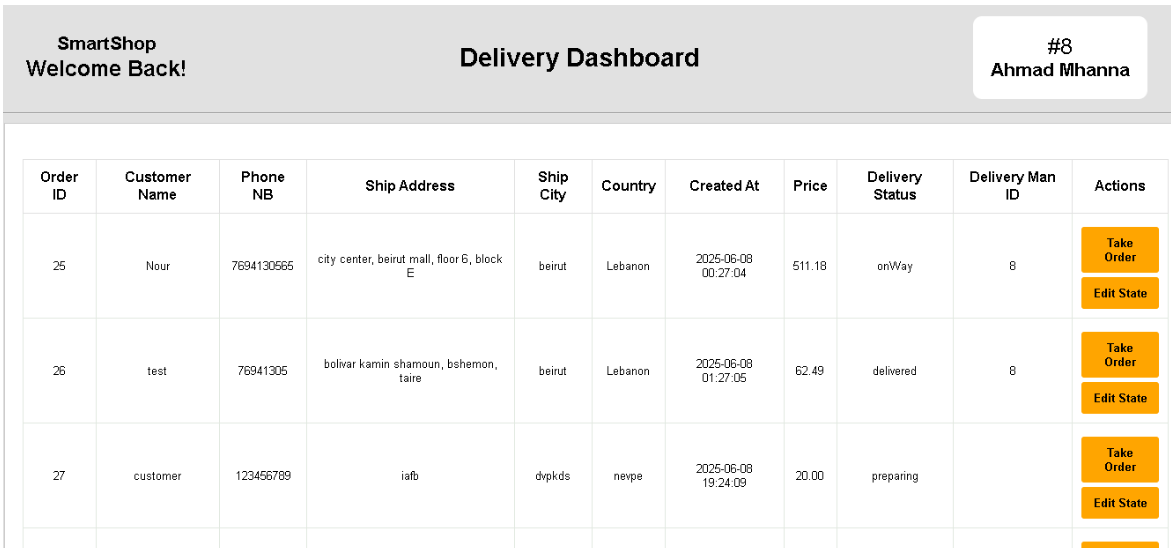
All Products

Create Products

Name	Price	Description	Images	Category	Featured	SalesNB	Actions
Vegetable Cutter	20.00	Vegetable cutter contains different cutter for all vegetable types like tomatos, cucumbers, onions, carrots and so many others. it comes with different head shapes.	vigtableCutter.png	Kitchenware	Yes	128	Edit Delete
Hover Brush	299.00	Hover Brush N wash carpet and hard floor washer, gray, HOV-02f5916911	hoverBrush.jpg	Electronics	Yes	300	Edit Delete
Clock Crystal	49.99	Clock Crystal fits all home designs, shines in darkness.	clockCrystal.jpg	Home Decor	Yes	150	Edit Delete







SmartShop
Welcome Back!

Delivery Dashboard

#8
Ahmad Mhanna

Order ID	Customer Name	Phone NB	Ship Address	Ship City	Country	Created At	Price	Delivery Status	Delivery Man ID	Actions
25	Nour	7694130565	city center, beirut mall, floor 6, block E	beirut	Lebanon	2025-06-08 00:27:04	511.18	onWay	8	Take Order Edit State
26	test	76941305	bolivar kamin shamoun, bshemon, taire	beirut	Lebanon	2025-06-08 01:27:05	62.49	delivered	8	Take Order Edit State
27	customer	123456789	iafb	dvpkds	nevpe	2025-06-08 19:24:09	20.00	preparing		Take Order Edit State