1. **INTRODUCTION**
   1. **Project Significance and Need**

The increasing demand for online shopping has transformed how people purchase products, including flowers, garlands, and pooja items. Many traditional flower shops lack an online presence, making it difficult for customers to browse, select, and order products conveniently.

**VivekFlowers** aims to solve this problem by providing a user-friendly e-commerce platform where customers can explore different categories, add products to their cart, and place orders seamlessly. This platform also includes an admin panel for managing products, categories, orders, and users.

* 1. **Existing System and Upgrades**
* Traditional flower shops rely on **physical stores** or **phone-based orders**.
* Limited product availability updates.
* No real-time order tracking or inventory management.
* Manual payment processing without online payment support.

**Upgraded System Features:**

* A **fully functional online store** where users can browse products and categories.
* **Product variation selection** for customized orders.
* **Secure login, registration, and checkout** using PayPal.
* **Admin dashboard** for managing products, orders, and analytics.
* **Search functionality** for easy product discovery.
  1. **Technology Stack**
* **Frontend**: React.js (for dynamic UI and component-based structure)
* **Backend:** Python Flask (for handling API requests and business logic)
* **Database:** MySQL (for data storage and retrieval)
* **Payment Integration:** PayPal
* **Deployment:** Hosted on a cloud server
* **Testing:** Unit and Integration Testing
  1. **Organization and Resources**

This project was developed during my **internship at SNInfoserv**. Available resources included:

* **Development tools:** Visual Studio Code, Postman, Git
* **Frameworks & Libraries:** React.js, Flask, XAMPP
* **Testing tools:** Postman, Jest, PyTest
  1. **Modules Description**

**1.5.1 User Panel:**

Vivek Flowers is an online store that sells flowers, garlands, pooja items, and more. The website offers a seamless shopping experience with the following features:

* **Home Page:**
  + A dynamic slider showcasing promotional banners.
  + Product suggestions based on trending or featured items.

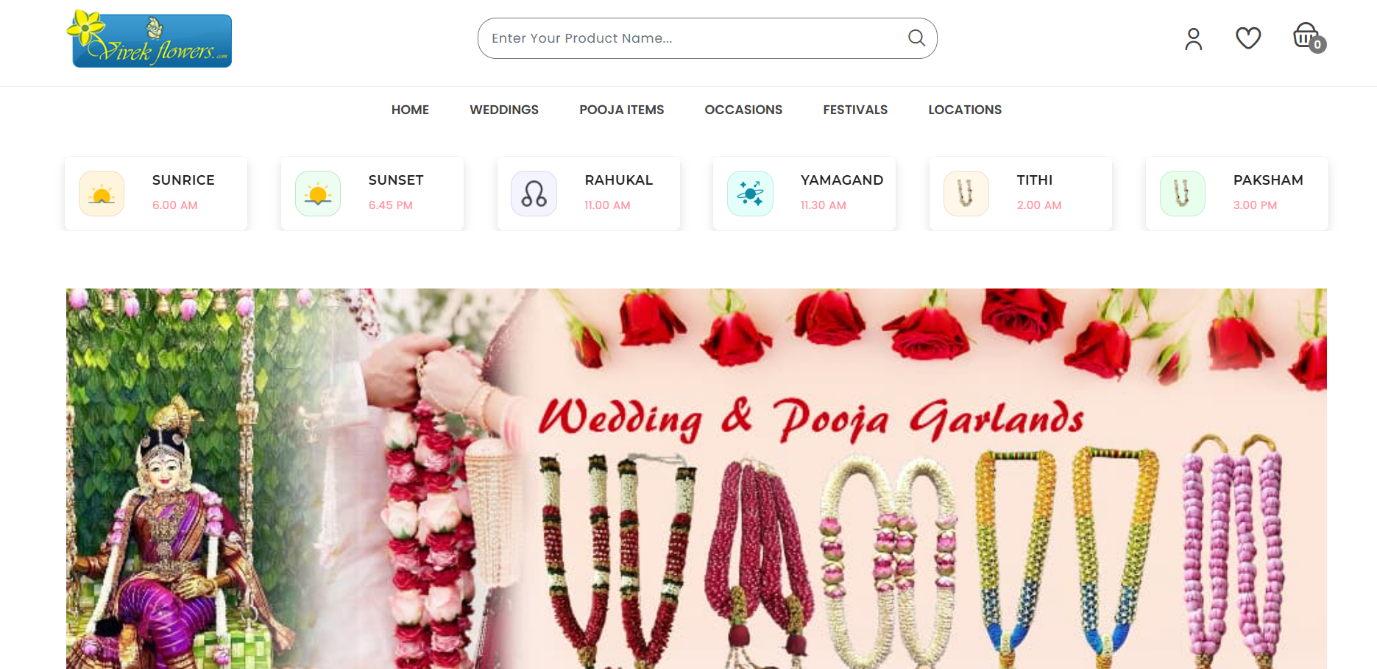


Figure 1.5.1.1 Home Page

* **Categories:**
  + The header includes categories such as Wedding, Pooja Items, Occasions, and Festivals.
  + Each category has its own subcategories, displaying relevant products.

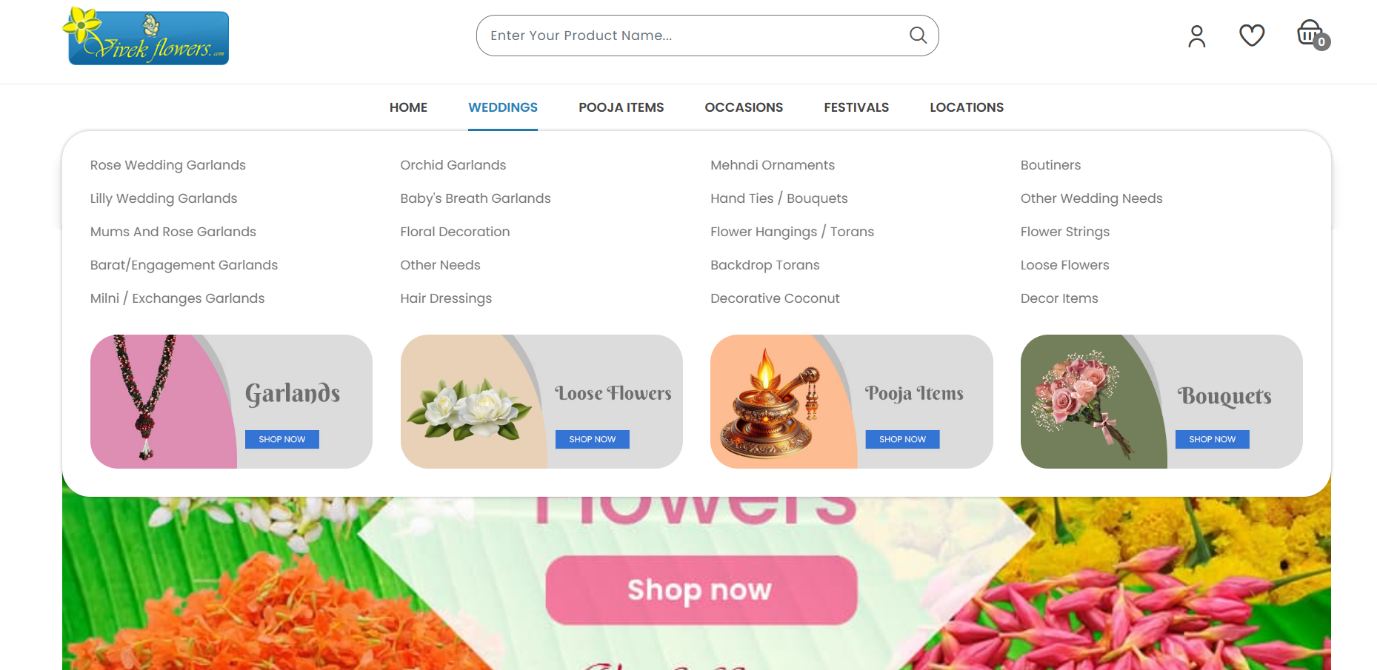


Figure 1.5.1.2 Category Page

* **Product Page:**
  + Users can view individual product details.
  + Related products are displayed for better recommendations.
  + Options to add products to the cart, wishlist, and select variations (size, type, etc.) before purchase.

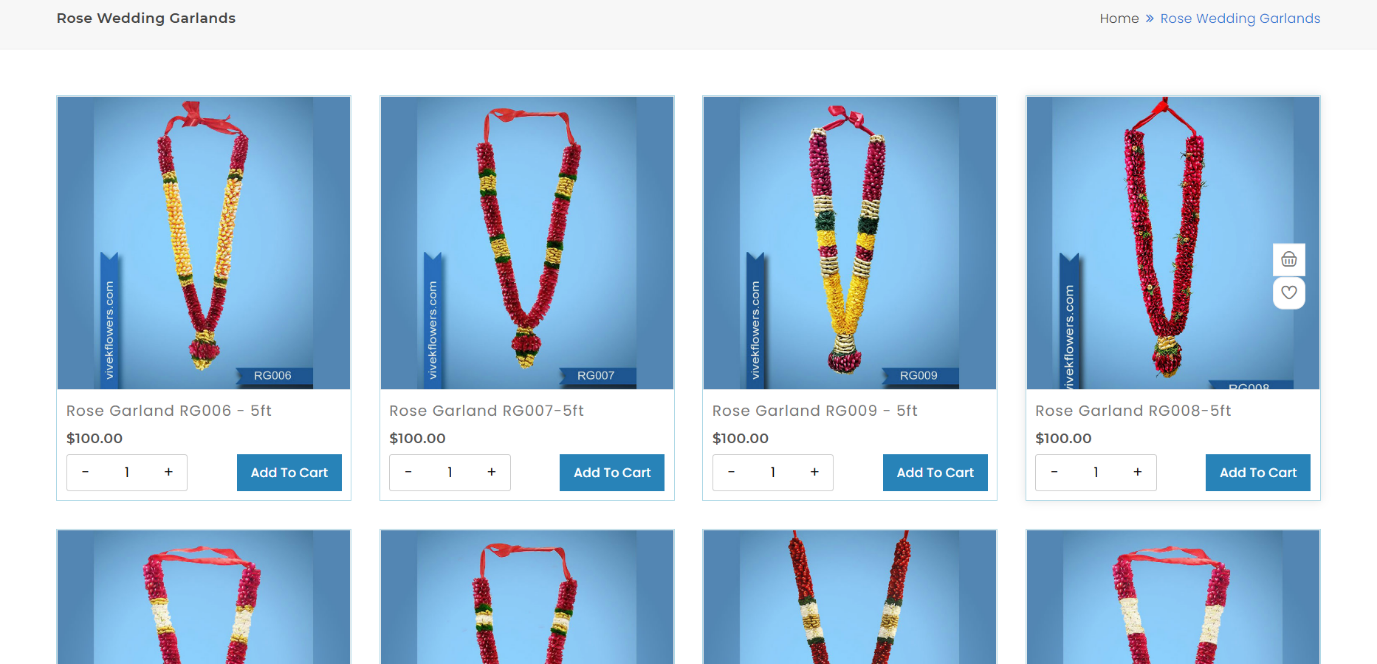


Figure 1.5.1.3 Product Page

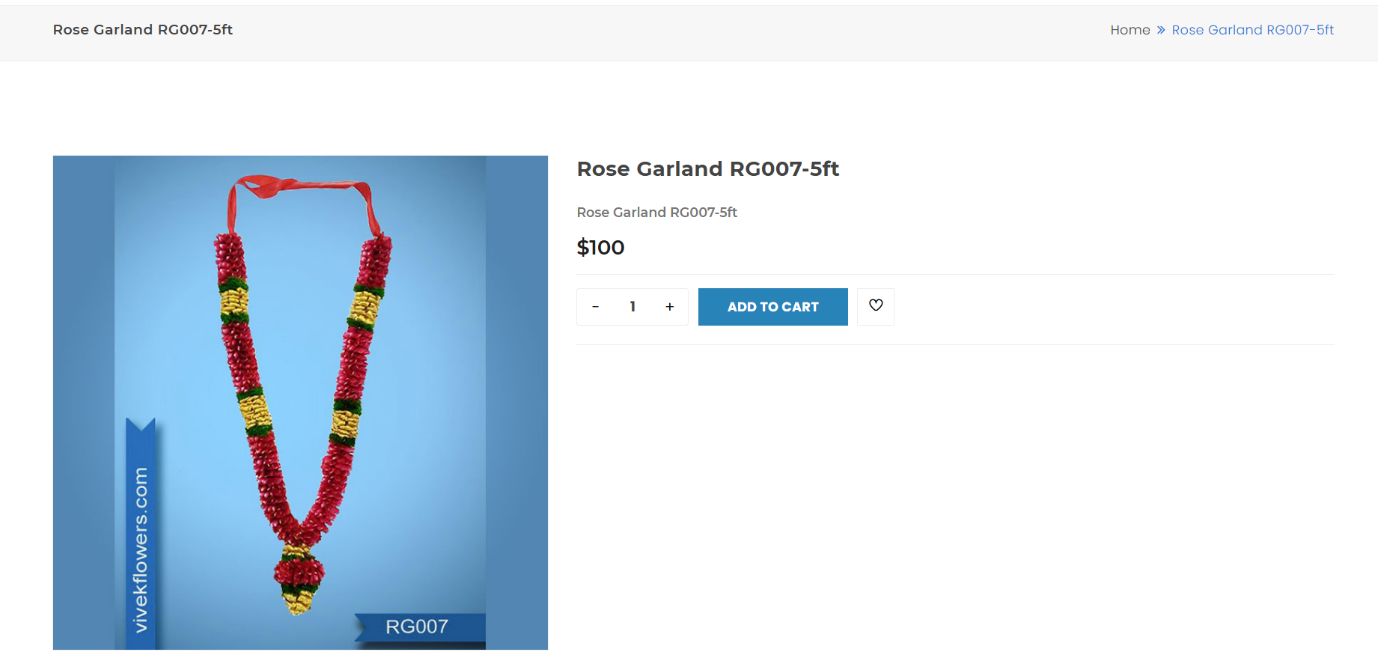


Figure 1.5.1.4 Single Product Page

* **Search Bar:**
  + Located in the header for quick product discovery.

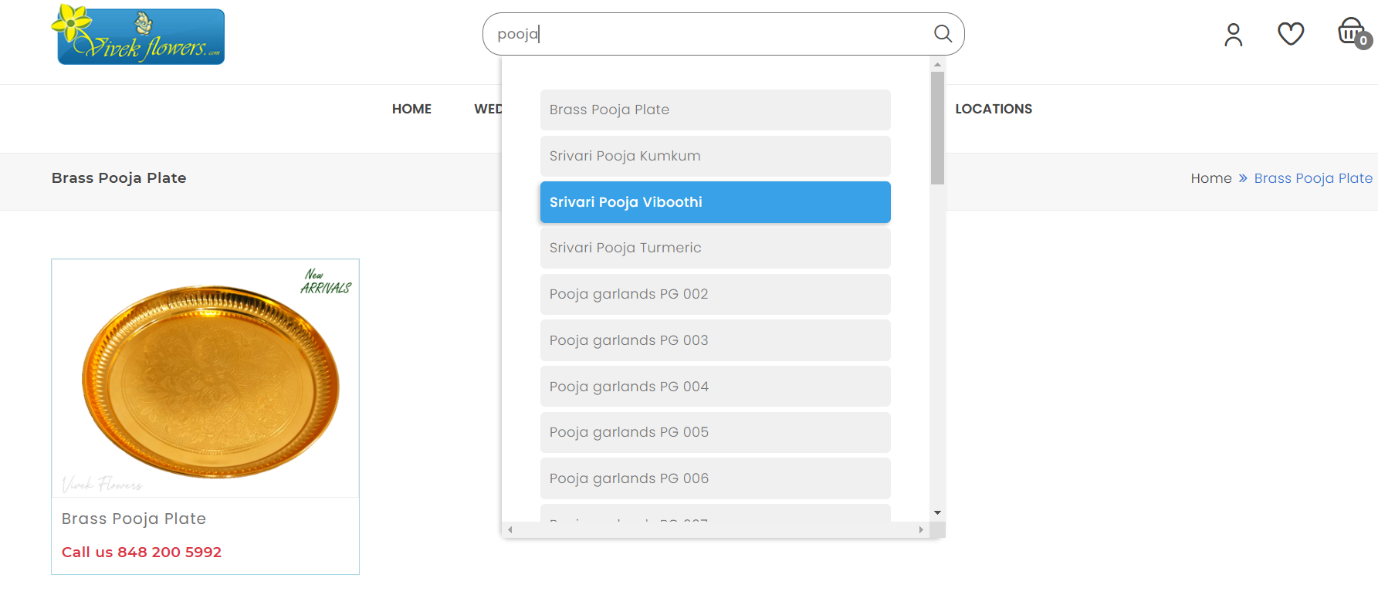


Figure 1.5.1.5 Search Bar

* **Account Page:**
  + Users can view and edit their personal information.
  + Order history and account details are accessible.

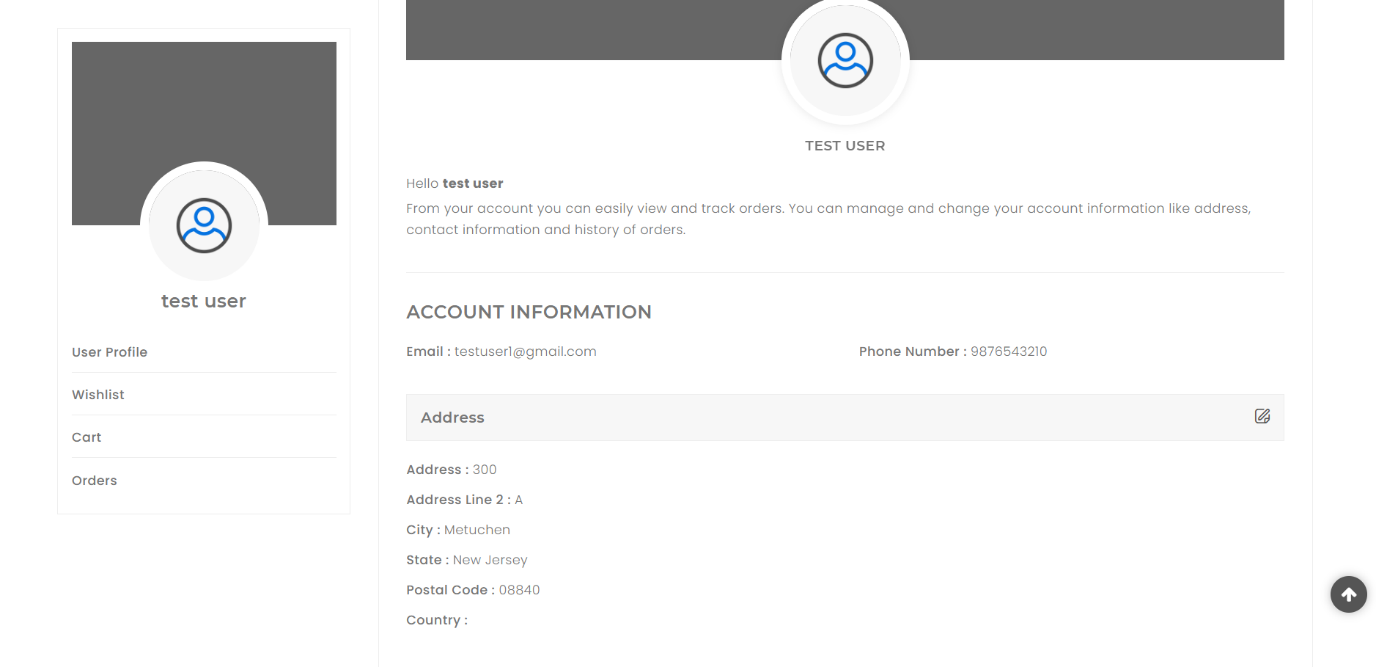


Figure 1.5.1.6 Account Page

* **Wish list Page:**
  + Users can save products for future purchases.
  + Option to move items from Wish list to Cart.

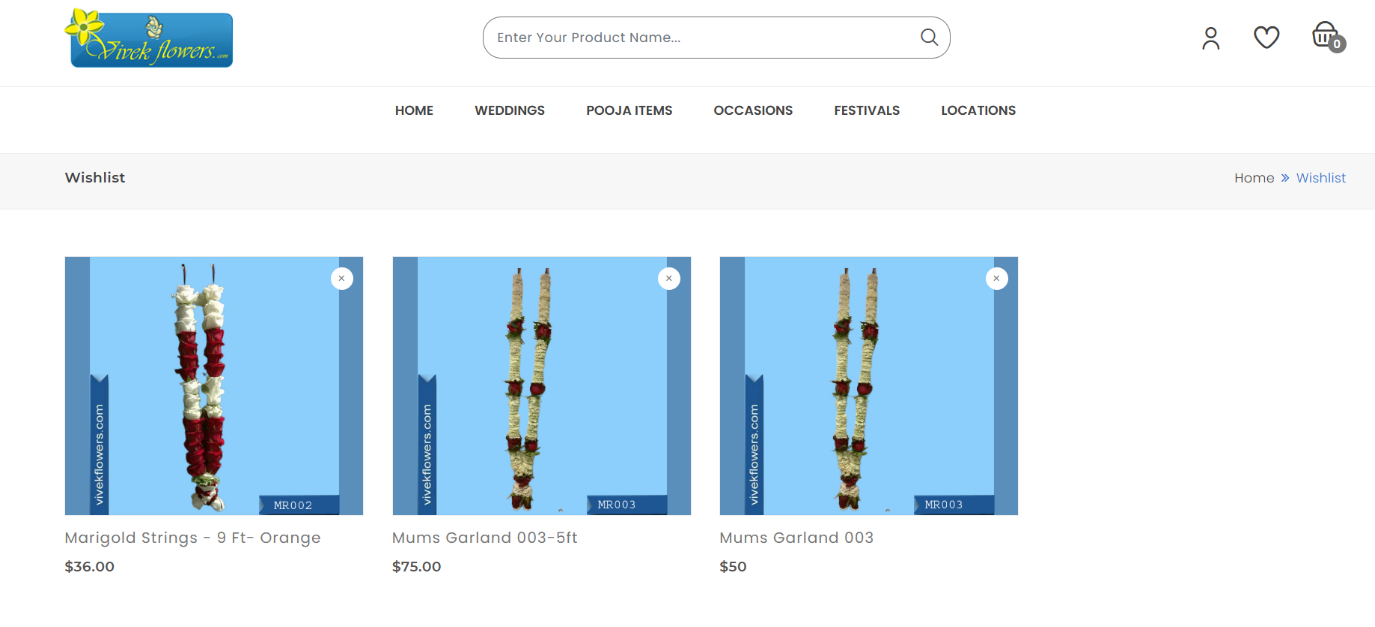


Figure 1.5.1.7 Wishlist Page

* **Orders Page :**
  + Users can track the status of their orders.
  + Order details, including invoice and payment status, are available.

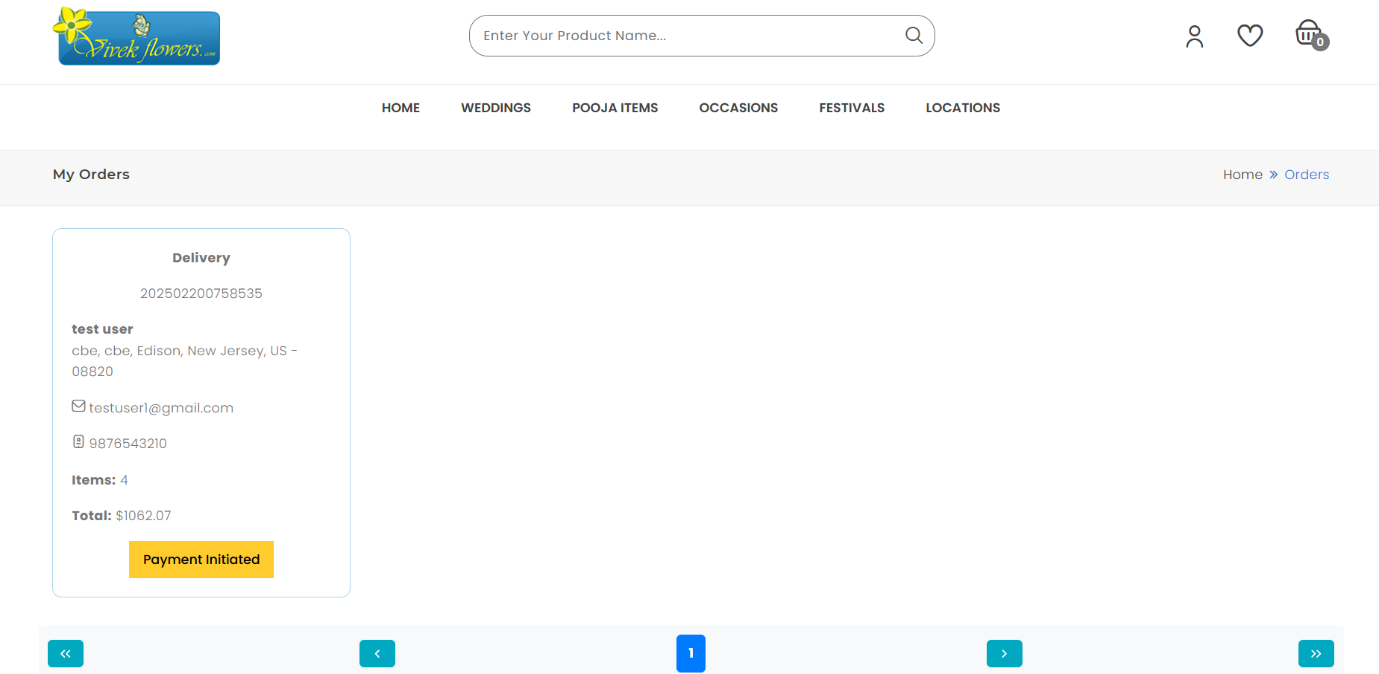


Figure 1.5.1.8 Order Page

* **Cart & Checkout:**
  + Products can be reviewed before checkout.
  + Secure online payment integration via PayPal.

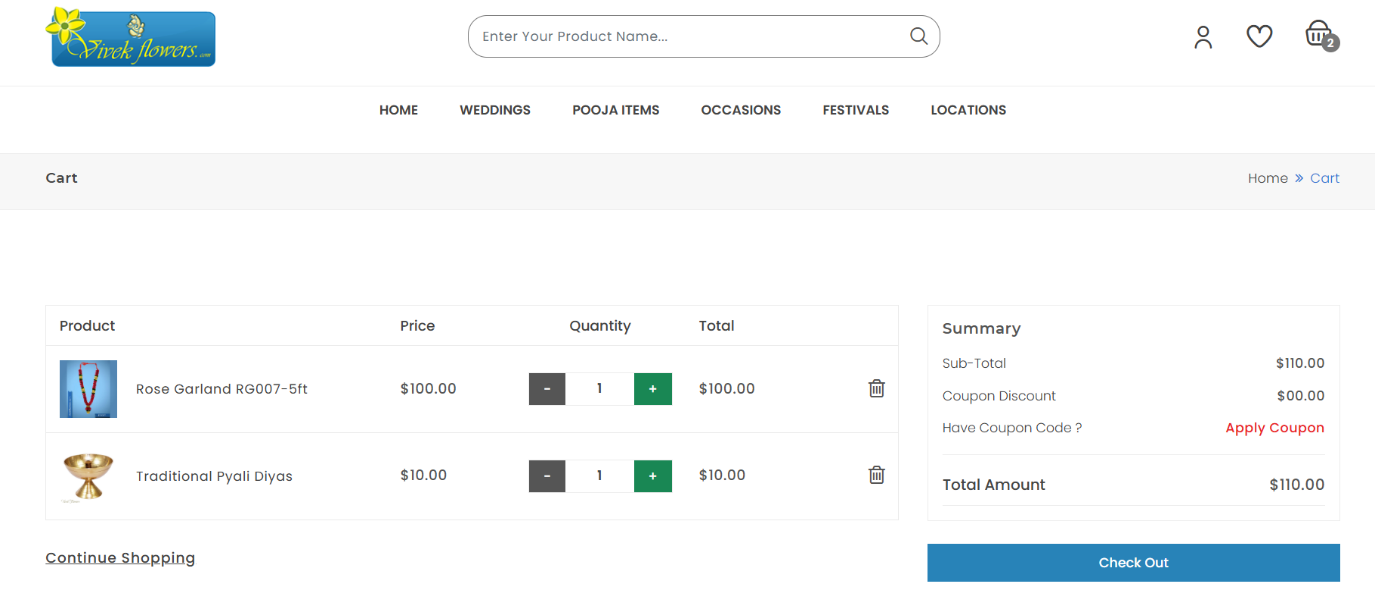


Figure 1.5.1.9 Cart Page

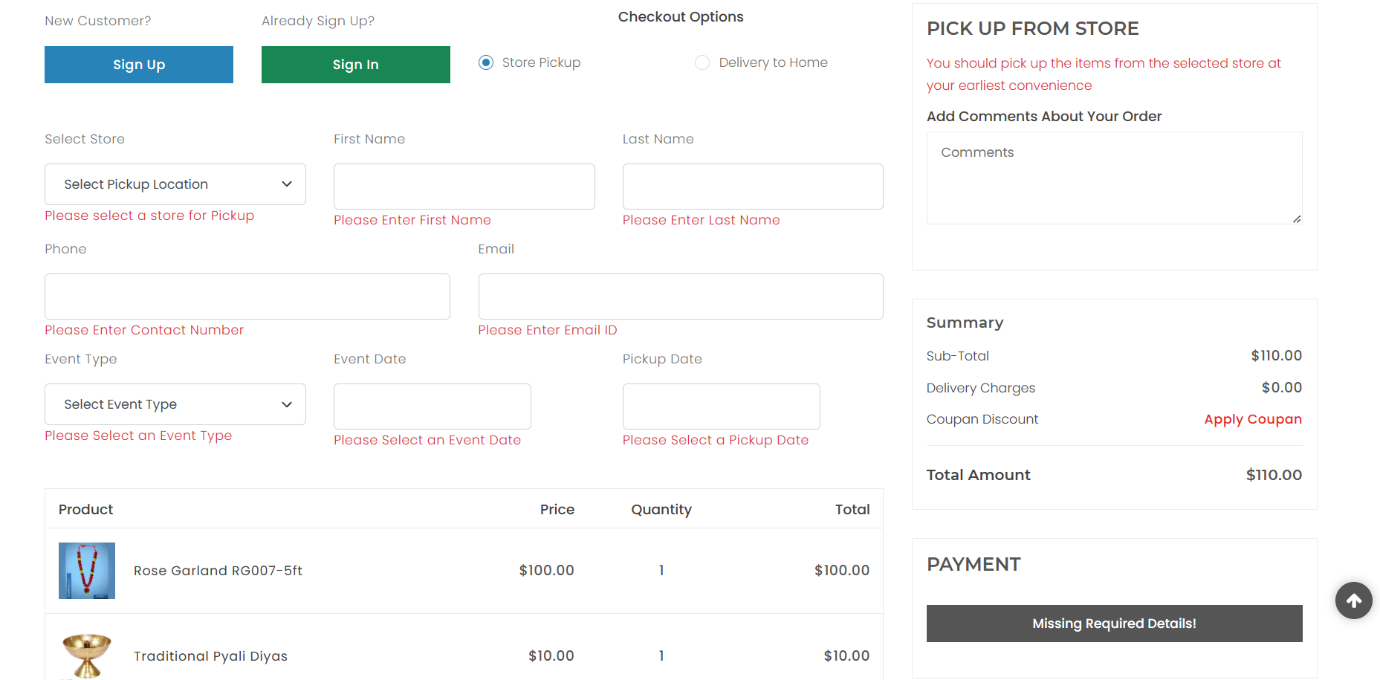


Figure 1.5.1.10 Checkout Page

#### ****1.5.2 Admin Panel:****

The admin panel provides full control over the website’s management, including:

**Dashboard:**

* + Displays key reports and analytics related to sales, orders, and user activity.

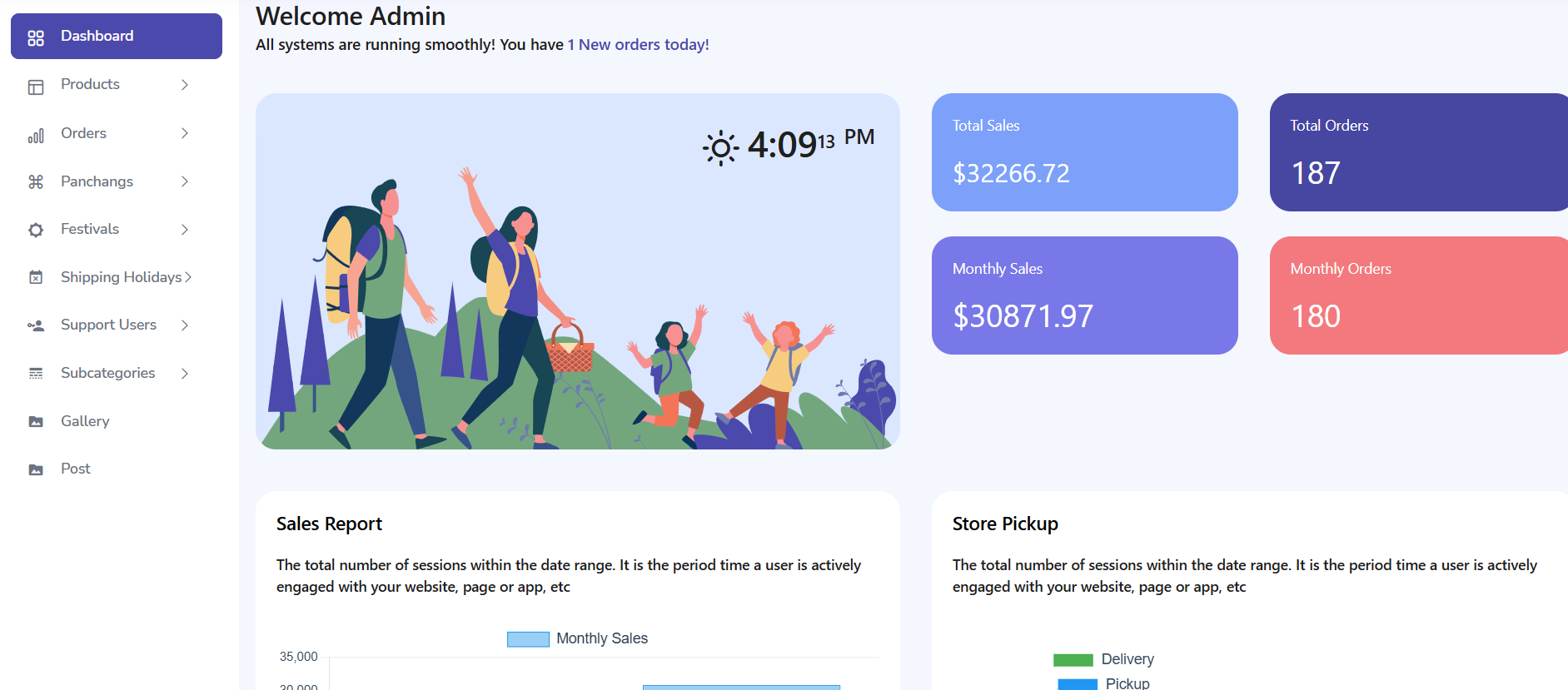


Figure 1.5.2.1 Admin Dashboard



Figure 1.5.2.2 Admin Dashboard

* **Product Management:**
  + Add, edit, change visibility, or remove products and product variations.
  + Integrated search functionality for quick product lookup.

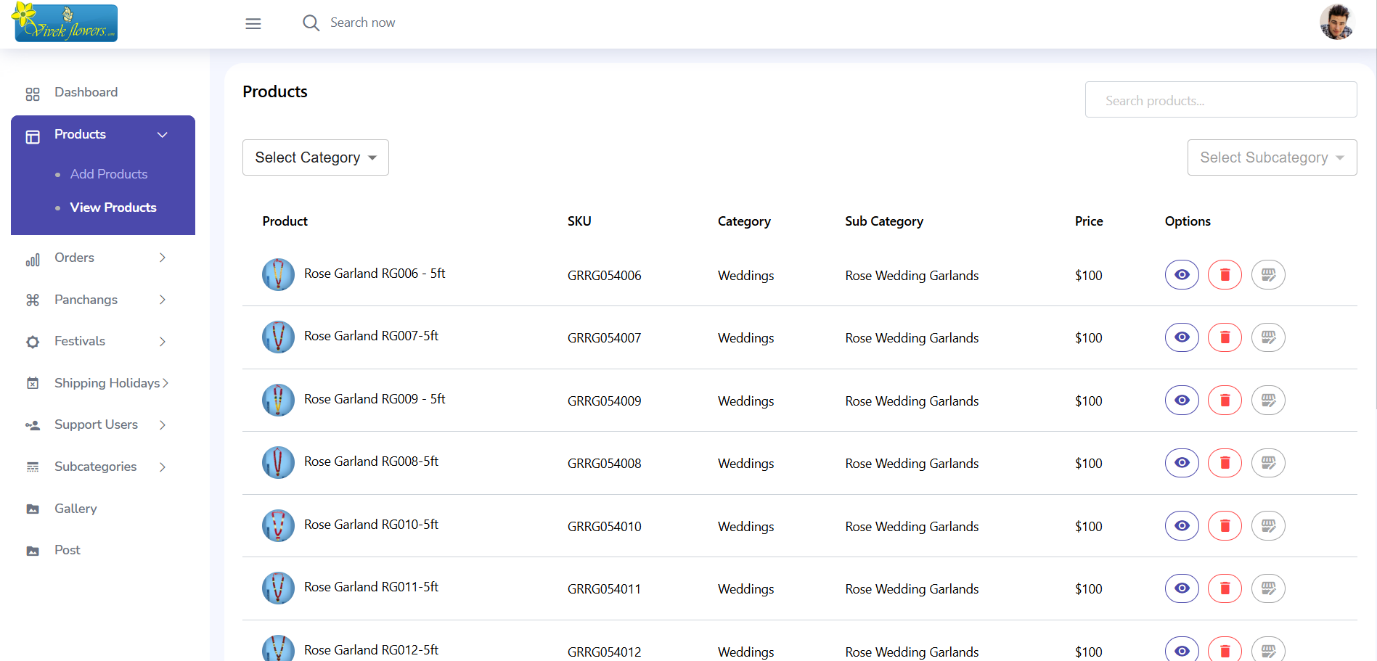


Figure 1.5.2.3 Product Page

* **Orders Management:**
  + View all orders with search functionality.
  + Update order details, including delivery status, comments, and delivery date.

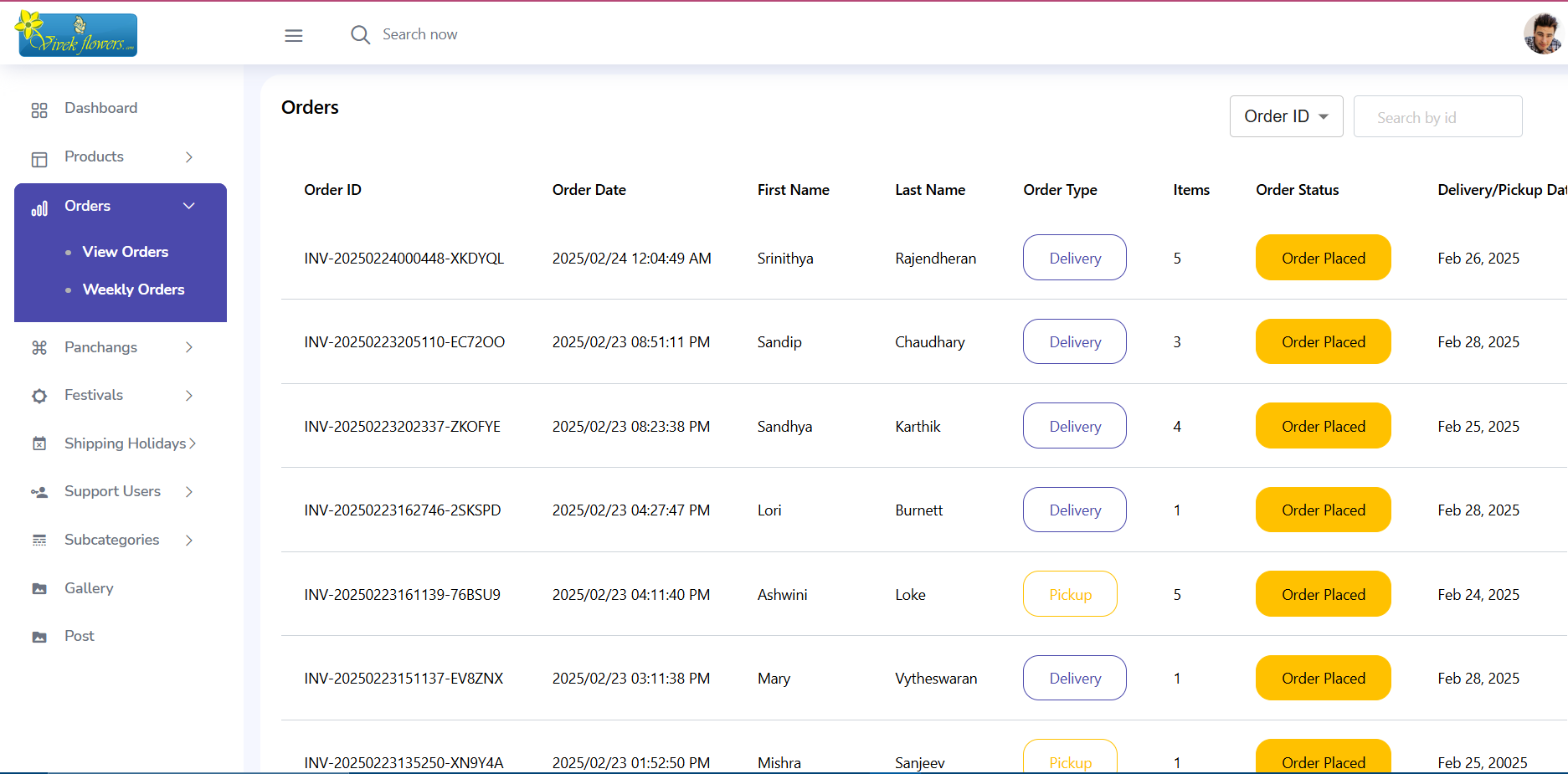


Figure 1.5.2.4 Orders Page

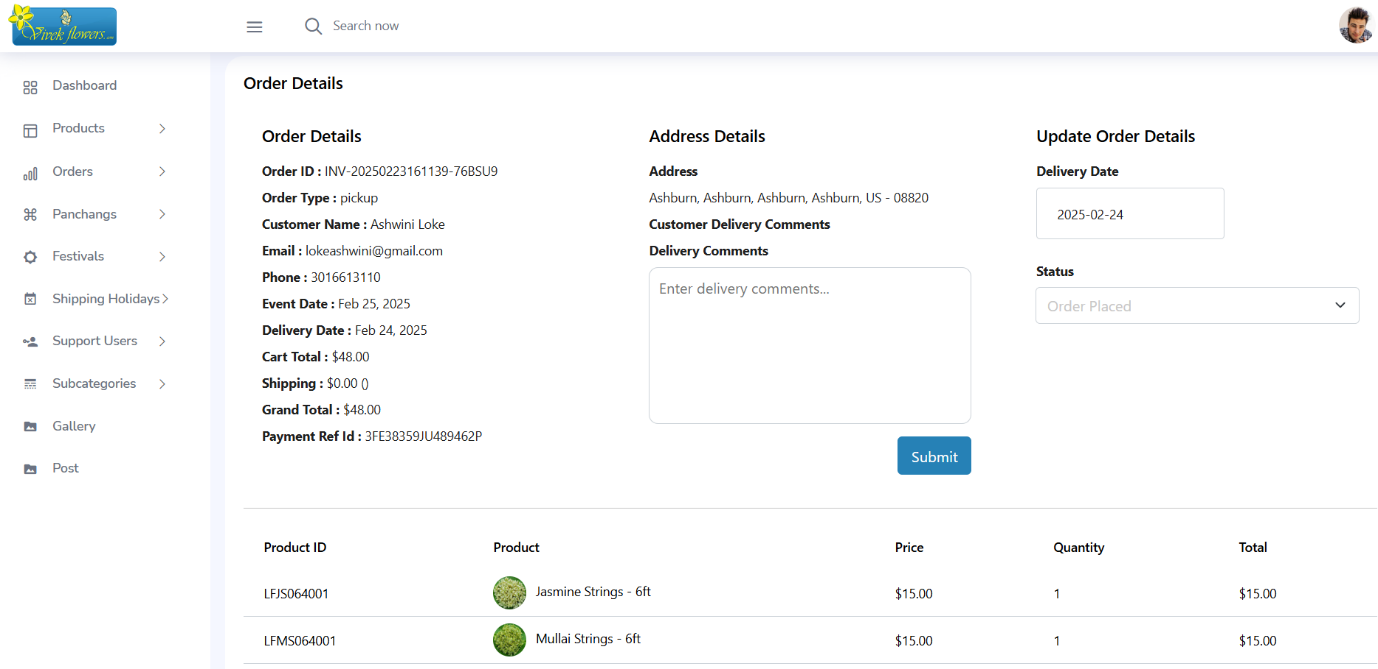


Figure 1.5.2.5 Order Details Page

* **Pangangs (Special Offerings) Management:**
  + Add, edit, and delete Pangangs.
* **Festival, Shipping Holidays, and Posts Management:**
  + Separate pages for adding, editing, and deleting related content.

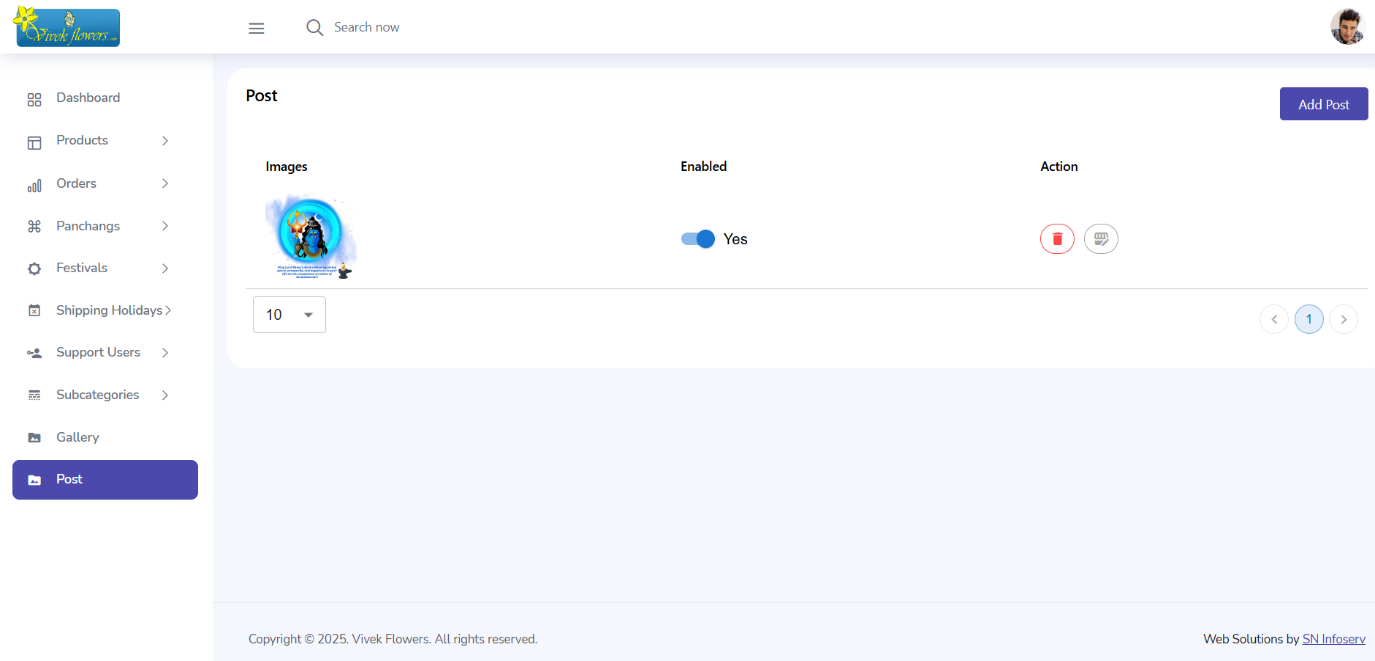


Figure 1.5.2.6 Post Page

* **Gallery Management:**
  + Add, edit, delete, and enable/disable sliders for the homepage.
  + Adjust the display order of sliders.

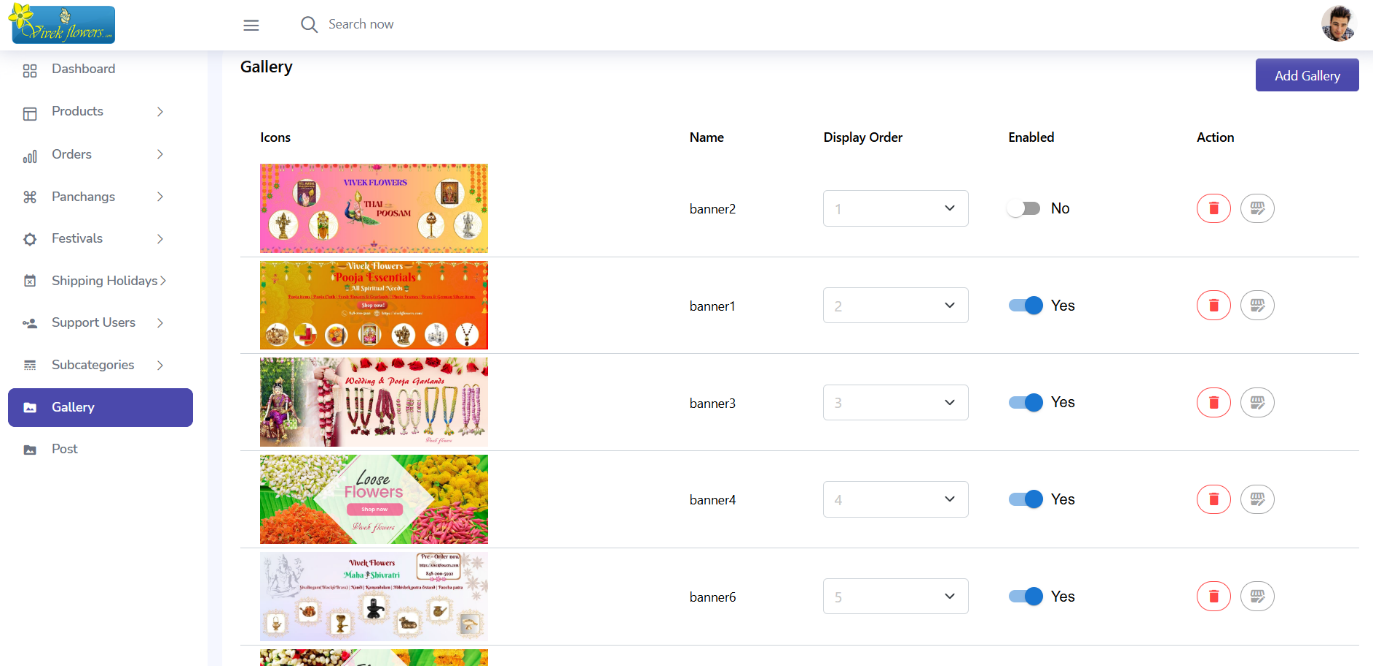


Figure 1.5.2.7 Gallery Page

* **Subcategory Management:**
  + Add, edit, delete, and search for subcategories.

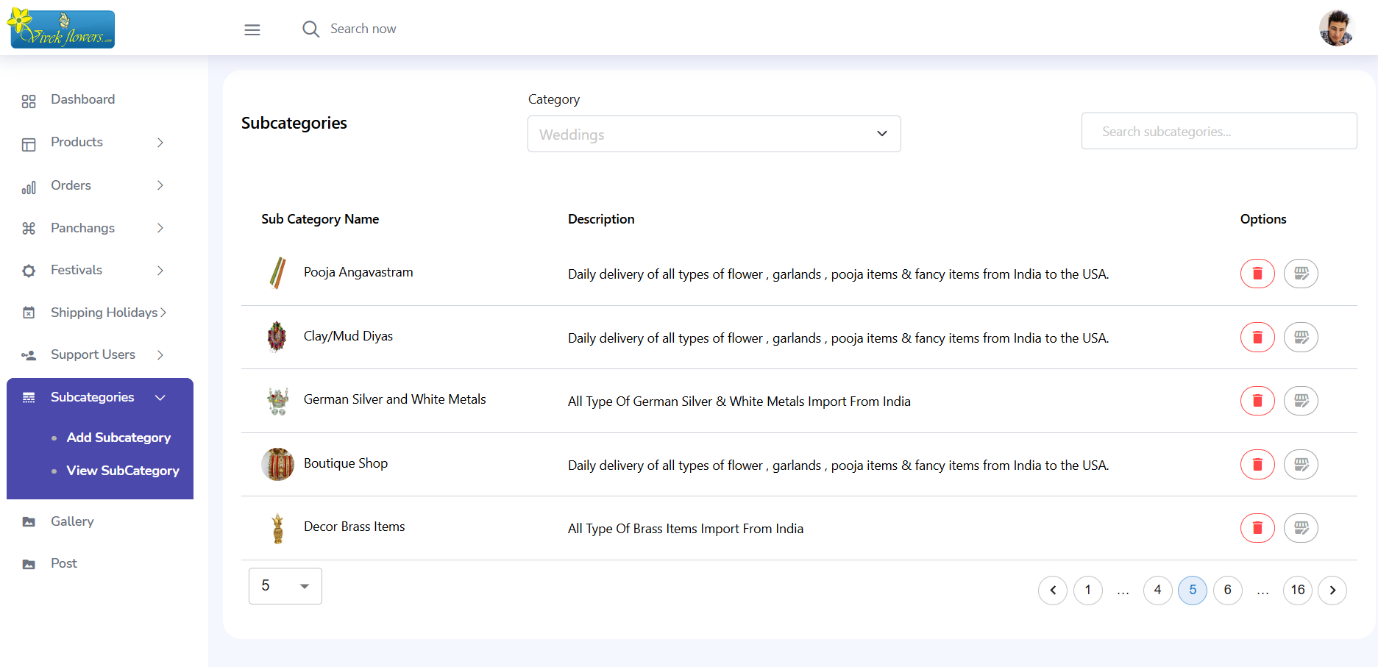


Figure 1.5.2.8 Subcategory Page

* **Support User Management:**
  + Add, edit, and delete support users who assist in website operations.

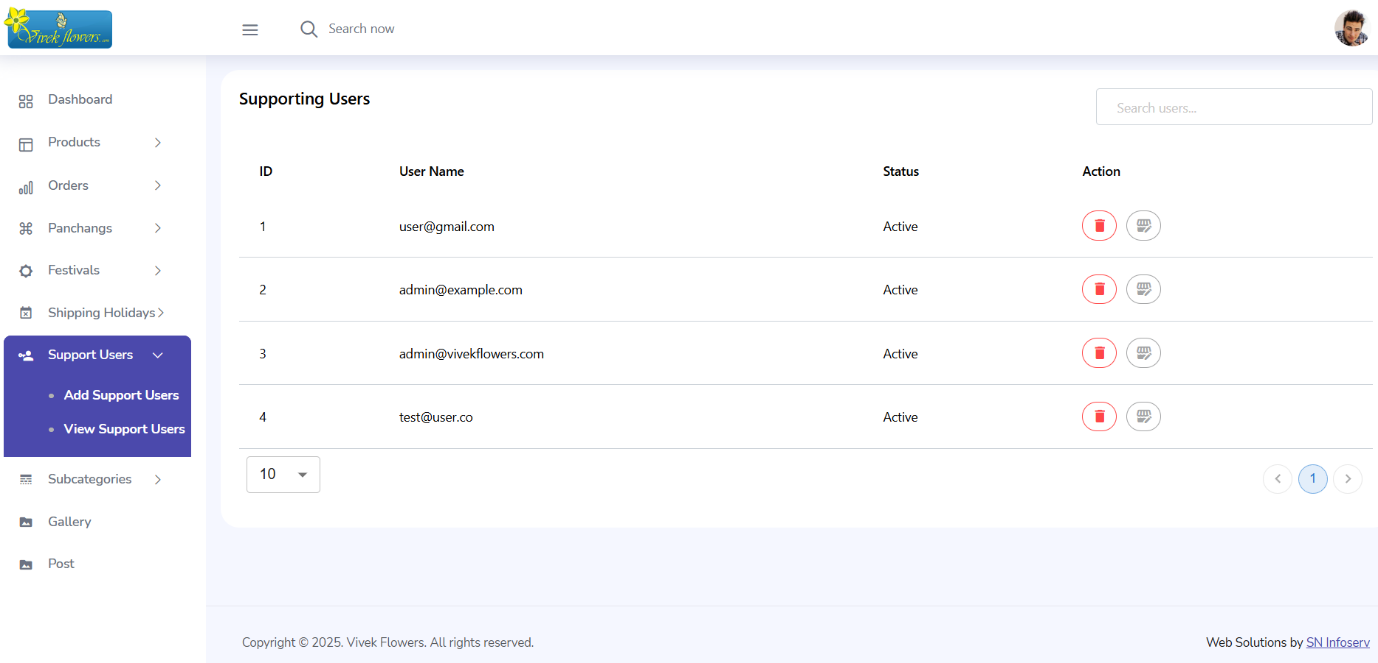


Figure 1.5.2.9 Support User Page

* 1. **Description of Web Technologies**

**1. React.js**

* **Type:** Frontend JavaScript Library
* **Overview**: React.js is an open-source JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create reusable UI components that manage their own state.

**Key Features:**

* **Virtual DOM**: Improves performance by minimizing direct manipulation of the actual DOM, resulting in faster rendering.
* **Component-Based Architecture**: Encourages the development of encapsulated components that can manage their own state and logic.
* **Unidirectional Data Flow**: Simplifies data management by ensuring that data flows in a single direction, enhancing the predictability of the application.
* **Use in the Project:** React.js is employed to build the user interface of the application, allowing users to interact with the product categories, manage their cart, and navigate through different pages seamlessly.

**2. Flask**

* **Type:** Python Web Framework
* **Overview:** Flask is a lightweight and flexible micro-framework for Python that allows rapid web development. It follows the WSGI standard and is highly extensible.
* **Key Features:**
  + **Lightweight & Modular:** Provides only the essential features, making it fast and easy to customize.
  + **Built-in Development Server:** Helps in testing applications locally.
  + **RESTful API Support:** Ideal for creating backend APIs to interact with the frontend.
* **Use in the Project:**
  + Acts as the backend framework to handle API requests, authentication, product management, orders, and user profiles.
  + Implements JWT authentication for secure user sessions.
  + Manages interactions with the MySQL database using pymysql.

**3. MySQL**

* **Type:** Relational Database Management System (RDBMS)
* **Overview:** MySQL is an open-source relational database system widely used for structured data storage and retrieval. XAMPP provides a local development environment to run MySQL along with Apache.
* **Key Features:**
* **Relational Data Storage:** Stores structured data in tables, ensuring data integrity.
* **ACID Compliance:** Ensures reliable transactions and data consistency.
* **High Performance & Scalability:** Handles complex queries efficiently.
* **Use in the Project:**
  + Stores user data, product information, order details, and transaction history.
  + Allows querying of product details, order management, and user authentication records.

**4. Bootstrap**

* **Type:** Frontend CSS Framework
* **Overview:** Bootstrap is a popular open-source framework used for building responsive and mobile-first web applications. It provides pre-designed CSS styles, components, and JavaScript plugins to speed up UI development.
* **Key Features:**
* Responsive Grid System – Uses a 12-column grid for flexible layouts across different screen sizes.
* Pre-styled Components – Includes ready-to-use UI components like buttons, forms, modals, and navigation bars.
* Customizable – Allows easy customization using SCSS variables and themes.  
  Cross-Browser Compatibility – Works seamlessly across modern web browsers.
* JavaScript Plugins – Provides built-in components like carousels, tooltips, and modals without needing extra coding.
* **Use in the Project:**
* Bootstrap is used to design a user-friendly and responsive UI for the website.
* Ensures proper alignment of elements like the header, product cards, buttons, forms, and navigation menus.
* Bootstrap’s grid system helps create an adaptive layout for different devices (desktop, tablet, mobile).
* Components like modals, alerts, and buttons are styled using Bootstrap’s pre-built styles.

1. **System Environment**

Start

Cart

Location

Category

Search

Home

Log in/Sign up

View Cart

yes No

Select category

Checkout

wishlist

View Product

Payment

profile

Log in/Sign up

yes No

View orders

End

Logout

Add to Cart

Add wish list

**System Flow Diagram (User panel)**

Add

Edit

View

Product

View

Orders

Update

Add

Edit

View

Panchangs

Add

Festivals

Edit

View

Log in

Start

Add

Edit

View

Shipping Holidays

Add

Support User

Edit

View

Add

View

Subcategory

Edit

Add

Gallery

Edit

View

Add

Post

Edit

View

**System Flow Diagram (Admin Panel)**

**Operating System**

The project is platform-independent and can be developed and deployed on various operating systems, including Windows, macOS, or Linux, ensuring broad compatibility. Web Server. A web server is required to host the project's web pages and provide access to users. You can choose from a range of web server options such as Apache, Nginx, IIS, or any other server that supports the specific programming language and framework used in the project.

**Front-End Technologies**

* HTML for structuring web content.
* Bootstrap CSS for responsive and modern styling.
* ReactJS for building reusable user interface components and managing state effectively.

**Back-End Technologies**

* Flask is used for building the server-side logic and handling requests efficiently.
* MySQL serves as the database for storing product details, user information, and orders.

**Version Control System**

* To facilitate collaborative development and track code changes effectively, the project can be managed using Git, a widely adopted version control system (VCS). Git helps maintain code integrity, manage branches, and enable efficient collaboration among team members.

This system environment ensures that your project is accessible and adaptable across different operating systems, provides the necessary tools and technologies for MERN Stack development using Mongo DB, Express JS, React JS, Node JS and emphasizes efficient collaboration through version control using Git.

**2.0.1Hardware Configuration**

The required system configuration for software development and implementation for this project is as follows:

* **Processor**: AMD PRO A4-4350B, 2.50GHz
* **Hard Disk**: 500 GB
* **RAM**: 8GB
* **Monitor**: 15" VGA Colour

**2.0.2 Software Configuration**

For this application project, the basic software requirements include the operating system, front-end tools for user interaction, and back-end tools for data storage and processing:

* **Operating System**: Windows (Versions 7, 8, 8.1, 10)
* **Front-End and Back-End:**
  + React JS
  + Python(Flask)
  + CSS
  + MySQL

**2.1 Software Features**

**System Specification - Windows 10:**

* Windows 10 is the selected operating system for this project.
* It is the successor to Windows 8.1 and was developed under the project name "Threshold."
* Windows 10 is the last major version of the Windows operating system according to Microsoft.
* Microsoft offered a free upgrade to Windows 10 for users of Windows 7 SP1 and Windows 8.1 within the first year since its release.
* Windows 10 is designed for various types of devices, including desktop computers, laptops, tablets, and phones.
* The operating system is suitable for a wide range of applications and offers different editions, including:
  + Windows 10 Home
  + Windows 10 Pro
  + Windows 10 Enterprise
  + Windows 10 Education
  + Windows 10 Mobile
  + Windows 10 Mobile Enterprise
  + Windows 10 IoT Core
  + Windows Server 2016
  + Windows 10 for XBOX One

1. **SYSTEM DESIGN**

The **Vivek Flowers** e-commerce application is designed to provide a seamless shopping experience, ensuring efficiency, scalability, and maintainability. This section outlines the system architecture, including input design, database design, and user interface design.

#### 3.1 Input Design

Input design is crucial for ensuring that user data is collected efficiently and accurately. In the Vivek Flowers application, user inputs are gathered through various forms across different modules. Below are the key aspects of input design:

**3.1.1 User Panel**

1. **User Registration Form:**
   * Fields: First Name, Last Name, Email, Password, Phone No, Address, Address Line 2, City, Postal Code, State.
   * Validation: Email format validation, password strength validation (minimum 8 characters, including uppercase, lowercase, numbers, and special characters), and required fields.

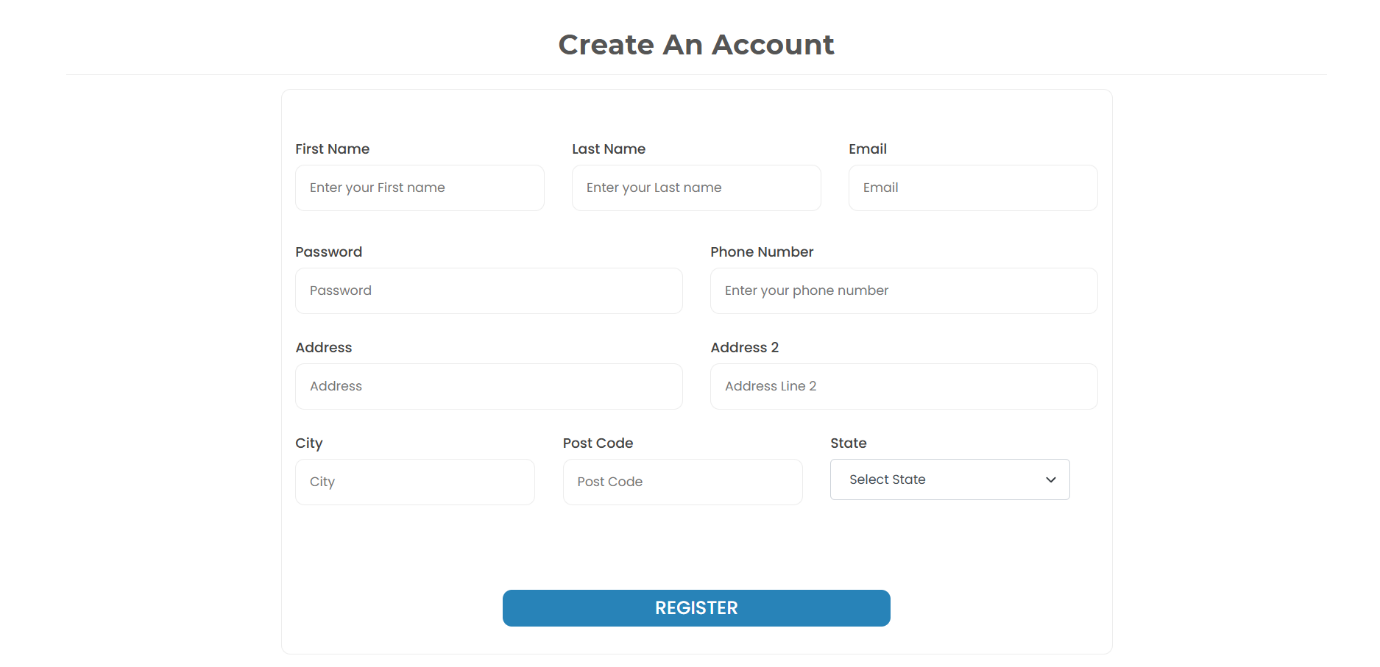


Figure 3.1.1.1 Register Page

1. **Login Form:**
   * Fields: Email, Password.
   * Validation: Ensures that both fields are filled and the email format is valid.

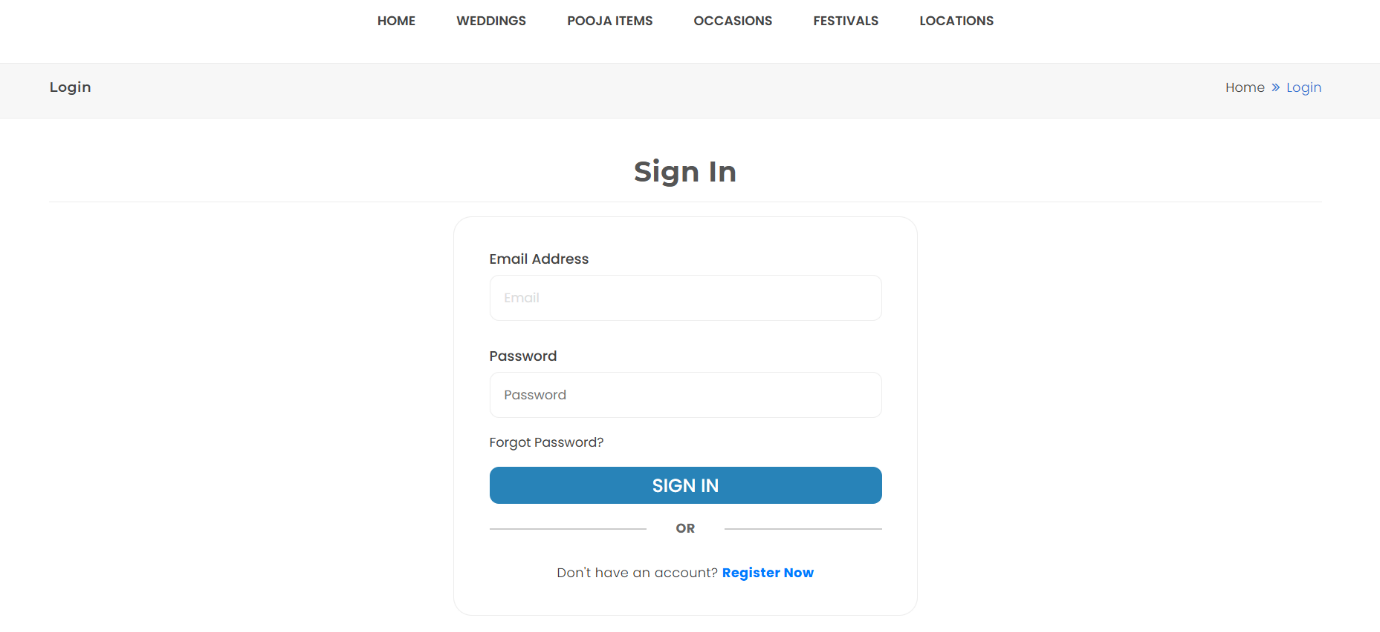


Figure 3.1.1.2 Login Page

1. **Checkout Page Pickup Form:** 
   * Fields: Pick up Location, First Name, Last Name, Email, Phone No, Event Type, Event Date, Pick up Date.
   * Validation: Email format validation and required fields.

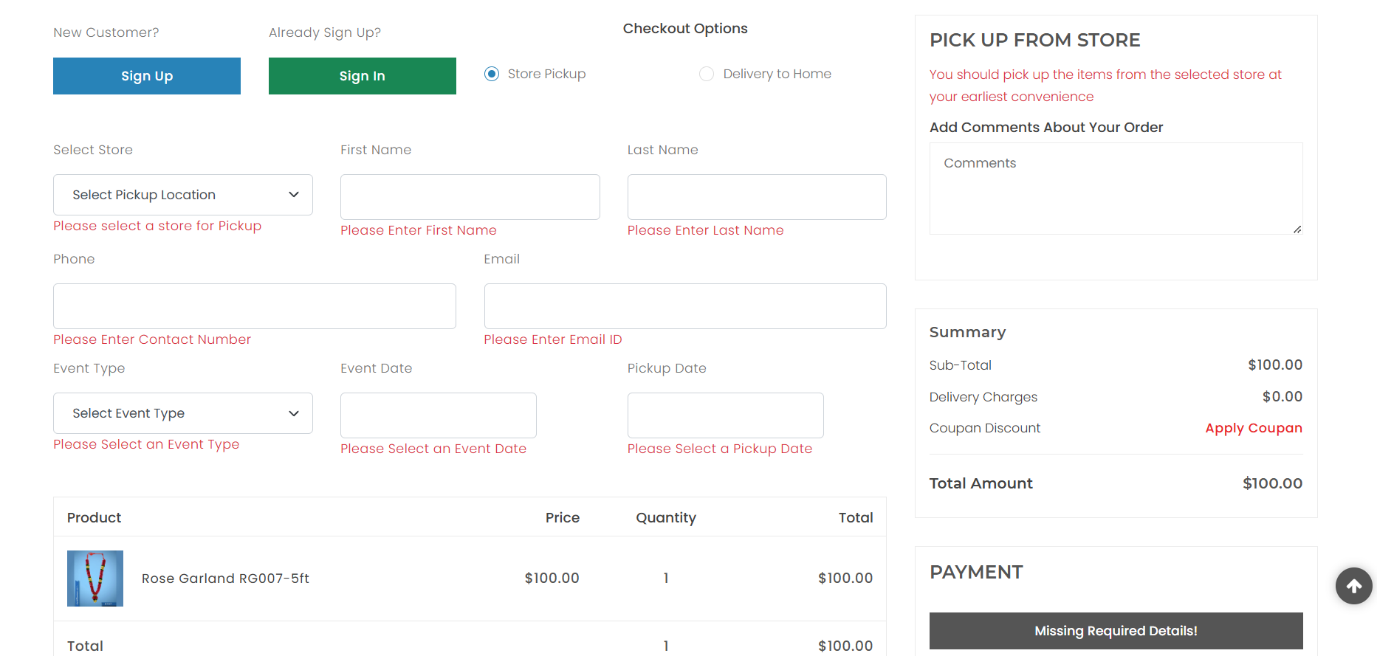


Figure 3.1.1.3 Checkout Page (Pickup)

1. **Checkout Page Delivery Form:**
   * Fields: First Name, Last Name, Email, Phone No, Address, Address Line 2, City, Zip Code.
   * Validation: Email format validation, zip code validation and required fields.

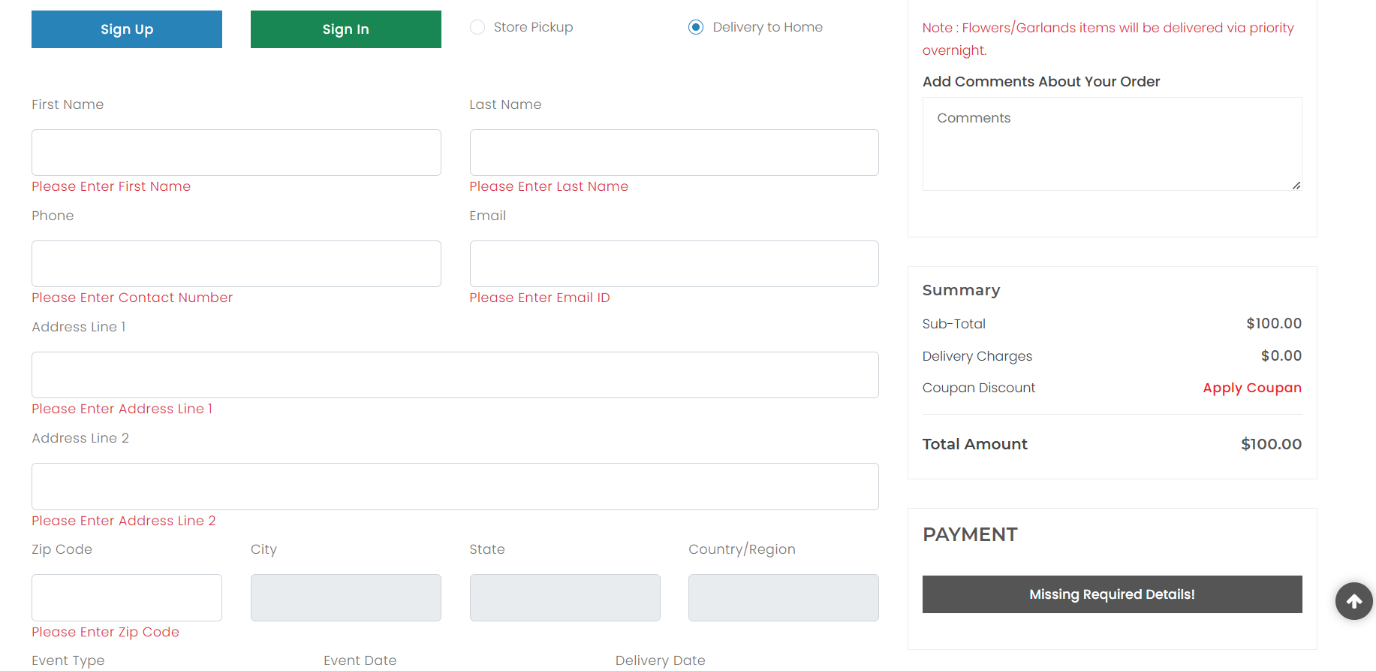


Figure 3.1.1.4 Checkout Page(Delivery)

**3.1.2 Admin Panel:**

1. **Product Addition Form :**
   * Fields: Product Id, SKU, Product Name, Description, Price, Category, Subcategory, Images, Weight, With Ice Weight.
   * Validation: Checks for required fields, numeric values for price, and valid image file types.

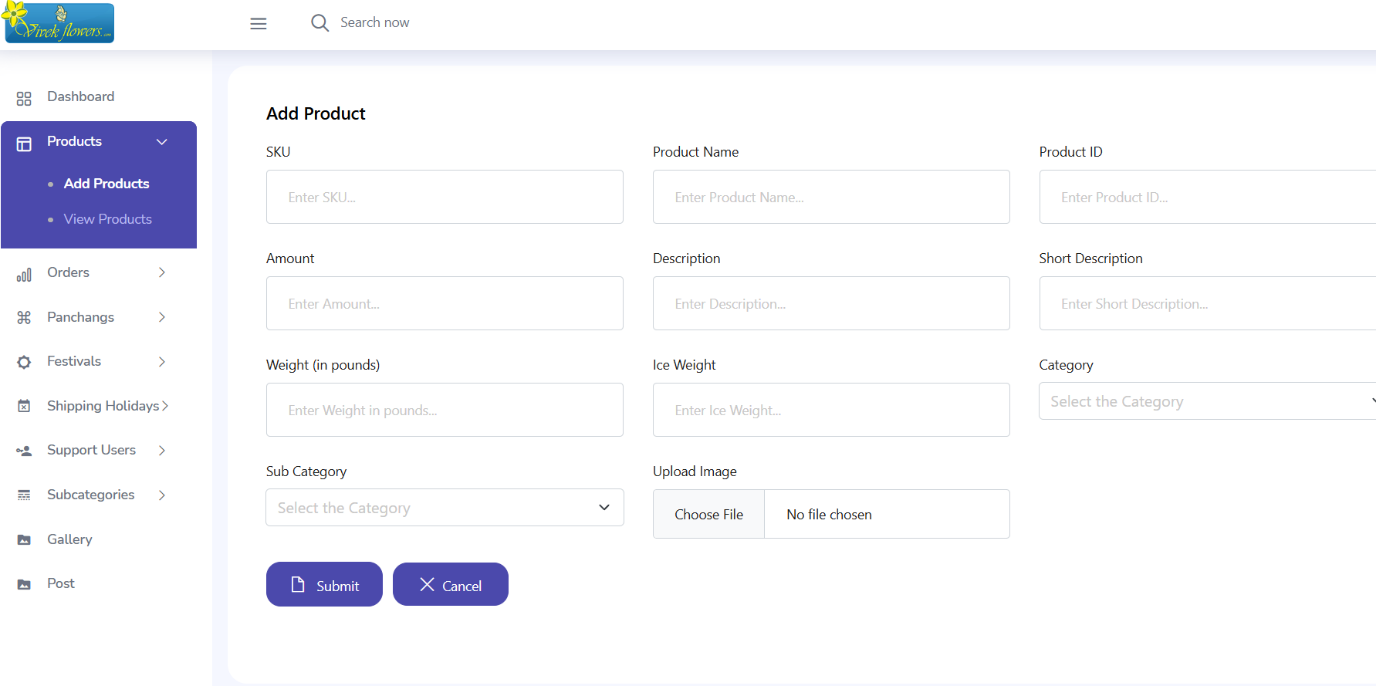


Figure 3.1.2.1 Add Product Page

1. **Panchang Adding Form:**
   * Fields: Date, Sunrise, Sunset, Rahu Kala, Yamaganda, Tithi
   * Validation: Required fields.

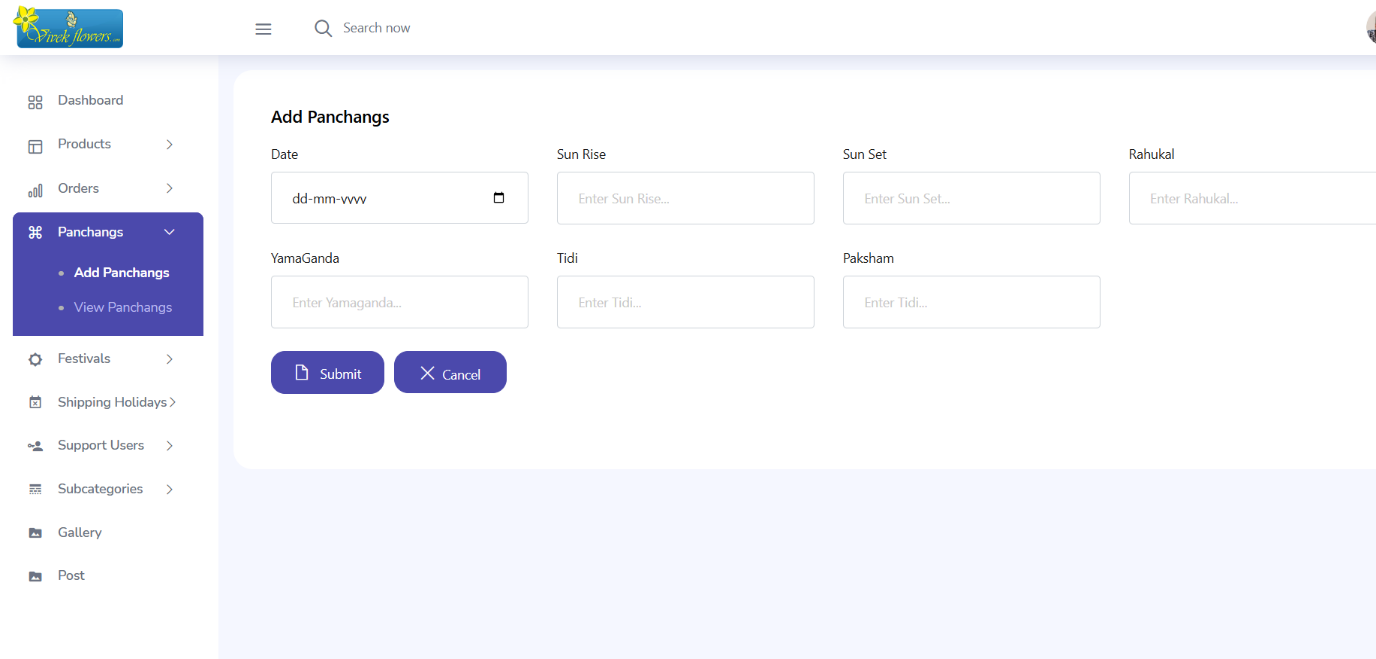


Figure 3.1.2.2 Add Panchangs Page

1. **Shipping Holidays**:
   * Fields: Date.

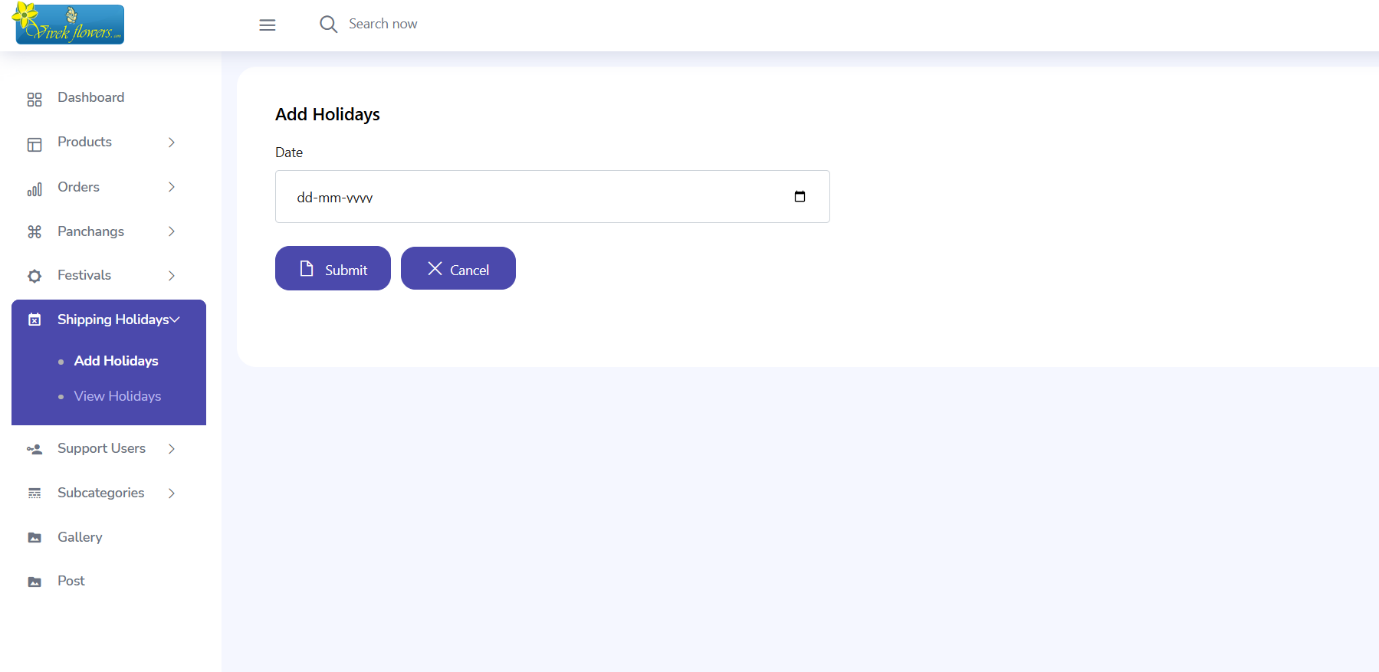


Figure 3.1.2.3 Add Holidays Page

1. **Support users:**
   * Fields: Email, Password.
   * Validation: Ensures that both fields are filled and the email format is valid.

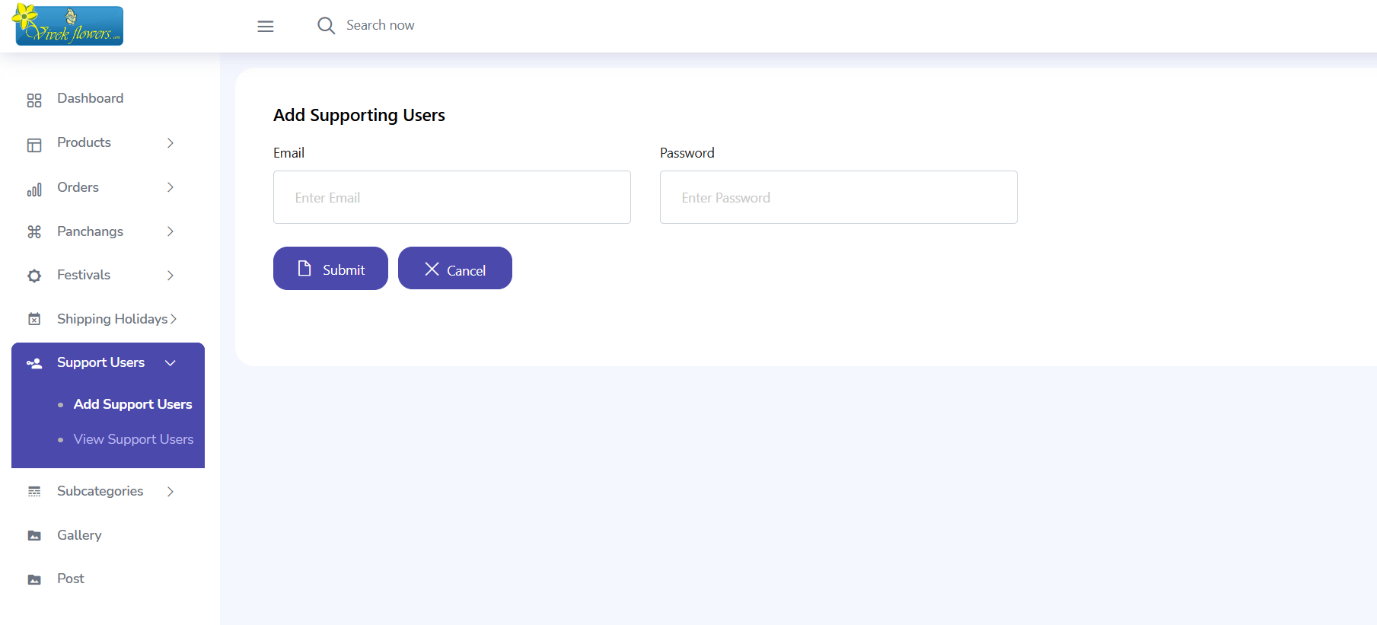


Figure 3.1.2.4 Add Support User Page

1. **Subcategories**:
   * Fields: Category, Subcategory Name, Description, Subcategory Image.
   * Validation: Checks for required fields and valid image file types.

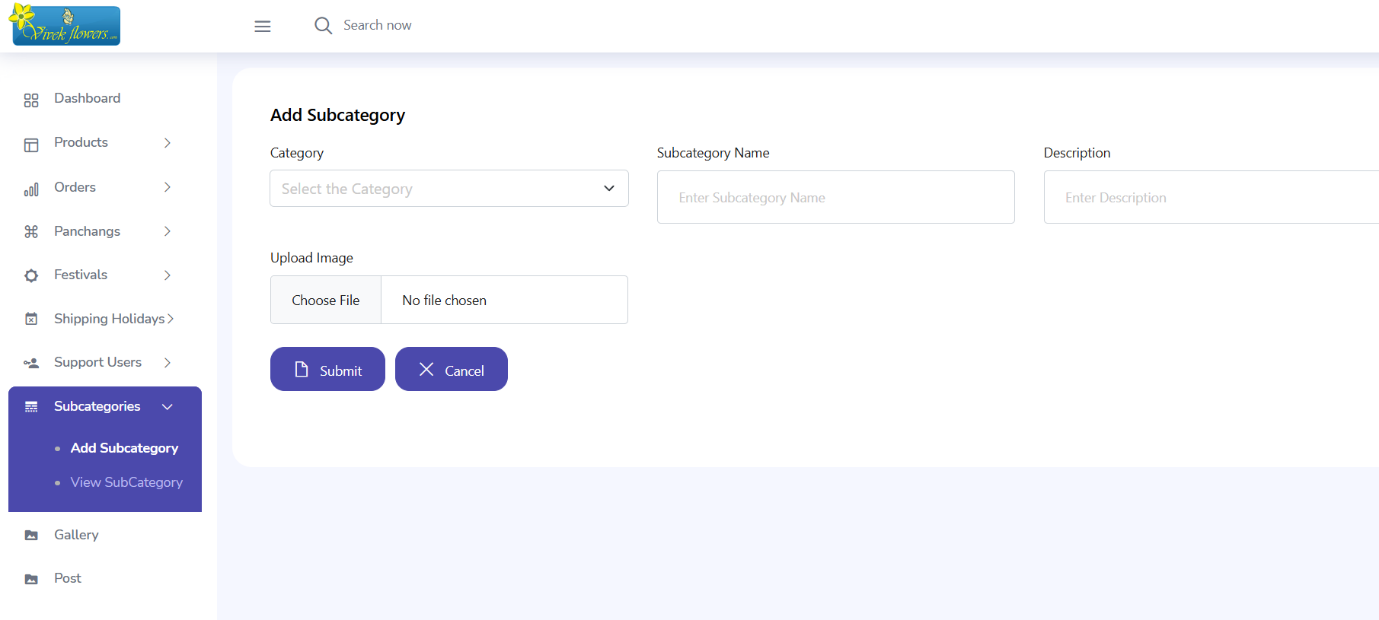


Figure 3.1.2.5 Add Subcategory Page

#### 3.2 Database Design

The database design for the **VivekFlowers** application is structured to efficiently store user, product, order, and transaction information. **MySQL**, a relational database, is used to manage structured data, ensuring integrity and optimized query performance.

1. **Collections**:

* **Users Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), first\_name (VARCHAR), last\_name (VARCHAR), email (VARCHAR, UNIQUE), password (VARCHAR), phone (VARCHAR), status (ENUM), created\_at (TIMESTAMP).
* **Customers Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), first\_name (VARCHAR), last\_name (VARCHAR), email (VARCHAR, UNIQUE), password (VARCHAR), phone (VARCHAR), address (TEXT), city (VARCHAR), state (VARCHAR), postal\_code (VARCHAR), created\_at (TIMESTAMP).
* **Address Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), cust\_id (INT, FOREIGN KEY REFERENCES customers(id)), address\_line1 (TEXT), address\_line2 (TEXT), city (VARCHAR), state (VARCHAR), postal\_code (VARCHAR), country (VARCHAR).
* **Products Table:**
  + **Fields**: p\_id (INT, AUTO\_INCREMENT, PRIMARY KEY), SKU (VARCHAR, UNIQUE), productName (VARCHAR), description (TEXT), short\_description (TEXT), amount (DECIMAL), category\_id (INT), subcategory\_id (INT), image\_url (TEXT), visibility (BOOLEAN), weight (DECIMAL), weightWithIce (DECIMAL).
* **Product Variations Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), product\_id (INT, FOREIGN KEY REFERENCES products(p\_id)), variation\_name (VARCHAR), variation\_amount (DECIMAL), weight (DECIMAL), withIceWeight (DECIMAL), SKU (VARCHAR), visibility (BOOLEAN).
* **Wishlist Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), cust\_id (INT, FOREIGN KEY REFERENCES customers(id)), product\_id (INT, FOREIGN KEY REFERENCES products(p\_id)), variation\_id (INT, FOREIGN KEY REFERENCES product\_variation(id)).
* **Orders Table:**
  + **Fields:** orderId (INT, AUTO\_INCREMENT, PRIMARY KEY), orderType (VARCHAR), custId (INT, FOREIGN KEY REFERENCES customers(id)), invNo (VARCHAR, UNIQUE), total (DECIMAL), shipping (DECIMAL), grandTotal (DECIMAL), shippingMode (VARCHAR), status (VARCHAR), orderPlaced (TIMESTAMP), paymentStatus (VARCHAR), paymentId (VARCHAR), deliveryDate (DATE), fName (VARCHAR), lName (VARCHAR), phone (VARCHAR), email (VARCHAR), address1 (TEXT), address2 (TEXT), city (VARCHAR), state (VARCHAR), zip (VARCHAR), country (VARCHAR).
* **Order Details Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), order\_id (INT, FOREIGN KEY REFERENCES orders(orderId)), invoiceNo (VARCHAR, FOREIGN KEY REFERENCES orders(invNo)), productId (INT, FOREIGN KEY REFERENCES products(p\_id)), qty (INT), price (DECIMAL), name (VARCHAR), img (TEXT).
* **Payments Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), payer\_email (VARCHAR), payment\_status (VARCHAR), last\_transaction\_id (VARCHAR, UNIQUE).
* **Gallery Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), icon (TEXT), name (VARCHAR), displayOrder (INT), enable (BOOLEAN).
* **Image Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), image (TEXT), enable (BOOLEAN).
* **Panchangs Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), date (DATE), sunrise (TIME), sunset (TIME), rahukal (TIME), yamaganda (TIME), tidi (VARCHAR), paksham (VARCHAR).
* **Festivals Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), date (DATE), name (VARCHAR).
* **Shipping Holidays Table:**
  + **Fields:** id (INT, AUTO\_INCREMENT, PRIMARY KEY), date (DATE).

1. **Relationships**:
   * Each customer can have multiple addresses, forming a one-to-many relationship.
   * Each customer can place multiple orders, forming a one-to-many relationship.
   * Each order can include multiple products, forming a many-to-many relationship between the Orders and Products tables.
   * The Wishlist table creates a many-to-many relationship between Customers and Products.
   * The Product Variations table maintains different variations of a product under a one-to-many relationship with the Products table.
   * The Order Details table stores each product in an order, forming a one-to-many relationship between Orders and Order Details.
   * The Payments table tracks payment transactions, forming a one-to-one relationship with Orders.
2. **Indexing**:
   * Primary keys (id, orderId, p\_id) ensure uniqueness.
   * Indexes are created on frequently queried fields like email (in Customers and Users), category\_id (in Products), and invNo (in Orders) to improve query performance.
   * Foreign keys enforce referential integrity between related tables.

#### 3.3 Code Design

#### import React, { useEffect, useState } from "react";

#### import { useLocation, Link, useNavigate } from 'react-router-dom';

#### import axios from "axios";

#### import Header from "../components/Header";

#### import Footer from "../components/Footer";

#### import Flower from './flower';

#### import "./style.css";

#### const ProductCategories = () => {

#### const [products, setProducts] = useState([]);

#### const [subcategory\_id, setsubcategory\_id] = useState('');

#### const [loader, setLoader] = useState(true);

#### const location = useLocation();

#### const navigate = useNavigate();

#### const [selectedVariationOptions, setSelectedVariationOptions] = useState({});

#### console.log("product",products)

#### // console.log("selectedop",selectedVariationOptions)

#### const handleVariationOptionChange = (e, variationId, productId) => {

#### const selectedOptionId = parseInt(e.target.value);

#### // Find the product and the selected variation

#### const updatedProducts = products.map(product => {

#### if (product.p\_id === productId) {

#### const updatedVariations = product.variations.map(variation => {

#### if (variation.variation\_id === variationId) {

#### const selectedOption = variation.variation\_options.find(opt => opt.vOptionId === selectedOptionId);

#### return {

#### ...variation,

#### selectedOption: selectedOption,

#### };

#### }

#### return variation;

#### });

#### return {

#### ...product,

#### variations: updatedVariations,

#### };

#### }

#### return product;

#### });

#### setProducts(updatedProducts);

#### setSelectedVariationOptions((prev) => ({

#### ...prev,

#### [productId]: {

#### ...prev[productId],

#### [variationId]: selectedOptionId,

#### },

#### }));

#### };

#### useEffect(() => {

#### const intervalId = setInterval(() => {

#### setLoader(false);

#### }, 1500);

#### return () => clearInterval(intervalId);

#### }, []);

#### const getQueryParam = (param) => {

#### const urlParams = new URLSearchParams(location.search);

#### return urlParams.get(param);

#### };

#### useEffect(() => {

#### if (location.state && location.state.products) {

#### setProducts(location.state.products);

#### }

#### }, [location]);

#### useEffect(() => {

#### const category = getQueryParam('id');

#### setsubcategory\_id(category);

#### }, []);

#### const fetchProducts = async (cat\_id) => {

#### try {

#### const response = await axios.get(`http://127.0.0.1:5000/api/products-with-variations/subcategory?name=${cat\_id}`);

#### if (response.data) {

#### const updatedProducts = response.data.products.map((product) => ({

#### ...product,

#### quantity: 1,

#### selectedVariation: null,

#### }));

#### setProducts(updatedProducts);

#### setLoader(false);

#### }

#### } catch (error) {

#### console.error("Error fetching products:", error.message);

#### }

#### };

#### useEffect(() => {

#### const category = getQueryParam('id');

#### fetchProducts(category);

#### setsubcategory\_id(category);

#### }, [subcategory\_id]);

#### const handleIncreaseQty = (index) => {

#### const updatedProducts = [...products];

#### updatedProducts[index].quantity += 1;

#### setProducts(updatedProducts);

#### };

#### const handleDecreaseQty = (index) => {

#### const updatedProducts = [...products];

#### if (updatedProducts[index].quantity > 1) {

#### updatedProducts[index].quantity -= 1;

#### setProducts(updatedProducts);

#### }

#### };

#### const handleVariationChange = (productIndex, variationId) => {

#### const updatedProducts = [...products];

#### const selectedProduct = updatedProducts[productIndex];

#### const selectedVariation = selectedProduct.variations.find(variation => variation.variation\_id === variationId);

#### updatedProducts[productIndex].selectedVariation = selectedVariation;

#### setProducts(updatedProducts);

#### };

#### const addCart = (pId, qty) => {

#### const selectedProduct = products.find(item => item.p\_id === pId);

#### const variation = selectedProduct.selectedVariation;

#### const cartKey = selectedProduct.category\_id === 2 && (selectedProduct.subcategory\_id !== 37 || selectedProduct.subcategory\_id !== 3) ? 'poojaItems' : 'cartItems';

#### let pName = selectedProduct.productName;

#### let price = parseFloat(selectedProduct.amount || 0);

#### let weight = parseFloat(selectedProduct.weight || 0);

#### let iceWeight = parseFloat(selectedProduct.weightWithIce || 0);

#### 

#### if (selectedProduct.variations && selectedProduct.variations.some(v => v.selectedOption)) {

#### const selectedOptions = selectedProduct.variations

#### .filter(v => v.selectedOption)

#### .map(v => v.selectedOption);

#### pName = selectedProduct.productName + " - " + selectedOptions.map(opt => opt.variationName).join(", ");

#### price = selectedOptions.reduce((sum, opt) => sum + parseFloat(opt.variationAmount || 0), 0);

#### weight = selectedOptions.reduce((sum, opt) => sum + parseFloat(opt.weight || 0), 0);

#### iceWeight = selectedOptions.reduce((sum, opt) => sum + parseFloat(opt.withIceWeight || 0), 0);

#### }

#### else if (variation) {

#### pName = variation.variation\_name;

#### price = parseFloat(variation.variation\_amount || 0);

#### weight = parseFloat(variation.weight || 0);

#### iceWeight = parseFloat(variation.withIceWeight || 0);

#### }

#### const existingCartItems = JSON.parse(sessionStorage.getItem(cartKey)) || [];

#### const existingItemIndex = existingCartItems.findIndex(item => item.productId === pId && item.name === pName);

#### if (existingItemIndex !== -1) {

#### const newQty = existingCartItems[existingItemIndex].qty + qty;

#### if (newQty > 0) {

#### existingCartItems[existingItemIndex].qty = newQty;

#### }

#### } else {

#### const newItem = {

#### productId: selectedProduct.p\_id,

#### qty: qty,

#### price: price,

#### name: pName,

#### img: selectedProduct.image\_url,

#### weight: weight,

#### iceWeight: iceWeight,

#### };

#### existingCartItems.push(newItem);

#### }

#### sessionStorage.setItem(cartKey, JSON.stringify(existingCartItems));

#### const addToCartToast = document.getElementById("addtocart\_toast");

#### const spanElement = addToCartToast.querySelector(".desc span");

#### spanElement.textContent = pName;

#### addToCartToast.classList.add("show");

#### setTimeout(() => {

#### addToCartToast.classList.remove("show");

#### }, 3000);

#### };

#### const handleAddToWishlist = async (productId, variationId) => {

#### const cust\_id = sessionStorage.getItem('VVKuserId');

#### if (!cust\_id) {

#### navigate("/sign-in", { state: { from: location } });

#### return;

#### }

#### if (!productId) {

#### alert('Invalid product selection.');

#### return;

#### }

#### try {

#### const response = await axios.post('https://api.vivekflowers.com/api/wishlist', {

#### cust\_id,

#### product\_id: productId,

#### variation\_id: variationId || null,

#### }); if (response.data.status === 200) {

#### // Show success toast for wishlist

#### const wishlistToast = document.getElementById("wishlist\_toast");

#### const spanElement = wishlistToast.querySelector(".desc span");

#### spanElement.textContent = 'Product added to wishlist successfully!';

#### wishlistToast.classList.add("show");

#### setTimeout(() => {

#### wishlistToast.classList.remove("show");

#### }, 3000);

#### } else {

#### alert(response.data?.message || 'Failed to add product to wishlist.');

#### }

#### } catch (error) {

#### console.error("Error adding to wishlist:", error);

#### }

#### }; return (

#### <>

#### {loader && <Flower /> }

#### <div>

#### <Header />

#### <div className="sticky-header-next-sec ec-breadcrumb section-space-mb">

#### <div className="container">

#### <div className="row">

#### <div className="col-12">

#### <div className="row ec\_breadcrumb\_inner">

#### <div className="col-md-6 col-sm-12">

#### <h2 className="ec-breadcrumb-title">{subcategory\_id}</h2>

#### </div>

#### <div className="col-md-6 col-sm-12">

#### <ul className="ec-breadcrumb-list">

#### <li className="ec-breadcrumb-item">

#### <Link to="/home">Home</Link>

#### </li>

#### <li className="ec-breadcrumb-item active">{subcategory\_id}</li>

#### </ul>

#### </div>

#### </div>

#### </div>

#### </div>

#### </div>

#### </div>

#### <div className="shop-pro-content container">

#### <div className="shop-pro-inner">

#### <div className="row">

#### {products.length > 0 ? (

#### products.map((product, index) => (

#### <div className="col-lg-3 col-md-6 col-sm-3 col-xs-6 mb-6 col-12 pro-gl-content" key={index}>

#### <div className="ec-product-inner" style={{ border: 'solid 1px lightblue' }}>

#### <div className="ec-pro-image-outer">

#### <div className="ec-pro-image">

#### <a href={`/product?id=${product.p\_id}`} className="image">

#### <img

#### src={`https://api.vivekflowers.com/${product.image\_url}`}

#### alt={product.name}

#### style={{ width: '100%' }}

#### />

#### </a>

#### <div className="ec-pro-actions">

#### <button className="ec-btn-group wishlist" onClick={() => handleAddToWishlist(product.p\_id, product.selectedVariation?.variation\_id)} title="Wishlist">

#### <i className="fi-rr-heart"></i>

#### </button>

#### <a href={`/product?id=${product.p\_id}`} className="add-to-cart" title="View Product">

#### <i className="fi-rr-shopping-basket"></i>

#### </a>

#### </div>

#### </div>

#### </div>

#### <div className="ec-pro-content" style={{ padding: '10px' }}>

#### <h5 className="ec-pro-title">

#### <a href={`/product?id=${product.p\_id}`}>{product.productName || "Product Name"}</a>

#### </h5>

#### <span className="ec-price">

#### {product.amount && parseFloat(product.amount) > 0 &&

#### <span className="new-price">${Number(product.selectedVariation ? product.selectedVariation.variation\_amount : product.amount).toFixed(2)}</span>

#### }

#### </span>

#### {product.variations && product.variations.length > 0 && (

#### <>

#### {product.variations.some((variation) => variation.variation\_options?.length > 0) ? (

#### product.variations.map((variation) =>

#### variation.variation\_options?.length > 0 ? (

#### <div key={variation.variation\_id} className="variation-dropdown">

#### <select

#### id={`variation-option-select-${variation.variation\_id}`}

#### value={selectedVariationOptions[product.p\_id]?.[variation.variation\_id] || ''}

#### onChange={(e) => handleVariationOptionChange(e, variation.variation\_id, product.p\_id)}

#### className="form-control border border-primary my-3">

#### <option value="" disabled>Select {variation.variation\_name}</option>

#### {variation.variation\_options.map((option) => (

#### <option key={option.vOptionId} value={option.vOptionId}>

#### {option.variationName} - ₹{option.variationAmount}

#### </option>

#### ))}

#### </select>

#### </div>

#### ) : null

#### )

#### ) : (

#### <div className="variation-dropdown">

#### {product.variations && product.variations.length > 0 && (

#### <select

#### className="form-control border border-primary my-3"

#### onChange={(e) => handleVariationChange(index, parseInt(e.target.value))}

#### value={product.selectedVariation ? product.selectedVariation.variation\_id : ''}

#### >

#### <option value="" disabled>Select Variation</option>

#### {product.variations.map(variation => (

#### <option key={variation.variation\_id} value={variation.variation\_id}>

#### {variation.variation\_name} {parseFloat(variation.variation\_amount) > 0 && -` $ ${variation.variation\_amount}`}

#### </option>

#### ))}

#### </select>

#### )}

#### </div>

#### )}

#### </>

#### )}

#### <span className="ec-price pricing-sec">

#### {product.amount && parseFloat(product.amount) > 0 ? (

#### product.variations && product.variations.length > 0 ?

#### // Check if either a variation is selected or any variation has a selected option

#### (product.selectedVariation || product.variations.some(variation => variation.selectedOption)) ? (

#### <>

#### <div className="qty-plus-minus" style={{ display: 'flex', alignItems: 'center' }}>

#### <button className="qty-btn" onClick={() => handleDecreaseQty(index)}>-</button>

#### <input className="qty-input" type="text" value={product.quantity} readOnly />

#### <button className="qty-btn" onClick={() => handleIncreaseQty(index)}>+</button>

#### </div>

#### <div style={{ float: 'right' }}>

#### <button className="btn btn-primary" onClick={() => addCart(product.p\_id, product.quantity, product.productName)}>

#### Add to Cart

#### </button>

#### </div>

#### </>

#### ) : (

#### <small style={{ marginLeft: '5px', fontWeight: '500', color: 'darkred' }}>Please Select a Variation</small>

#### )

#### : (

#### <>

#### <div className="qty-plus-minus" style={{ display: 'flex', alignItems: 'center' }}>

#### <button className="qty-btn" onClick={() => handleDecreaseQty(index)}>-</button>

#### <input className="qty-input" type="text" value={product.quantity} readOnly />

#### <button className="qty-btn" onClick={() => handleIncreaseQty(index)}>+</button>

#### </div>

#### <div style={{ float: 'right' }}>

#### <button className="btn btn-primary" onClick={() => addCart(product.p\_id, product.quantity, product.productName)}>

#### Add to Cart

#### </button>

#### </div>

#### </>

#### )

#### ) : (

#### <span className="new-price text-danger">Call us 848 200 5992</span>

#### )}

#### </span>

#### </div>

#### </div>

#### </div>

#### ))

#### ) : (

#### <div className="col-12 text-center">

#### <h4>No Products Available</h4>

#### </div>

#### )}

#### </div>

#### </div>

#### <div id="addtocart\_toast" className="addtocart\_toast">

#### <div className="desc"><span> </span> Added To Cart</div>

#### </div>

#### <div id="wishlist\_toast" className="addtocart\_toast">

#### <div className="desc"><span> </span> </div>

#### </div>

#### <Footer />

#### </div>

#### </> );

#### };export default ProductCategories;

1. **SYSTEM TESTING**

Testing is a crucial phase in the software development lifecycle to ensure the reliability and functionality of the VivekFlowers platform. The following testing methodologies were applied:

**4.1 Unit Testing:**

* Individual components such as user authentication, product listing, and order placement were tested separately.
* Ensured correct validation of inputs during user registration and checkout.

**4.2 Integration Testing:**

* Verified seamless communication between frontend and backend components.
* Tested API responses for product retrieval, cart management, and order processing.

**4.3 Functional Testing:**

* Ensured that customers can browse products, add items to the cart, and successfully complete purchases.
* Checked the admin panel functionalities, including product addition, order status updates, and inventory management.

**4.4 Performance Testing:**

* Assessed system response time under various loads to ensure a smooth user experience.
* Verified fast loading times for product pages and checkout processes.

**4.5 Security Testing:**

* Ensured encrypted storage of user credentials and payment details.
* Tested protection against SQL injection, cross-site scripting (XSS), and unauthorized access.

**5. CONCLUSION**

The **VivekFlowers** e-commerce platform successfully addresses the challenges faced by traditional flower shops by providing a seamless online shopping experience. Customers can now browse a wide variety of flowers, garlands, and pooja items, place orders conveniently, and track their purchases in real time. The inclusion of an admin panel enhances the efficiency of inventory and order management. Secure payment integration and an intuitive user interface make VivekFlowers a reliable and user-friendly platform. With continuous improvements, this system can further revolutionize the way customers purchase floral and religious products online.

**6. FUTURE ENHANCEMENT**

To further improve the VivekFlowers platform, several enhancements can be implemented. Integrating AI-powered recommendation systems can personalize the shopping experience by suggesting products based on user preferences and purchase history. Expanding payment options to include UPI, credit/debit cards, and wallets will provide users with more flexibility. Adding real-time order tracking with notifications can enhance transparency and customer satisfaction. Implementing a subscription-based model for regular flower deliveries can cater to recurring customers. Additionally, developing a mobile application will improve accessibility and convenience for users. Enhancing the admin dashboard with detailed analytics and sales forecasting can help in better inventory and business management.

**7. BIBLIOGRAPHY**

* https://dev.mysql.com/doc/
* <https://docs.python.org/3/>
* https://flask.palletsprojects.com/
* <https://react.dev/>
* https://platform.openai.com/docs/
* https://getbootstrap.com/docs/