Smart Paint Shop

Project Report Submitted by

AKASH KRISHNA

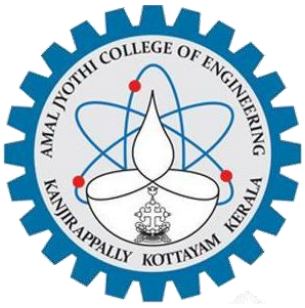
Reg. No.: AJC24MCA2006

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS

(MCA TWO YEAR)

APJABDULKALAMTECHNOLOGICALUNIVERSITY



AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2025-2026

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report titled "**Smart Paint Shop"** is the bona fide work of **Akash Krishna (Regno: AJC24MCA-2006)** carried out in partial fulfillment of the requirements for the award of the **Degree of Master of Computer Applications** at **Amal Jyothi College of Engineering Autonomous, Kajirappally.** The project was undertaken during the period from **July 07, 2025** to **October 30, 2025**

Mr.Amal K Jose

Internal Guide

Ms. Meera Rose Mathew

Coordintor

Dr. Bijimol TK
Head of the Department

DECLARATION

I hereby declare that the project report **"Smart Paint Shop"** is a bona fide work done at **Amal Jyothi College of Engineering Autonomous, Kanjirappally**, towards the partial fulfilment of the requirements for the award of the **Master of Computer Applications (MCA)** during the period from **July 07, 2025** to **October 30, 2025**.

Date:30-10-2025 **Akash Krishna**
KANJIRAPPALLY **Reg: AJC24MCA-2006**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Dr. Bijimol T K.** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr.Amal K Jose** for his inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

Akash Krishna

ABSTRACT

The **Smart Paint Shop** is a comprehensive, all-in-one mobile application developed using the **Flutter framework** and **Firebase backend (Firestore, Authentication, Storage)**, designed to revolutionize the traditional paint retail experience. The primary objective of this project is to solve common market challenges—specifically **customer struggles with color visualization**, **inefficient inventory management**, and a **fragmented purchasing journey**—by integrating advanced digital tools and intelligent business insights.

The system efficiently consolidates the entire paint purchasing process. Key features include a **multi-brand e-commerce catalogue** , a robust **Product Management System** for administrators , and a complete **secure checkout workflow** incorporating location-based delivery and payment gateway integration. Crucially, the platform includes a **Visualization and Recommendation Engine** , which processes images for color preview and generates personalized recommendations. This engine is critical for enhancing customer decision-making and is projected to **reduce high return rates** due to color dissatisfaction.

To extend functionality and ensure seamless retail operations, the system provides separate, **role-based dashboards** for **Admin, Manager, and Customer**. Managers are equipped to handle **Point of Sale (POS) operations**, **process orders**, and conduct **inventory updates**. This unified approach successfully merges the front-end customer journey with an intelligent business insights backend.

Future development plans include integrating **AR-based room scanning** for accurate measurements and advanced **computer vision for wall segmentation** , alongside **AI-based demand prediction** and **automated stock alerts**. By combining these technologies, the **Smart Paint Shop** offers a smart, accessible, and comprehensive platform that successfully digitizes the entire paint purchasing journey.

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	3
2	SYSTEM STUDY	5
2.1	INTRODUCTION	6
2.2	LITERATURE REVIEW	6
2.3	PROPOSED SYSTEM	7
2.4	ADVANTAGES OF PROPOSED SYSTEM	8
3	REQUIREMENT ANALYSIS	9
3.1	FEASIBILITY STUDY	10
3.1.1	ECONOMICAL FEASIBILITY	10
3.1.2	TECHNICAL FEASIBILITY	10
3.1.3	BEHAVIORAL FEASIBILITY	11
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	12
3.1.5	GEOTAGGED PHOTOGRAPH	13
3.2	SYSTEM SPECIFICATION	14
3.2.1	HARDWARE SPECIFICATION	14
3.2.2	SOFTWARE SPECIFICATION	14
3.3	SOFTWARE DESCRIPTION	15
3.3.1	FLUTTER	15
3.3.2	FIREBASE	15
4	SYSTEM DESIGN	17
4.1	INTRODUCTION	18
4.2	UML DIAGRAM	20
4.2.1	USE CASE DIAGRAM	22

4.2.2	SEQUENCE DIAGRAM	23
4.2.3	CLASS DIAGRAM	28
4.2.4	OBJECT DIAGRAM	30
4.2.5	COMPONENT DIAGRAM	31
4.3	USER INTERFACE DESIGN USING FIGMA	32
4.4	DATABASE DESIGN	35
5	SYSTEM TESTING	39
5.1	INTRODUCTION	40
5.2	TEST PLAN	40
5.2.1	UNIT TESTING	41
5.2.2	INTEGRATION TESTING	42
5.2.3	VALIDATION TESTING	43
5.2.4	USER ACCEPTANCE TESTING	43
5.2.5	AUTOMATION TESTING	44
5.2.6	FLUTTER TESTING	44
6	IMPLEMENTATION	53
6.1	INTRODUCTION	54
6.2	IMPLEMENTATION PROCEDURE	54
6.2.1	USER TRAINING	54
6.2.2	TRAINING ON APPLICATION SOFTWARE	55
6.2.3	SYSTEM MAINTENANCE	56
7	CONCLUSION & FUTURE SCOPE	57
7.1	CONCLUSION	58
7.2	FUTURE SCOPE	58
8	BIBLIOGRAPHY	60
9	APPENDIX	62
9.1	SCREEN SHOTS	63
9.2	SAMPLE CODE	64
9.3	GIT LOG	72

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

In the current market, purchasing paint for a home or project is a multi-step, often cumbersome process. Customers must visit physical hardware stores, browse limited physical swatches, manually calculate the required paint quantity (often leading to significant error), and separately source painting services. While some e-commerce websites exist, they often lack specialized features for paint, such as accurate color visualization or complex product filtering. This disjointed experience results in wasted time, incorrect purchases, and customer frustration. The "Smart Paint Shop" project aims to create a centralized, intelligent mobile platform that solves these specific problems.

The project is structured in phases:

Mini Project Phase (Current Focus)
This phase concentrated on delivering the core functionalities of the "Smart Paint Shop" application.

Core Functionality:Development of the user-facing mobile app for browsing paints and textures, viewing combinations, adding items (with specific pack sizes) to a cart, user authentication (Customer, Manager, Admin roles), and the checkout process including payment integration (Razorpay) and location-based delivery capture. It also includes the Admin/Manager dashboard for managing products and textures.
Key Features: Implementation of the multi-role system, Firebase backend setup (Auth, RTDB, Storage), dynamic product/texture display, cart management, and initial AR calculator framework setup.

Main Project Phase (Future Scope)
Further development could expand the system significantly.

Extended Scope: Full implementation of the AR paint visualization ("Try on Wall"), integration with real-time inventory systems, a dedicated painter booking marketplace module, and advanced order tracking.

Advanced Features:Potential integration of AI for color recommendations based on user photos or preferences.

Technology Stack
The system is built using Flutter and Firebase

1.2 PROJECT SPECIFICATION

The AI-Powered Paint Shop app is a smart platform developed as a Mini Project. The system is built using the Flutter + Firebase technology stack to create a cross-platform application that improves paint shop operations using Artificial Intelligence and digital tools.

Scope and Goal

The primary goal of the Mini Project is to establish a smart platform that combines traditional paint shop services with AI-powered visual tools, online ordering, and intelligent business insights. This is intended to address current market challenges, including customer difficulty in color visualization and inefficient, manual inventory management. Ultimately, the system aims to improve operations and enhance customer satisfaction.

Core Functionalities

The system divides responsibilities among three main user types. The Admin module oversees the system by registering and managing paint products , monitoring sales and inventory levels , and generating business reports. The Manager is responsible for handling Point of Sale (POS) operations , managing daily operations , processing customer orders and payments , and managing local painter information. The User (Customer) can browse the paint catalog with color visualization , place orders with quantity estimation , track order status , and manage a wishlist/cart.

Future Scope

This project is designed as the foundation (Mini Project Phase) for a full-scale integrated system. Future phases (Main Project) will incorporate advanced modules, including AR-based room scanning for accurate measurements , advanced AI-powered color recommendations , and a full stock tracking system with supplier alerts.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The Smart Paint Shop app is a targeted technological response to significant systemic challenges inherent in the traditional, specialized retail process for home improvement. The development of this system is rooted in the recognition that while customer interest in home improvement is high, the manual and fragmented processes severely undermine the customer experience and business efficiency. The current, non-automated system is often characterized by customer struggles with color visualization, leading to decision uncertainty and high return rates due to color dissatisfaction. The core problem addressed is the inefficiency in guiding customers to a confident purchase and the failure to manage inventory and sales in a data-driven, structured manner that respects the business's resources.

2.2 LITERATURE REVIEW

The core objective of this review is to offer a comprehensive synthesis of the relationship between computer vision, AI-driven recommender systems, and digital retail management. It proceeds by first establishing the foundational principles that govern these relationships. This includes recognizing digital color visualization as a complex problem, where highly variable environmental lighting and camera quality affect the perception of digital swatches . Foundational computer vision models are introduced, particularly semantic image segmentation (to identify walls) and photorealistic rendering, which quantitatively link a digital RGB value to its perceived appearance in a physical space

The review then explores the three main measurement methodologies (i.e., applications) that translate these principles into clinical (i.e., retail) applications. These include Augmented Reality (AR) "virtual try-ons" for real-time customer decision support , AI-based recommender systems for personalized style and theme-based suggestions and Business Intelligence (BI) dashboards for analyzing sales patterns and predicting inventory needs.

2.3 PROPOSED SYSTEM

The Smart Paint Shop app is proposed as a comprehensive smart retail support system designed to directly resolve the critical challenges of customer color selection uncertainty and inefficient, manual shop management. The core objective of the system is to ensure the entire customer journey is visual, confident, and convenient, and that the business operations are data-driven and efficient. The project is built using the robust and scalable Flutter + Firebase stack.

The initial Mini Project Phase establishes the foundational functionality as a smart product catalog, an AI-powered color suggestion tool, and a multi-role operational dashboard. This system begins its process when the Admin registers and manages the paint products in the database. A User (Customer) can then browse this catalog, explore color options, and use the AI tool to get recommendations or visualize colors.

The system introduces crucial automation for both the customer and the manager. The Customer can place an order, get help with quantity estimation, and track their order status. The Manager module handles the Point of Sale (POS) operations, processes the customer order and payments, and manages inventory with stock alerts. Critically, to improve customer engagement and service, the system incorporates a WhatsApp bot integration for automated order updates and a Professional Painter Management module to connect customers with local painter info. The Admin and Manager dashboards provide real-time sales analytics and business insights

The design is intentionally flexible, with architecture ready to support future expansion into the Main Project Phase. This will introduce advanced features such as AR-based room scanning for precise measurements, AI-powered visual simulation with realistic lighting effects, and a full-scale stock tracking system with automated supplier alerts and delivery integration.

2.4 ADVANTAGES OF PROPOSED SYSTEM

The implementation of the AI-Powered Paint Shop app offers significant advantages by directly resolving the major drawbacks of the existing manual process, focusing on customer experience, operational efficiency, and sales intelligence.

- **Enhanced Customer Confidence and Experience:** The system's primary advantage is its focus on solving the customer's color visualization struggles. By providing AI-powered suggestions and digital previews, the system makes the selection process easy and confident, directly addressing a key market challenge.
- **Improved Efficiency and Reduced Errors:** The system automates crucial steps such as inventory management, Point of Sale (POS) operations, and order tracking. This eliminates manual errors, reduces delays, and frees up staff to focus on customer service rather than disorganized paperwork.
- **Reduced Return Rates:** By allowing customers to visualize colors accurately before buying, the system directly targets the problem of color dissatisfaction. This is projected to significantly reduce high product return rates, saving the business money and improving customer satisfaction.
- **Increased Customer Engagement and Loyalty:** The platform increases customer touchpoints through features like WhatsApp bot integration for order updates, wishlists, and providing useful local painter info. These tools build a stronger customer relationship and encourage repeat business.
- **Data-Driven Business Insights:** The Admin and Manager dashboards provide real-time sales analytics and inventory levels. This allows the business to move from reactive to predictive management, optimizing stock and understanding sales trends, which is impossible in the current manual system.
- **Scalability and Modernization:**Building the system on the robust and scalable Flutter + Firebase stack provides a flexible foundation.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A Feasibility Study was conducted for the Smart Paint Shop project to rigorously assess its viability and practicality before committing significant resources to development. The study focused on three critical areas: Economical, Technical, and Behavioral feasibility, ensuring the proposed system is both justified and achievable

3.1.1 Economical Feasibility

The Smart Paint Shop system is economically viable as it is built using Flutter, an open-source framework, and Firebase,. This combination eliminates server-side development and maintenance costs, and the 'Spark' plan (free tier) covers initial development and low-traffic operation entirely. Costs are predictable and scale with usage, such as Razorpay's transaction-based fees and Firebase's 'Blaze' (pay-as-you-go) plan, which are minimal for a small-to-medium enterprise. By automating inventory management, order processing, and providing a superior digital storefront, the system significantly reduces manual administrative work for store managers and opens a new 24/7 digital sales channel. The high potential for increased sales, improved customer retention (through features like the AR calculator), and operational savings results in a highly favorable cost-benefit ratio.

3.1.2 Technical Feasibility

The Smart Paint Shop system is technically feasible as it is built using Flutter and Firebase, a reliable, scalable, and widely supported cross-platform technology stack. Flutter is a mature, high-performance framework for building natively compiled applications for both Android and iOS from a single Dart codebase. Firebase provides a secure backend (Auth, RTDB, Storage) that handles real-time data synchronization. The architecture supports integration with native device features, including the camera and AR (via ARCore/ARKit plugins), which is essential for the paint calculator feature. As a cloud-based system, it is accessible from anywhere. The integration of the Razorpay payment gateway and Geolocator packages is a standard, reliable practice. Skilled Flutter and Firebase developers are readily available, ensuring the project can be maintained and scaled. The technology stack allows for future scalability, including the full implementation of the AR visualizer and AI-based recommendation engines.

3.1.2 Behavioral Feasibility

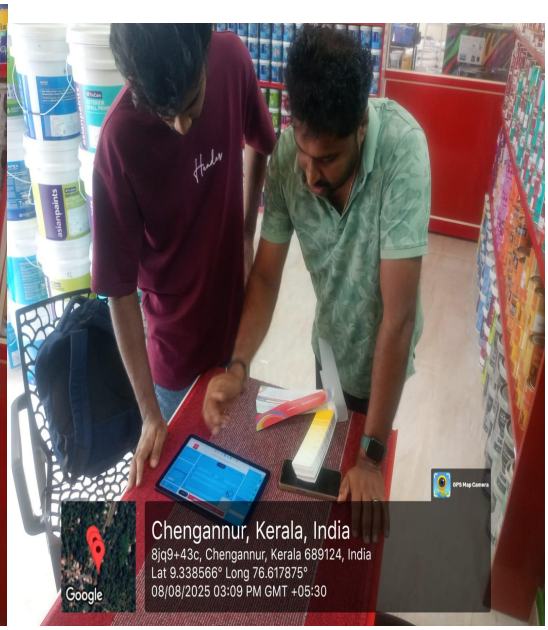
Behavioral (or Operational) feasibility gauged the organizational and user acceptance of the new system. High user acceptance is anticipated for both customers and managers. For customers, the app is designed to solve the primary frustrations of the current process: inaccurate quantity estimation (solved by the AR calculator), poor visualization (solved by the texture/combination viewer), and the fragmented, time-consuming purchasing journey. For managers, the system provides a clear, digital dashboard for managing products, textures, and orders, which is a significant improvement over manual or ledger-based systems. This simplification of tasks ensures managers will readily adopt the tool. The system's alignment

with real-world needs was validated through informal interviews with local paint store managers and potential customers.

3.1.1 Feasibility Study Questionnaire

• Is there any digital system currently in place for tracking sales, inventory, or customer orders?
• What are the biggest daily challenges faced by the shop manager and staff?
• What are the main inconveniences or frustrations experienced by customers when buying paint?
• What difficulties do customers face when trying to select the right paint color or product type?
• What are the main concerns customers have before making a purchase (e.g., "the color will look different at home," "I'm buying the wrong amount," "it won't match my furniture")?
• Can you explain all the steps involved from when a customer first asks about a product to when they complete the purchase?
• What features would you like to see in a new digital system to improve shop management and sales tracking?
• What features could make the paint selection and buying process easier and more efficient for customers?
• What are the challenges faced by the business *after* a sale (e.g., handling returns, managing complaints, getting customer feedback)?

3.1.2 Geotagged Photograph



3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - Intel Core i3 (or equivalent)

RAM - 4 GB (Minimum) / 8 GB (Recommended)

Hard disk - 250 GB (Minimum)

3.2.2 Software Specification

Front End - **Flutter**

Back End -**Firebase**

Database - **Firebase Database**

Client on PC - **Android 13+**

3.3 SOFTWARE DESCRIPTION

3.3.1 Flutter

Flutter is the modern UI toolkit from Google used to build the entire "Smart Paint Shop" mobile application. As a cross-platform framework, it allows a single Dart codebase to be compiled into high-performance native applications for both Android and iOS. Flutter is responsible for rendering all user interfaces, from the customer-facing ProductDetailPage (with its parallax headers) and the ExploreProductPage (with its grid and Hero animations) to the secure, role-based dashboards for Managers and Admins. Its stateful widget system is essential for managing the app's complex UI state, such as the interactive cart (where users change pack sizes) and the dynamic texture combination selector.

3.3.2 Firebase

Firebase serves as the comprehensive Backend-as-a-Service (BaaS) platform for the entire application, eliminating the need for traditional, manually-coded server-side logic and REST APIs. It provides a suite of tightly integrated services that handle the core functionality:

Firebase Authentication is used to securely manage all user accounts and roles (Customer, Manager, Admin), handling login, registration, and session persistence.
Firebase Storage is used as the scalable, cloud-based file system. It hosts all static media,

including the high-resolution product images , texture combination images , benefit images, and the downloadable PDF brochures that managers upload.

3.3.3 Firebase (Database)

The Firebase Realtime Database (RTDB) serves as the system's persistent, real-time data store. As a NoSQL database, its flexible, JSON-based structure is ideal for managing the varied and complex data models required by the Smart Paint Shop. This includes structured data like user profiles (with userType), the main /products node (with nested packSizes and benefits lists), the /textures node (with nested combinations and productsUsed lists), and the /colorCategories used for linking. The RTDB's real-time synchronization is its most critical feature, ensuring that any changes made by a Manager (like a price update or stock adjustment) are reflected in the customer's app almost instantly. This guarantees data consistency across all user-facing components.

CHAPTER 4
SYSTEM DESIGN

4.1INTRODUCTION

The Smart Paint Shop Mini Project is designed as a cross-platform mobile application (built with Flutter and Firebase) that connects Customers, store Managers, and Administrators through a unified platform. The system focuses on product browsing, advanced texture visualization, e-commerce workflow, and backend inventory management, ensuring a seamless user experience and efficient store operations.
System design is the process of defining the architecture, modules, interfaces, and data flow of the entire system to meet functional and non-functional requirements. It serves as a blueprint for developers to build a scalable, secure, and maintainable application.
In this project, the system design phase includes:

• Architectural Design: Defines how various components (the Flutter client, Firebase backend services, and the mobile UI) interact.
• Database Design: Structures the data models (Product, Texture, User, Cart, Order, ColorCatalogue) within the Firebase Realtime Database (a NoSQL JSON tree).
• Component Design: Defines the core application modules such as Authentication, Product Management (with pack sizes and benefits), Texture Combinations, Cart & Checkout, and the AR Calculator.
• Interface Design: Describes how Customers, Managers, and Admins interact with the system through the Flutter-based mobile interface (including the user-facing app and the manager-side dashboard).
The overall design ensures that:
• The system is modular, with separate, well-defined modules (e.g., product_model.dart, cart_page.dart, manage_textures_page.dart).
• Data is secured through Firebase Authentication tokens and Firebase Security Rules to enforce role-based access.
• The application remains responsive and reliable, supporting real-time data synchronization.
• Communication between the Flutter client and backend is handled by the Firebase SDKs, with data stored and synced in real-time via the Realtime Database.

4.2UML DIAGRAM

Unified Modeling Language (UML) diagrams are a standardized way to visualize the design and architecture of a software system. They help developers, designers, and stakeholders understand the structure, behavior, and interactions within a system before actual implementation. For the Smart Paint Shop project, UML diagrams provide a comprehensive view of how Customers, Managers, Administrators, and backend services interact to manage paint products, texture combinations, and customer orders in an efficient and secure manner. By using UML, the project ensures clarity in requirements, reduces ambiguity, and serves as a blueprint for development.

The Use Case Diagram captures the functional requirements of the system from the perspective of different actors. In this project, the primary actors are the Customer, who interacts with the system to register, login, browse products, view texture combinations, add items to the cart, and checkout; the Manager, who manages products, textures, and orders; and the Admin, who

oversees all manager functions and also manages user roles. The diagram also includes external systems like Firebase Authentication and the Razorpay Payment Gateway. By showing the relationships between actors and use cases, the diagram clearly identifies all the operations that the system must support, making it easier for developers to focus on key functionalities.

The Class Diagram represents the static structure of the system, defining the data entities, their attributes, methods, and relationships. For this project, core classes include User, Product, Texture, Benefit, PackSize, CartItem, and Order. Each class reflects a corresponding Firebase Realtime Database (RTDB) node and encapsulates the operations that can be performed on the data. Relationships between classes, such as a Product having multiple Benefit and PackSize objects, are also clearly defined. This diagram helps in designing the database schema, the Dart data models (e.g., product_model.dart), and understanding how data flows between different parts of the system.

The **Sequence Diagram** illustrates the dynamic behavior of the system by showing how objects interact over time. It depicts step-by-step message exchanges between the User, the Frontend (Flutter UI), and the Backend (Firebase Services). For example, during the checkout process, the user request (tapping "Proceed to Checkout") travels from the CartPage to the DeliveryLocationPage, which calls the Geolocator plugin. After confirmation, the user navigates to the PaymentPage, which triggers the Razorpay SDK. Upon successful payment, a call is made to the Firebase RTDB to create an Order node and delete the Cart node, finally returning a success message to the user. Sequence diagrams are valuable for understanding the exact order of operations and ensuring that processes execute correctly.

The **Collaboration Diagram**, also called a Communication Diagram, complements the sequence diagram by focusing on object interactions and their structural relationships. It shows how services such as the Flutter Client, Firebase Authentication, Firebase RTDB, Firebase Storage, and external plugins like Razorpay_Flutter and Geolocator collaborate to perform user requests. By numbering the messages, the diagram highlights the sequence of operations while also showing which components interact directly. This is useful for analyzing the efficiency of service communication and identifying dependencies.

The **Deployment Diagram** models the physical architecture of the system. It visualizes where software components are deployed, which in this case is a serverless architecture. It shows the Flutter application (client) running on user devices (Android/iOS) and the Firebase backend running on Google's cloud infrastructure. The database (RTDB) and file storage (Storage) are fully managed cloud services. It also depicts the communication links, typically via secure HTTPS and WebSocket connections managed by the Firebase SDK. This diagram is essential for understanding that the application does not rely on a traditional, self-hosted application server, ensuring high scalability and reliability.

Finally, the **Activity Diagram** represents the workflow of operations within the system. It shows the sequence of actions that a user can perform, such as the "Add New Product" workflow for a Manager (fill form, upload images, save to Firebase) or the "Purchase Product" workflow for a

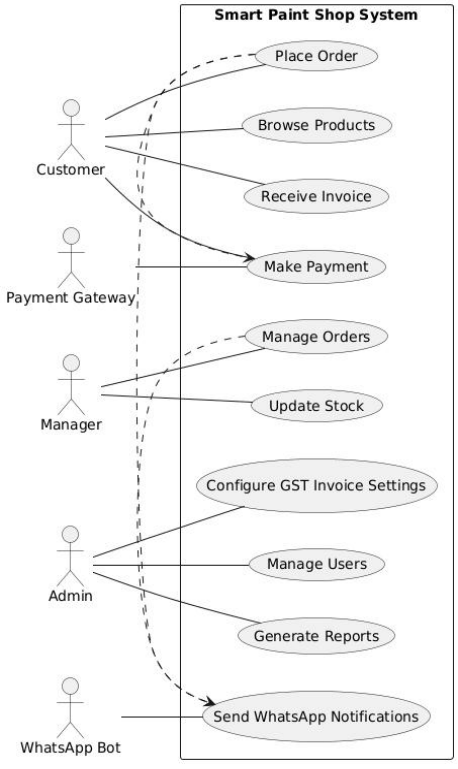
Customer (browse, select product, add to cart, change pack size, checkout, pay), along with decision points and conditional flows. Activity diagrams help in visualizing processes clearly and are particularly useful for testing and verifying that the system logic aligns with user expectations.

In summary, UML diagrams collectively provide a comprehensive blueprint of the Smart Paint Shop project. They cover the functional, structural, and behavioral aspects of the system. Use case and activity diagrams focus on user interactions and workflows, class diagrams define the data structure and relationships, sequence and collaboration diagrams illustrate component interactions, and deployment diagrams ensure a proper serverless architecture. Together, these diagrams ensure that the system is well-planned, maintainable, scalable, and secure before actual development begins.

4.2.1 USE CASE DIAGRAM

Unified Modeling Language (UML) diagrams are a standardized way to visualize the design and architecture of a software system. They help developers, designers, and stakeholders understand the structure, behavior, and interactions within a system before actual implementation. For the Smart Paint Shop project, UML diagrams provide a comprehensive view of how Customers, Managers, Administrators, and Firebase backend services interact to manage paint products, texture combinations, and customer orders in an efficient and secure manner. By using UML, the project ensures clarity in requirements, reduces ambiguity, and serves as a blueprint for development.

Fig 4.2.1. USE CASE DIAGRAM



4.2.2 SEQUENCE DIAGRAM

1. User Login and Authorization

The sequence begins when a Customer opens the Smart Paint Shop application. The Mobile App sends a POST request with login credentials to the Backend API. The Backend uses the Authentication Service to verify user details through validateUse Credentials.

The service fetches the corresponding record from the Firestore Database, and after verification, the Backend responds with a 200 OK along with JWT tokens and profile data, enabling the Customer to access the dashboard and begin shopping.

2. Product Ordering and Cart Management

This sequence outlines how a Customer selects and confirms paint products:

The Customer chooses a paint item and adds it to the cart, triggering a POST request to the Backend.

The Backend invokes the Order Service to checkProductAvailability(productId).

The service queries the Firestore inventory, updates the cart/order details, and returns a confirmation.

A Notification Service is triggered to send order updates, and finally, the Customer receives a 200 OK response with “Order Added to Cart”.

3. Review Submission

This explains how users submit feedback after a purchase:

The Customer enters a rating and comment and sends a POST request to the Backend.

The Backend calls the Review Service to createReviewEntry.

The service inserts the review into the Firestore Database.

A 201 Created response is returned, confirming that the review has been successfully recorded.

4. Painter Service Request

This sequence handles requesting professional painter assistance:

The Customer submits a request specifying paint type, location, and preferred service time, sending a POST request to the Backend.

The Backend interacts with the Painter Management Service using processPainterRequest(RequestData).

The service stores the request in Firestore and notifies available painters through the Notification Service.

The Customer receives a 200 OK response with “Painter Request Sent”.

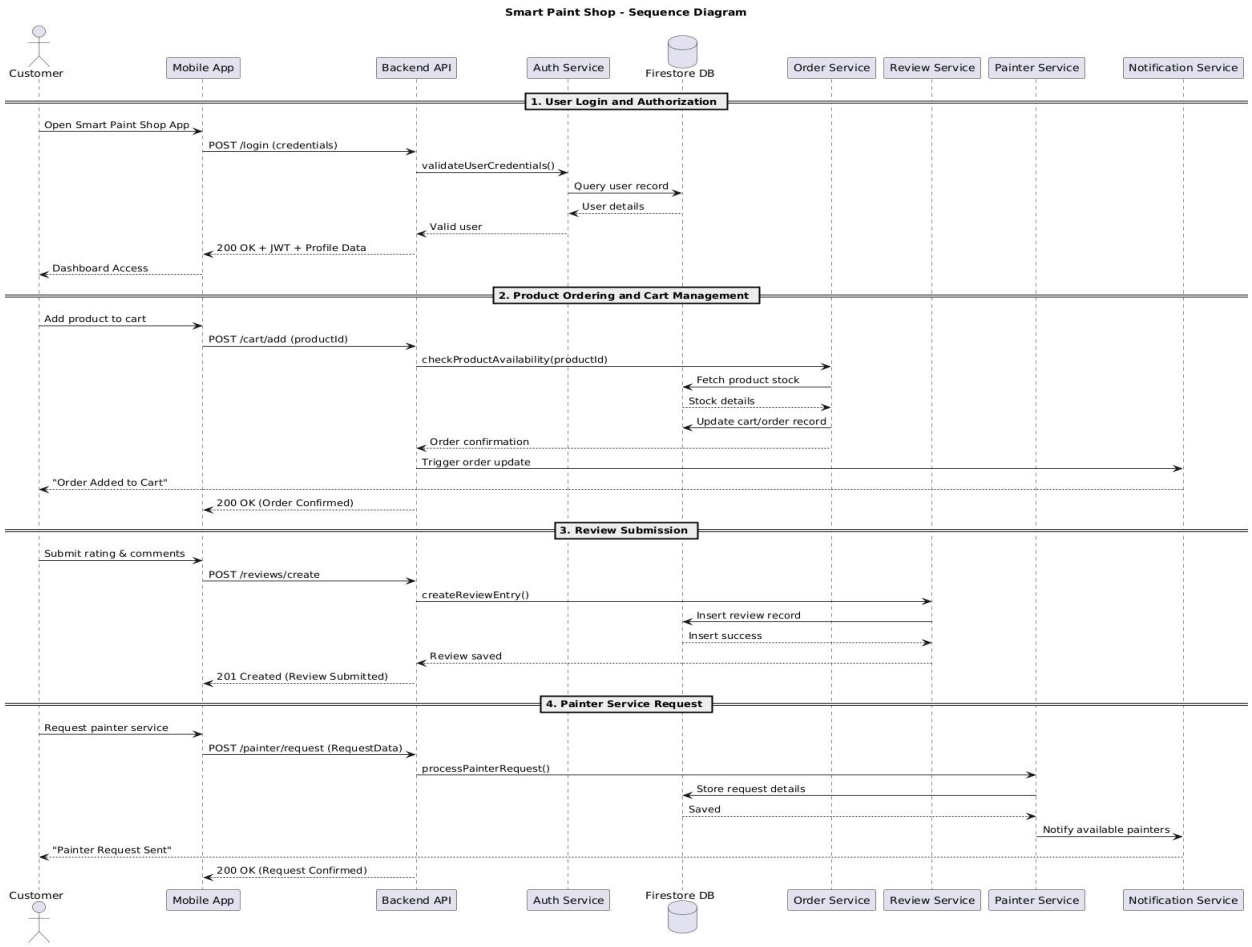


Fig 4.2.2. SEQUENCE DIAGRAM

4.2.3 Object Diagram

The Smart Paint Shop system illustrates the lifecycle of a customer interacting with the platform, covering both the user onboarding process and the product ordering workflow. The diagram is separated into two interconnected flows.

The first is the User Registration Flow, which begins when a New user registers and immediately moves into the PendingVerification state. After the system checks the provided details, the transition proceeds either to the Verified state when Verification successful, or to the Rejected state if Verification failed.
The second major flow is the Product Ordering and Fulfillment Process. A verified customer can initiate an order by entering the CartItemAdded state through an Add to Cart action. The system then updates stock and progresses to the OrderPlaced state when the User confirms order. The order request transitions to either ApprovedOrder if Stock available & payment confirmed, or moves to the terminal RejectedOrder state if Stock unavailable or payment failed.

If approved, the workflow continues when the Order assigned to delivery partner, changing the state to OutForDelivery. Upon arrival at the delivery location, the state transitions to Delivered. After delivery, the User submits review, moving the object to the ReviewSubmitted state before the process terminates.
Alternatively, the process can move into the Failed state if Delivery failed or Order cancelled by user, with both RejectedOrder and Failed leading to the final termination point.

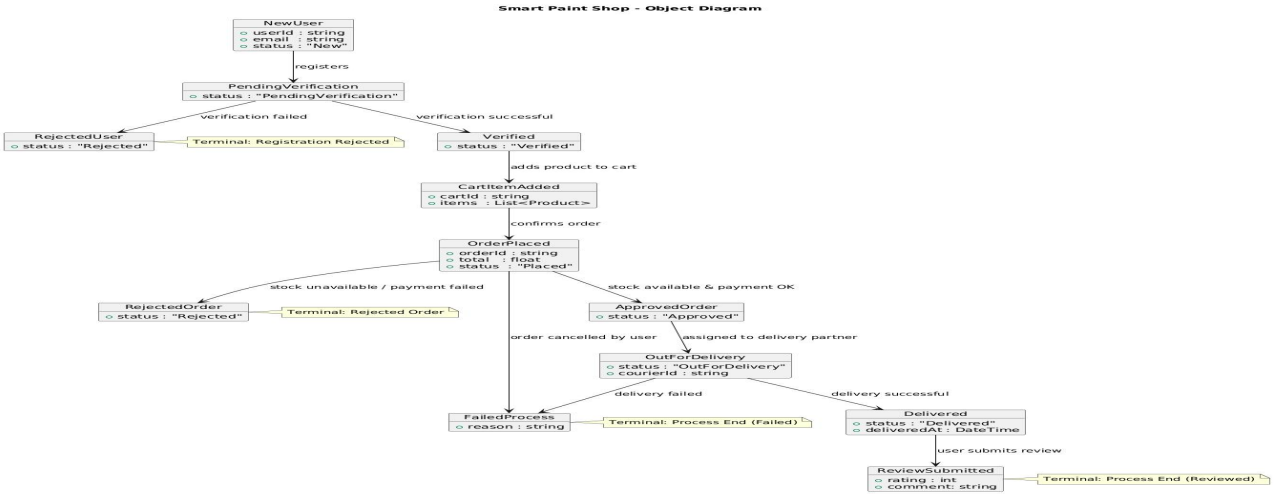


Fig 4.2.2. Object Diagram

4.2.4 Activity Diagram

Registration and Login

The process begins with User Registration in the Smart Paint Shop app.
A system check evaluates Verification Approved?
If verification fails, the flow ends with Registration Rejected.
If approved, the user can Login to the System and access the platform.

Browsing and Product Search

After login, the user proceeds to Search Products / Browse Catalogue.
A conditional check evaluates Products Available?
If no matching products are found, the process terminates with No Products Found.
If products are available, the user can Add to Cart / Continue Browsing.

Order Review and Confirmation

The selected items move into the Order Review stage.
The system performs a check: Stock & Payment Approved?
If approval fails, the process ends with Order Rejected.
If approved, the user proceeds to Place Order, and the system transitions the flow to Order Confirmed.

Delivery & Fulfillment Process

Once confirmed, the order enters the fullfillment cycle:
The system triggers Assign Delivery Partner.
The partner proceeds with Out for Delivery.
Upon reaching the address, Delivery Attempt is made.
If delivery fails, the flow moves to Delivery Failed, and the process terminates.
If the delivery succeeds, the item is marked as Delivered to Customer.

Completion and Finalization

After delivery, the flow reaches Order Completed.
The final step is for the customer to Submit Review regarding the product or delivery experience.
Once the review is submitted, all activity paths converge to the Final Termination Point, marking the end of the process.

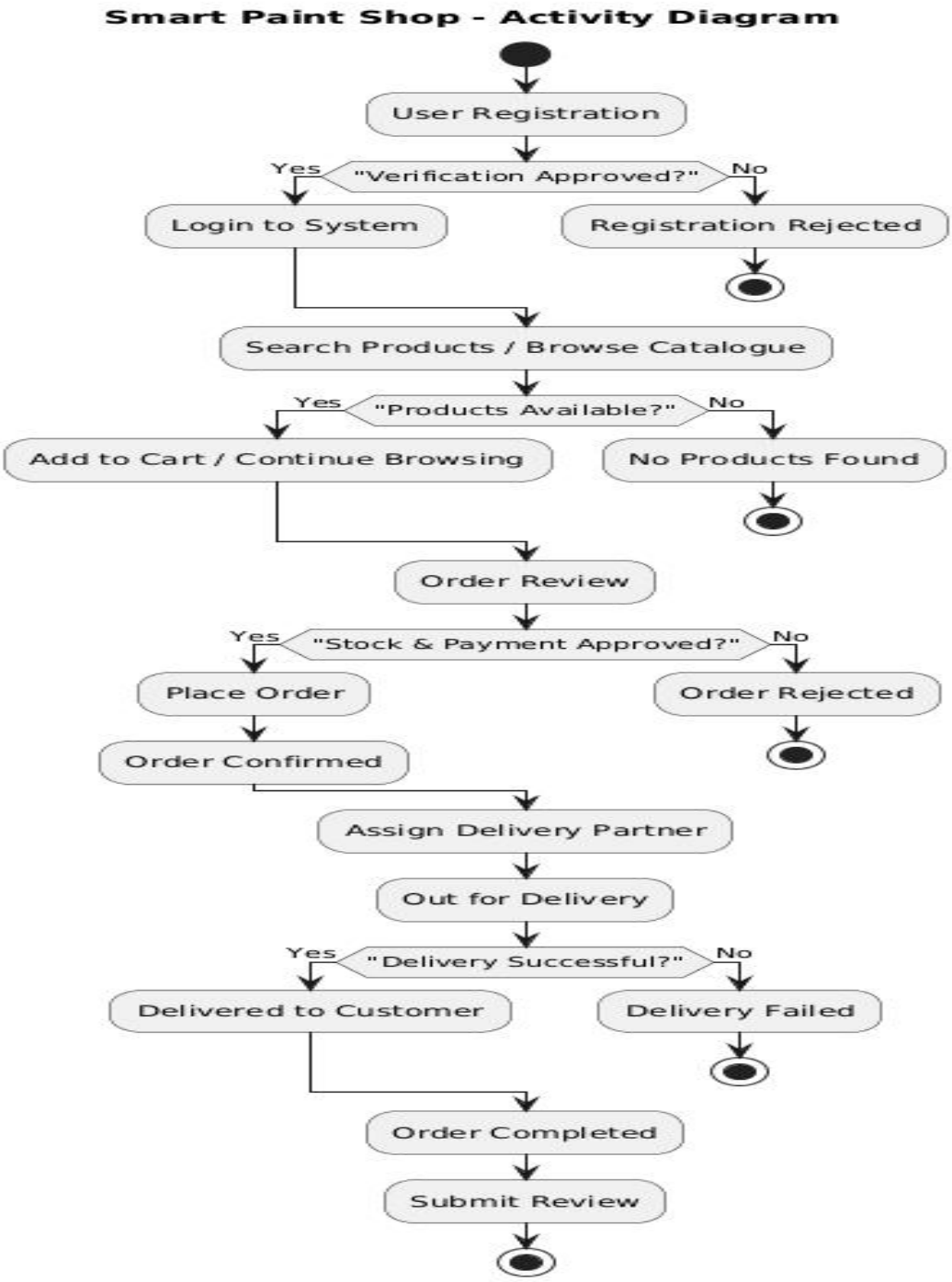


Fig 4.2.4 Activity Diagram

4.2.5 Class Diagram

The image shown represents the Class Diagram for the Smart Paint Shop – AI-Driven Paint Retail and Visualization System. This diagram provides a structural overview of the platform, illustrating the key classes, their attributes, their operations, and the relationships that form the core architecture of the application.

At the center of the system is the User class, which models the general customer interacting with the app. This class contains essential attributes such as userId, name, email, and password, along with operations like register(), login(), browseProducts(), and submitReview(). Extending the User class is the Customer class, which adds paint-specific functionalities including manageCart(), placeOrder(), and trackDelivery().
The Product class represents paint items and related supplies, containing attributes such as productId, name, brand, colorCode, price, and stock. The Product class supports key operations like updateStock(), applyDiscount(), and fetchColorPreview(). Supporting this is the Category class, grouping products under various categories such as interior paints, exterior paints, tools, or primers.
The Order class is central to the purchasing workflow, storing orderId, items, totalAmount, paymentStatus, and deliveryStatus. It includes operations such as confirmOrder(), processPayment(), and updateDeliveryStatus(). Linked to Order is the DeliveryPartner class, which contains delivery agent details and operations like assignOrder() and updateDeliveryProgress().

The system also includes the Review class, capturing customer feedback with attributes like rating, comment, and date. Connected to Product and User, this class supports operations like addReview() and computeAverageRating().
For administration, the Admin class provides elevated functionalities such as addProduct(), removeProduct(), manageInventory(), and viewSalesReports(). The InventoryManager class extends administrative capabilities by focusing on operational tasks such as restockProduct(), generateLowStockAlert(), and validateStockUpdates().

A specialized module is represented by the PainterService class, which manages professional painter requests. It includes attributes like serviceId, location, preferredDate, and operations such as processPainterRequest() and assignPainter(). The Painter class, associated with PainterService, includes painterName, experience, and contact details, and supports acceptJob() and updateJobStatus().
The relationships in the diagram show that a Customer interacts with Products, places Orders, and submits Reviews. The Admin oversees product and inventory management, while the DeliveryPartner and Painter classes collaborate with the order and service modules to fulfill customer requirements.

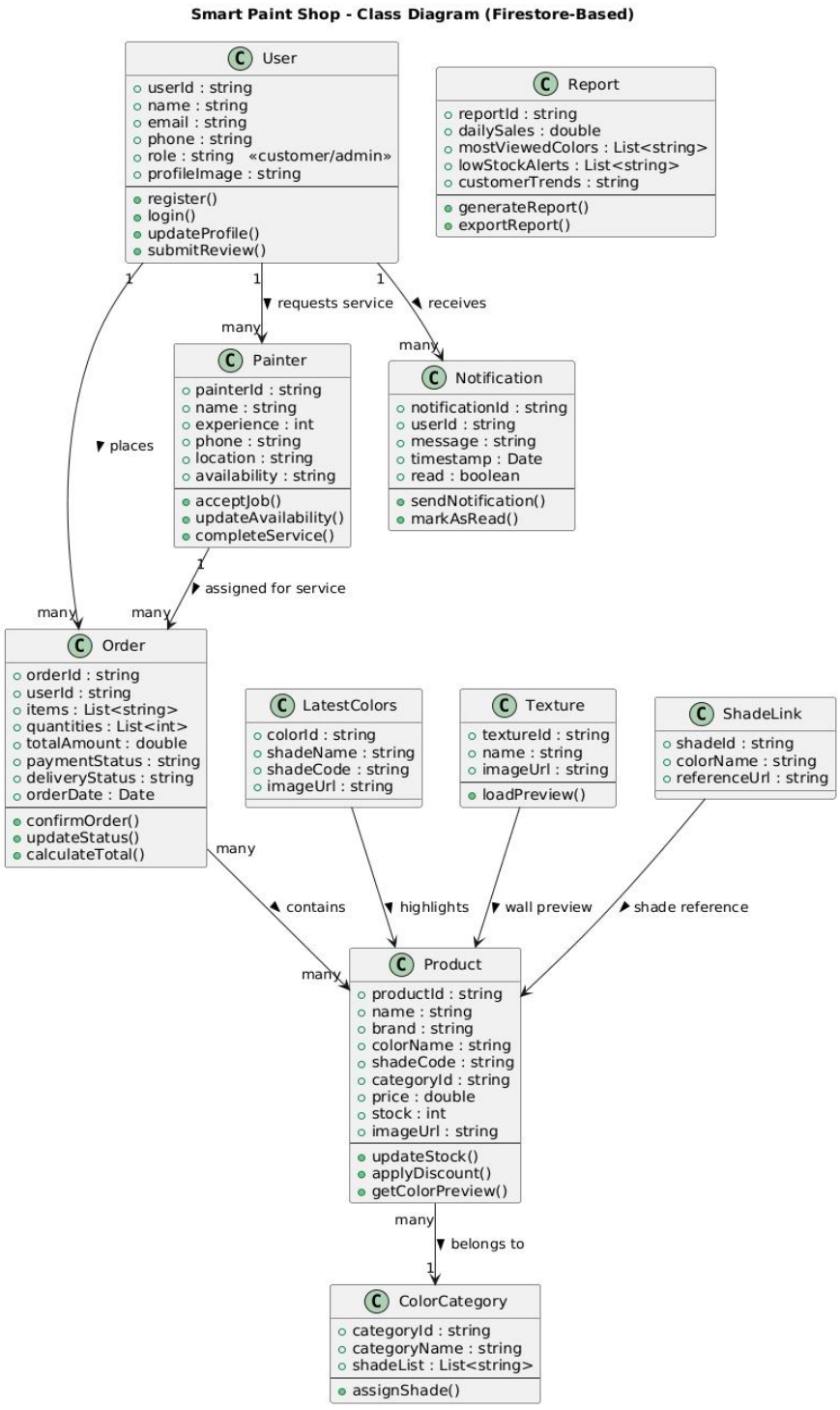


Fig 4.2.5 Class Diagram

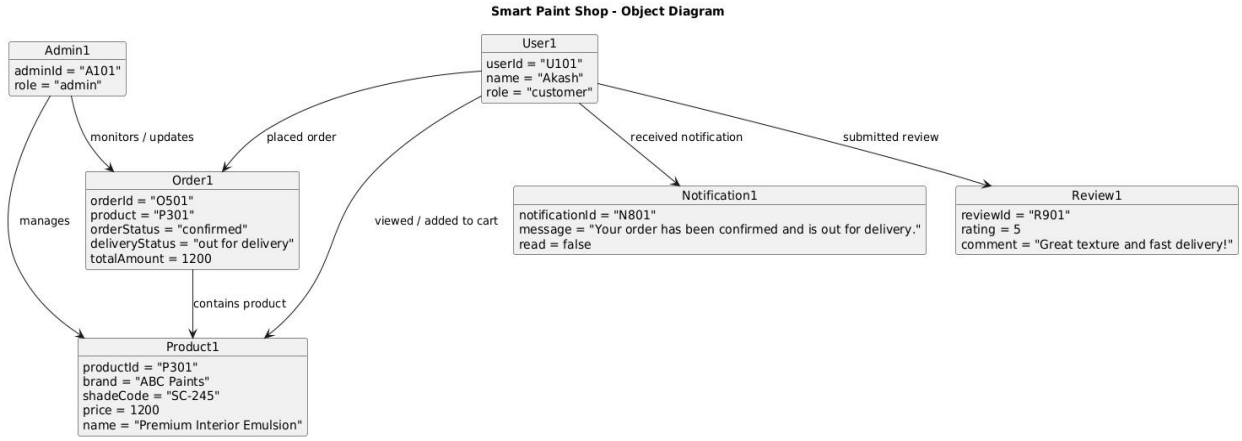
4.2.6 Object Diagram

The image displayed represents the Smart Paint Shop – Object Diagram, capturing a runtime snapshot of the system and showing how key instantiated objects interact within a typical customer journey. The diagram includes five main objects that reflect real entities managed in the application.

The Admin1 object (adminId = A101, role = "admin") oversees product-related and order-related actions, and is responsible for managing both the Product1 and Order1 objects. The User1 object (userId = U101) plays the central role in this scenario and is connected to all other operational objects in the system.

User1 has added Product1 (representing a paint item with shadeCode = “SC007”, brand = "Asian Paints", and price = 1200) to the cart and successfully placed Order1, which contains the selected product with orderStatus = "confirmed" and deliveryStatus = "out for delivery". The user has also received Notification1, which informs them that the order has been confirmed and assigned for delivery. After receiving the item, User1 submitted Review1, providing a rating of 5 and comment “Great texture and fast delivery”.

This object diagram clearly demonstrates the typical operational flow where a user browses a product, places an order, receives system updates, and finally submits feedback—all coordinated under administrative management.



4.2.6 Object Diagram

4.2.7 Component Diagram

The component diagram for the Smart Paint Shop – AI-Driven Paint Retail System illustrates the structural architecture and the interaction between major software components that power the application. At the center of the system lies the Backend API (Node.js + Express), which acts as the core controller responsible for processing business logic, managing requests, and orchestrating all service interactions. This Backend API directly interfaces with the Mobile Frontend (Flutter), the user-facing application used by customers, delivery partners, painters, and administrators.

The Backend API communicates with several backend services and data repositories. The primary persistent data is stored in Firebase Firestore, which holds collections such as users, products, orders, painters, textures, shadeLinks, notifications, and latestColors. This database ensures scalable, real-time data synchronization across all application layers.

To maintain modular and clean architecture, the system includes multiple specialized services. The Authentication Service handles user login, registration, and access control. The Order Processing Service manages cart operations, order placement, stock validation, payment status updates, and delivery tracking. The Product Management Service controls catalog operations, including product creation, updates, category assignments, and inventory monitoring.

The Painter Service Module processes customer requests for professional painters, manages painter availability, and allocates service appointments. Similarly, the Notification Service enables automated sending of delivery updates, promotional messages, order confirmations, and system alerts using Firebase Cloud Messaging (FCM).

Additionally, the Review Management Service captures and processes customer ratings and feedback on delivered products or painter services, contributing to analytics and product quality improvements. The system also integrates an Analytics & Reporting Component, which retrieves data from Firestore to generate sales insights, usage patterns, most-viewed colors, and inventory alerts for admins.
This component-based architecture ensures that Smart Paint Shop remains highly modular, scalable, and adaptable—allowing easy integration of future enhancements such as AR-based paint previews, AI-driven color recommendations, and real-time delivery tracking.

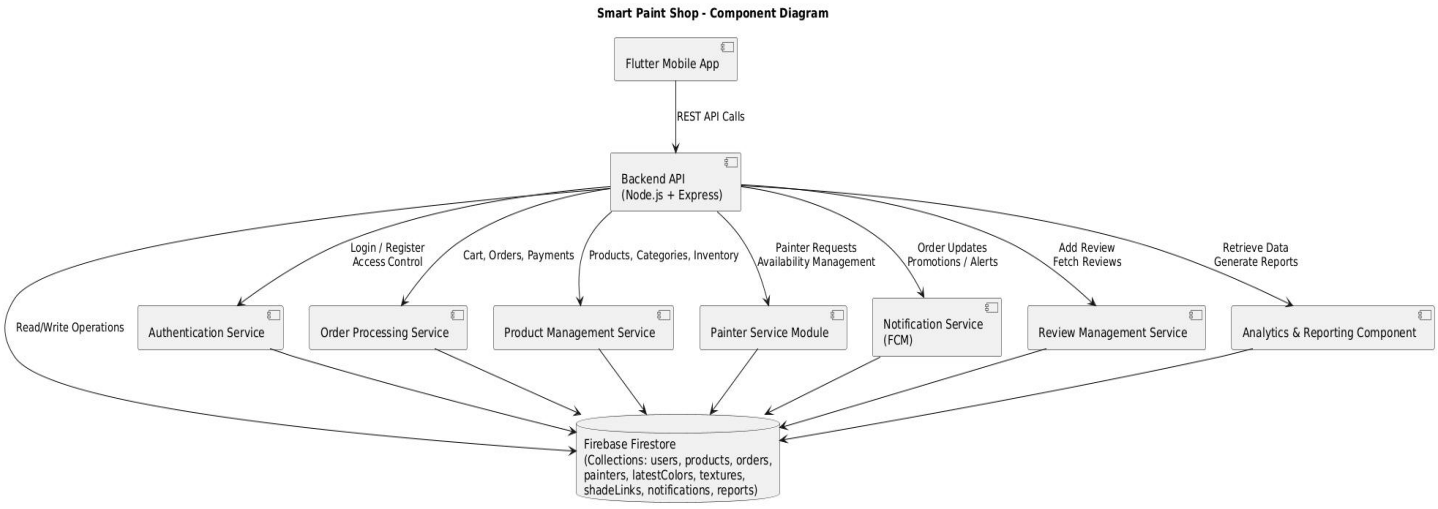


Fig 4.2.7 Component Diagram

4.2.8 Deployment Diagram

The image illustrates the Deployment Diagram for the Smart Paint Shop – AI-Powered Paint Retail System, outlining the physical architecture and communication flow between hardware and software components. The deployment environment is composed of four main nodes that work together to deliver a seamless user experience.

The Client Device (Android/iOS smartphone) hosts the Flutter Mobile App, which communicates with the system through secure HTTPS requests. These requests are directed to the Application Server, the central processing node of the architecture. The Application Server runs the Backend API (Node.js + Express), responsible for handling business logic, validating requests, and coordinating interactions with other backend modules.

Inside the Application Server, several key services operate as co-located components. These include the Authentication Service for user login and access control, the Order Processing Service for handling cart operations, payments, and order status updates, the Product Management Service for catalog and inventory operations, the Painter Service Module for managing painter availability and service requests, and the Review Management Service for collecting and processing user feedback.

The Backend API maintains a persistent, real-time Database Connection to the Firestore Database Server, which stores users, products, orders, textures, notifications, shadeLinks, painters, and other essential collections. Firestore's cloud-hosted structure ensures low-latency data synchronization across all users and services.
An additional node in the architecture is the Cloud Notification Server, used for delivering push notifications via Firebase Cloud Messaging (FCM). This node sends order confirmations, delivery updates, promotional alerts, and painter service notifications directly to user devices.

The Deployment Diagram highlights the system's cloud-centric, modular, and scalable architecture, supporting future enhancements such as AR-powered visualization, AI-based color recommendations, and real-time delivery tracking.

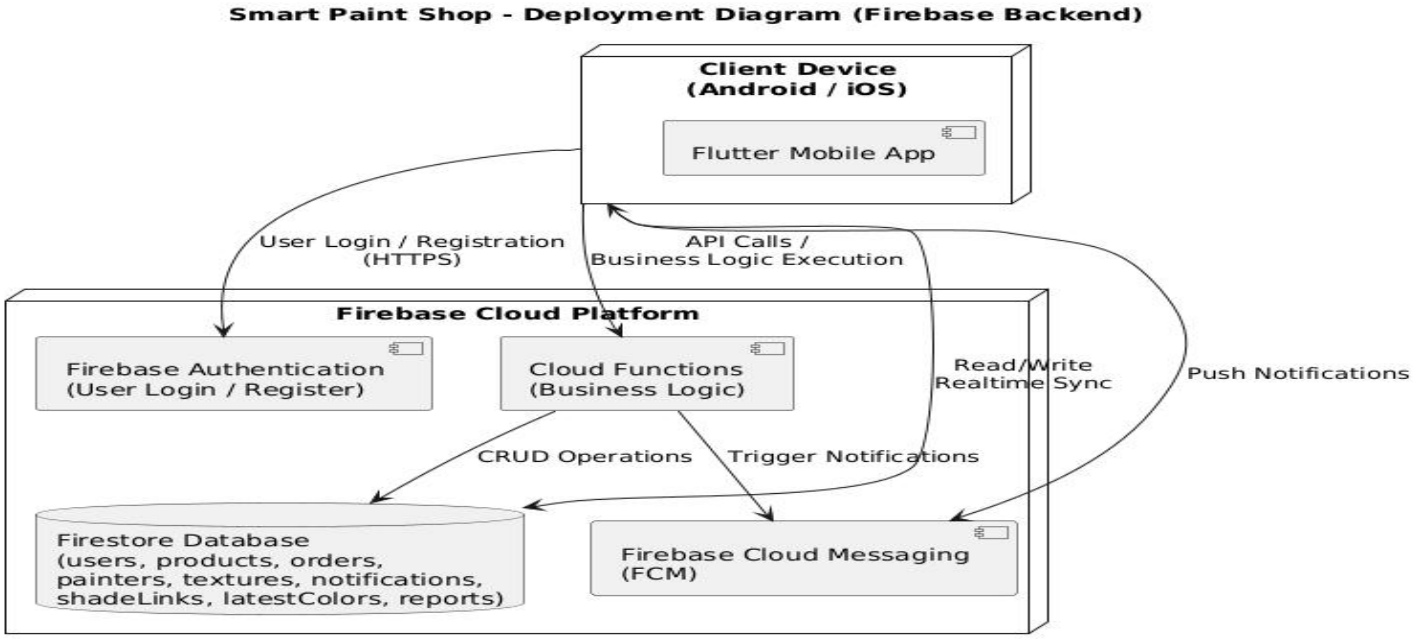
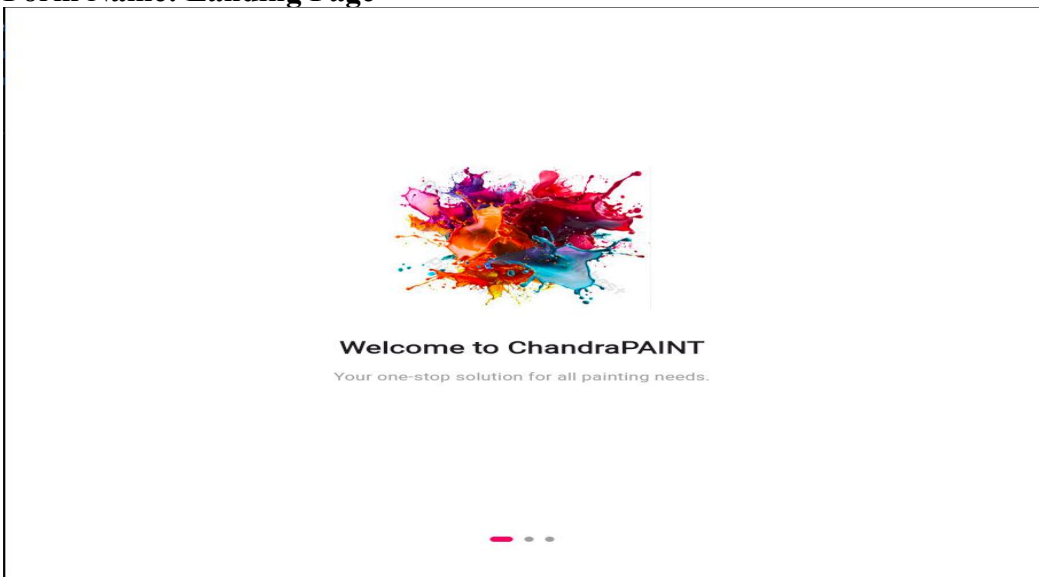


Fig4.2.8 Deployment Diagram

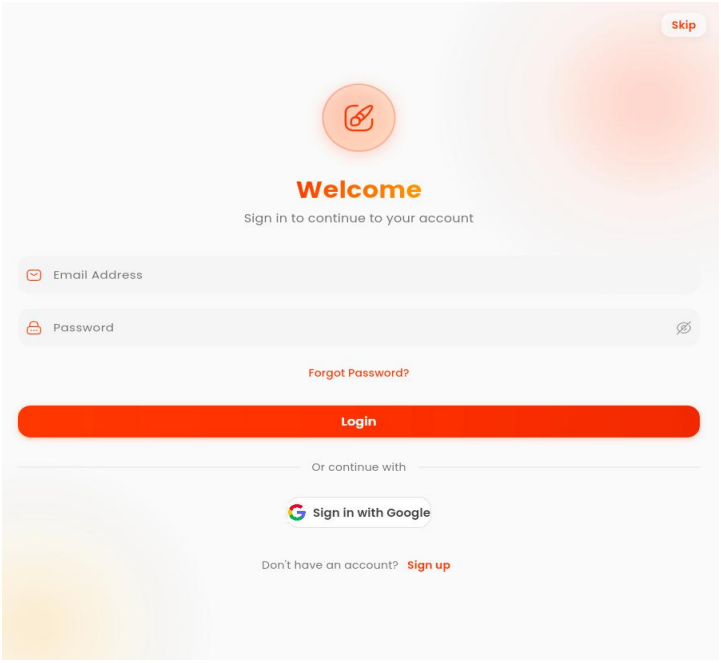
4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Landing Page



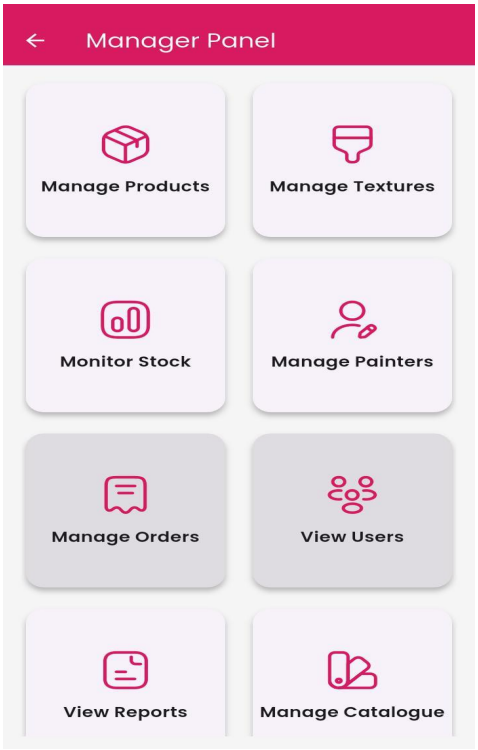
4.3.1 Landing Page

4.3.2 Form Name: Login



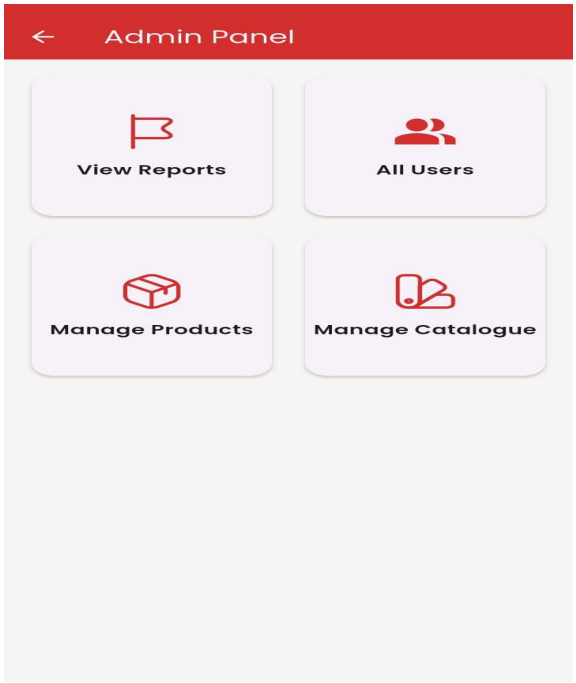
4.3.2. Login

4.3.3 Form Name: Manager Panel



4.3.3 Manager Panel

4.3.3 Form Name: Admin Panel



4.3.4 Admin Panel

4.4 DATABASE DESIGN

4.4.1 NoSQL Document-Based Design

Although the Smart Paint Shop utilizes NoSQL databases, specifically Firebase Firestore and Realtime Database, the design incorporates concepts related to a relational approach to ensure data consistency. Key logical relationships managed include linking a Customer's profile (userid) to a specific Order, linking Products to a customer's order, and managing information for services like Painters. These logical connections ensure that data is not isolated and that the system can accurately track a customer's journey from browsing the catalog to order completion

4.4.2 Normalization

Normalization principles are applied logically within the document-based Firebase database to reduce redundancy and improve data integrity. This is achieved by Minimizing Duplication; for instance, data entities like Product details (e.g., name, brand, description) are stored in their own products collection and are referenced by their ID in documents like a customer's Order, rather than duplicating the full product details in every order. This dedication to Data Consistency ensures that separate collections exist for core entities such as users, products, Painters, and orders, so that information like a product's stock level or a user's profile data is updated in a single, authoritative place.

4.4.3 Sanitization

Data Sanitization is critical to security and involves cleaning and validating user input before it is processed or stored in the database. This includes rigorous Input Validation for fields like user registration details (email, password), which is largely managed by Firebase Authentication. It also applies to data entry by Admins (e.g., adding new products) and Customers (e.g., placing orders) to ensure they adhere to the required format and type. Security Protection is paramount; using the Firebase SDKs for database operations helps mitigate NoSQL injection by ensuring inputs are treated strictly as data. Credential Security is handled directly by Firebase Authentication, which securely manages, hashes, and stores user passwords to protect user, admin, and manager access.

4.4.4 Indexing

Indexing is vital for increasing the efficiency of data retrieval, which is essential for the core functionality of the Smart Paint Shop system , such as browsing and visualization. To achieve Optimizing Search Queries, indexes are created on frequently queried fields. For the "Smart Paint Shop," essential indexes would include:

products records indexed by category, brand, or color to facilitate fast browsing and searching in the paint catalog.
orders records indexed by customer ID (userid) to allow users to track their order status

orders records indexed by date or status for managers to process orders and for admins to generate sales reports.
Painters records indexed by location (if applicable) to help users find local painter info.

4.5 TABLE DESIGN

1. User Collection (users)

Primary Key: **_id**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	userId	String	PK, Auto (Firebase Auth)	Primary Key (from Firebase Auth)
2	email	String	Required, Unique	User login (email format)
3	name	String	Required	Full name of the user
4	password	String	Handled by Firebase Auth	Hashed password (managed by service)
5	role	String	Required, Enum: customer, admin, manager	User role for app permissions
6	phone	String	Nullable	Contact phone number
7	address	Object	Embedded	User's default delivery address
8	profilePictureUrl	String	Nullable	URL to user's profile picture
9	wishlist	Array[String]	Default: []	Array of productIds
10	createdAt	Timestamp	Auto	Created timestamp

2. Product Collection (products)

Primary Key: **_id**

Foreign Key: userId references **_id**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key
2	name	String	Required	Product name (e.g., "Apex Ultima")
3	brand	String	Required	Brand name (e.g., "Asian Paints") [8]
4	description	String	Required	Detailed product description
5	price	Number	Required, Min: 0	Price per unit (e.g., per liter)
6	stockQuantity	Number	Required, Min: 0	Current inventory level [9999]
7	category	String	Required	e.g., "Interior", "Exterior", "Enamel"
8	imageUrls	Array[String]	Default: []	URLs to product images
9	colorCode	String	Nullable	Hex or brand-specific color code
10	textureId	String	FK (textures)	Reference to a texture document
11	createdAt	Timestamp	Auto	Created timestamp

3. Order Collection (orders)

Primary Key: **_id**

Foreign Key:**userId references User (userId)**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key
2	userId	String	FK (User), Required	Customer who placed the order
3	items	Array[Object]	Required	Array of {productId, name, quantity, price}
4	totalAmount	Number	Required, Min: 0	Total cost of the order
5	orderStatus	String	Required, Enum: pending, processing, shipped, delivered, cancelled	Current status of the order.
6	deliveryAddress	Object	Required, Embedded	Shipping address for this order.
7	paymentDetails	Object	Required, Embedded	{paymentId, method, status: 'completed'}
8	estimatedQuantity	Number	Nullable	Quantity estimated by the app's tool
9	orderDate	Timestamp	Auto	Created timestamp

4. Painter Collection (painters)

Primary Key: **_id**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key

2	name	String	Required	Painter's full name
3	contactNumber	String	Required, Unique, 10 digits	Contact phone number
4	location	String	Required	Primary service area (e.g., "Kanjirappally")
5	experienceYears	Number	Required, Min: 0	Years of professional experience
6	profileImageUrl	String	Nullable	URL to painter's photo
7	availability	Boolean	Default: true	Currently available for new projects
8	servicesOffered	Array[String]	Default: []	e.g., "Interior Painting", "Texture Work"
9	createdAt	Timestamp	Auto	Created timestamp

5. Review Collection (reviews)

Primary Key**: _id**

Foreign Key: **userId references User (userId), productId references Product (_id)**

No.	**Field Name**	**Data Type**	**Key Constraints**	**Description of the field**
1	_id	String	PK, Auto, Unique	Primary Key
2	userId	String	FK (User), Required	User who wrote the review
3	productId	String	FK (Product), Required	Product being reviewed

4	rating	Number	Required, Min: 1, Max: 5	Rating (1-5 stars)
5	comment	String	Nullable, Max: 500 chars	Review comment text
6	createdAt	Timestamp	Auto	Created timestamp

6. Texture Collection (textures)

Primary Key: **_id**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key
2	name	String	Required, Unique	Name of the texture (e.g., "Royal Play")
3	description	String	Required	Description of the texture effect
4	previewImageUrl	String	Required	URL to the texture image
5	brand	String	Required	Brand associated with the texture
6	createdAt	Timestamp	Auto	Created timestamp

7. Cart Collection (carts)

Primary Key: **_id**

Foreign Key: **userId references User (userId)**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key

2	userId	String	FK (User), Required, Unique	The user who owns this cart
3	items	Array[Object]	Default: []	Array of {productId, quantity}
4	lastUpdated	Timestamp	Auto	Timestamp of the last cart

8.Report Collection (reports)

Primary Key: **_id**

Foreign Key: **generatedBy references User (userId)**

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	String	PK, Auto, Unique	Primary Key
2	reportType	String	Required, Enum: sales, inventory, customer_behavior	Type of report
3	generatedBy	String	FK (User), Required	Admin/Manager who generated the report
4	data	Object	Required	The JSON/Object data of the report
5	dateRange	Object	Required	{startDate, endDate} for the report
6	createdAt	Timestamp	Auto	Created timestamp

9. Notification Collection (notifications)

Primary Key: **_id**

Foreign Key: **userId references User (userId)**

No.	Field Name	Data Type	Key Constraints	Description of the field

1	_id	String	PK, Auto, Unique	Primary Key
2	userId	String	FK (User), Required	The user who receives the notification
3	title	String	Required	Notification title (e.g., "Order Shipped")
4	message	String	Required	Notification body text
5	isRead	Boolean	Default: false	Read/unread status
6	type	String	Enum: order, promotion, system	Category of notification
7	createdAt	Timestamp	Auto	Created timestamp

10. **ColorCategory Collection (colorCategories)**

Primary Key**: _id**

No.	**Field Name**	**Data Type**	**Key Constraints**	**Description of the field**
1	_id	String	PK, Auto, Unique	Primary Key
2	name	String	Required, Unique	e.g., "Warm Neutrals", "Cool Blues"
3	description	String	Nullable	Description of the color family
4	hexCodes	Array[String]	Default: []	List of representative hex codes
5	isTrend	Boolean	Default: false	Marks this as a current trend
6	style	String	Nullable, Enum: modern, classic, vibrant	Associated style/theme

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

System Testing is the stage in the Smart Paint Shop project where the integrated system is evaluated against its defined requirements to ensure it operates correctly as a whole. The purpose is to validate that all components of the Flutter and Firebase architecture (Flutter Frontend, Firebase Backend services like Firestore, Realtime Database, and Authentication) function seamlessly together. This ensures the delivery of a smooth user experience, particularly in generating AI-powered color recommendations , product visualization , and order processing.

This phase is crucial to identify and rectify potential errors, integration mismatches, or logical flaws before deployment. It ensures the application fulfills user expectations and performs efficiently in real-world conditions.

Key focus areas of System Testing in the Smart Paint Shop project include:

• **End-to-End Workflow Validation:** Testing the complete flow from user registration, browsing the paint catalog , using the AI color visualizer , and completing a purchase, including checkout and payment.

• **AI Recommendation Accuracy:** Verifying that the recommendation engine correctly analyzes user input (like room images or themes) to produce relevant and accurate color and texture suggestions.

• **Security Testing:** Ensuring user credentials, personal data, and order history are securely handled, and proper authentication/authorization mechanisms are enforced via **Firebase Authentication** and Firestore Security Rules

• **Performance Testing:** Validating system responsiveness under multiple simultaneous operations, such as concurrent product queries, image uploads for visualization, or order processing.

• **Requirements Verification:** Confirming all functionalities defined in the project scope—like the **AI color suggestion tool** , **Professional Painter Management** , and the complete e-commerce checkout workflow —function as intended..

Successful System Testing provides confidence that the Smart Paint Shop is reliable, secure, and ready for deployment to its intended users—Customers, Managers, and Administrators .

5.2 TEST PLAN

The Test Plan for the Smart Paint Shop outlines the systematic approach to verifying the integrated system's core functionalities. It ensures the platform meets its objectives of delivering accurate AI-powered color visualizations and managing the complete e-commerce workflow.

Testing will cover all major aspects of the Flutter and Firebase stack integration—validating communication between the Flutter frontend, Firebase services (Firestore, Authentication, Cloud Functions), and Firebase Storage The plan includes multiple levels of testing.

A test case will be executed to validate the primary system functions, which include one login case and three core functionalities. The first case is User Role Authentication, which ensures a registered Customer, Manager, or Admin can successfully authenticate and access their respective dashboards . The second is the AI Color Visualization Flow, verifying that a user can upload an image, receive accurate AI-powered color suggestions, and preview combinations. The third, the E-commerce Order and Checkout Flow, validates the entire process from a customer browsing the product catalog, adding an item to the cart, and successfully completing the order with payment . The final critical case is Admin Product Management, which verifies that an Admin can successfully add, update, and manage paint products in the products collection, with changes reflected in the customer-facing app

The test environment will utilize the Flutter development environment (using emulators/simulators) and the Firebase project console. Testing will primarily be conducted through manual execution, with the option to use the automated Flutter Test framework for regression testing. A test will pass if the actual result matches the expected result and the system operates in accordance with the documented project requirements. Conversely, a test fails if the results do not match or a critical error occurs.

5.2.1 Unit Testing

Unit Testing is the initial and most granular phase of testing in the Smart Paint Shop project's development cycle. This testing technique involves validating the smallest testable parts of the application, known as 'units,' to ensure each operates correctly in isolation.

The primary purpose of Unit Testing is to verify the internal logical structure and functions of individual components before they are integrated into the larger system. For the Flutter and Firebase-based Smart Paint Shop project, this includes:

Backend Logic: Testing individual Firebase Cloud Functions (if implemented) or stand alone utility algorithms. This includes verifying the logic for the Quantity Estimation tool or any backend logic for processing orders.

Database Rules: Validating the Firebase Firestore Security Rules using the Firebase Emulator Suite. This ensures data is properly formatted and adheres to access constraints, such as checking that only an Admin can write to the products collection or that a customer can only create/update their own order document.

Frontend Components: Testing individual **Flutter widgets** using "widget tests". This ensures they render correctly and handle user input validation (e.g., verifying that the admin's "add product" form fields are filled out properly or that the AI color visualizer widget

responds to user input).

By completing Unit Testing successfully, the team ensures that the fundamental building blocks of the system are sound, which significantly reduces the probability of errors occurring later during the complex system-wide workflow

5.2.2 Integration Testing

Integration Testing is the phase that follows Unit Testing in the Smart Paint Shop project development cycle. Its purpose is to verify that different modules, services, or components of the system interact and communicate correctly when put together, ensuring the system works as a cohesive whole.

In the context of the Flutter and Firebase architecture, Integration Testing specifically focuses on verifying the flow and integrity of data and control between the separate layers:

Frontend-Backend Interface: This involves testing that the Flutter frontend correctly sends data (e.g., an image upload for visualization, a search query for products, or order submission data) to the Firebase Backend services (like Firestore, Cloud Functions, or Storage) and accurately handles the responses. For example, ensuring the user's color selection is correctly passed to the AI visualization logic and the result is displayed in the Flutter UI.

Backend-Database Interface: This verifies the data mapping and retrieval between the Flutter app and the Firebase Firestore database. A key test is confirming that when the system registers a new order, the data is correctly written to the orders collection and the total amount and items are accurately stored. Another example is confirming that product stock levels are updated correctly in the products collection after an order is completed.

Module-to-Module Communication: This ensures that data passed between separate modules of the application works as intended. For example, testing the transition where a product is added to the user's Cart, and upon clicking "Checkout," the data flows correctly to the Payment Gateway and then to the Order Collection upon success.

By performing Integration Testing, the project ensures that the distinct pieces of code, which functioned perfectly during Unit Testing, do not introduce errors or failures when working together across the entire system.

5.2.3 Validation Testing or System Testing

Validation or System Testing for the Smart Paint Shop evaluates the complete application as a unified, fully functional system. The objective is to verify that all integrated modules fulfill

the project's overall objectives and user expectations, particularly regarding the e-commerce workflow and AI visualization..

The testing validates:

• **End-to-End Functionality:** From a new Customer registering, browsing the Product Catalog, using the AI Color Suggestion tool, adding items to the Cart, and completing the full Checkout Flow with payment and order submission.

• **AI Recommendation Flow:** Ensuring the visualization service correctly processes user input (like image uploads or chosen themes) and returns logical, attractive, and accurate Color and Texture outputs for preview..

• **Requirement Compliance:** Confirming all listed project functionalities—such as the Professional Painter Management feature, the Admin's ability to generate reports, and the correct operation of all User Roles—work seamlessly.

• S**ecurity & Performance:** Testing session handling via Firebase Authentication, data integrity, and system responsiveness when handling concurrent operations like multiple product searches or order submissions.

This phase guarantees that the Smart Paint Shop delivers an accurate, reliable, and

secure e-commerce and visualization platform to all its real-world users.

5.2.5 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) in the Smart Paint Shop validates the system from the perspective of real users, specifically Customers, Managers, and Administrators, to ensure usability, clarity, and accuracy across all user roles.

• **Usability Testing:** Verifying that the interfaces (product catalog, AI color visualization tool, and secure checkout process) are intuitive, responsive, and easy for the customer to navigate, leading to a smooth shopping experience.

• **Real-World Scenarios:** Simulating actual user journeys—such as a Customer using the AI tool to select a color, getting a Quantity Estimation, and completing an order; a Manager updating inventory; and an Admin generating a sales report

• **Requirements Fulfillment**: Confirming that all project objectives (AI-driven color suggestion, a full e-commerce capability, and backend shop management tools) function as designed and meet the practical needs of the paint retail business.

Successful UAT ensures that the Smart Paint Shop is not only technically sound but also practically efficient, meeting the needs of all user types in a real-world e-commerce environment.

5.2.5 Automation Testing

Automation Testing in the Smart Paint Shop primarily utilizes the built-in Flutter Test framework to automate repetitive tasks and ensure consistent system behavior across updates, significantly improving the efficiency of the development cycle.

Automation will be primarily applied to:

• **Regression Testing:** Regression Testing: Ensuring that new feature additions (like new product categories, payment gateway updates, or advanced texture features) do not inadvertently affect core system stability, such as the checkout process or user login.

• **High-Volume Testing:** Automatically simulating multiple concurrent product searches, order placements, or image uploads for the visualization tool to test the application's scalability and performance under load.

By automating these repetitive tests, the Smart Paint Shop achieves faster validation cycles, higher test accuracy, and reduced manual effort—ensuring that the system remains robust as it evolves.

5.2.6 Flutter Testing

Flutter Testing is a core component of the testing strategy for the Smart Paint Shop, given its development in the Flutter framework. It focuses on verifying both the UI and functional behavior of the app in a simulated or real mobile environment, ensuring a smooth and consistent cross-platform user experience. Flutter provides three main levels of testing: unit tests, widget tests, and integration tests, all managed through the built-in Flutter Test framework.

Applications

The Flutter Test framework is ideal for testing the app’s user interfaces, navigation flows, and Firebase backend connectivity:

• Automating User Journeys: Flutter integration tests simulate real user interactions such as logging in, browsing the product catalog, using the AI color visualizer, and completing the order checkout. These automated flows ensure the app behaves as expected under real-world e-commerce scenarios without manual input.

• Regression Testing: When new modules or features (like advanced texture previews or admin report generation) are added, Flutter tests confirm that previously working features—such as user login via Firebase Auth, product data retrieval from Firestore, and order submission—continue to function correctly.

• UI Consistency and Behavior Verification: Widget tests validate that every component (e.g., product cards, cart summary, color picker, and payment forms) renders correctly and reacts properly to user input, ensuring a consistent and reliable user experience across devices.

• Backend and API Validation: Integration tests simulate network calls to Firebase to ensure that Flutter communicates correctly with the backend services (e.g., fetching product details, saving order status, or uploading an image to Firebase Storage), validating both response handling and data flow.

Benefits to the Project

By adopting the Flutter Testing framework, the Smart Paint Shop project ensures:

• Faster identification of bugs before release.

• Reliable performance across Android and iOS platforms.

• Reduced manual testing effort through automated workflows.

• Higher app stability and improved end-user satisfaction.

Example:

Test Case

Code

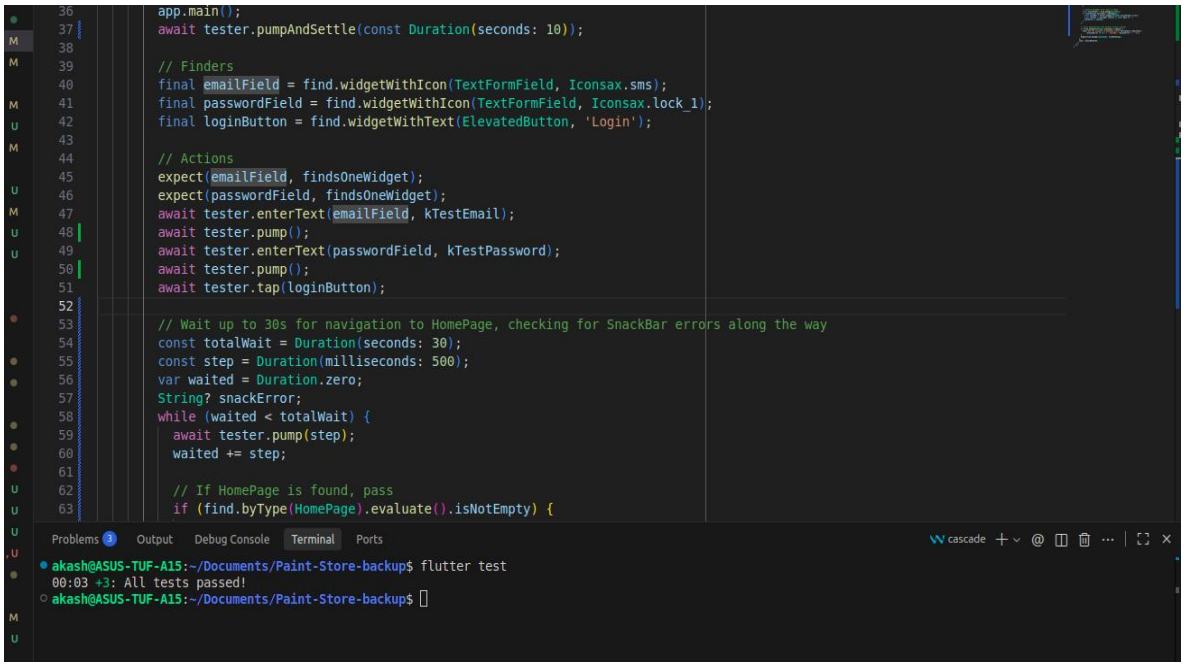
```
void main() {
  // Initialize Flutter integration test binding
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  group('E-commerce Flow: Cart Management', () {
    testWidgets('SPS_TC_04: Successfully add product to cart and verify count update', (tester) async
{
      // 1. Launch the app and simulate navigation to the main Catalog view
      app.main();
      await tester.pumpAndSettle();

      // --- Pre-condition: User is already logged in and on the Home/Catalog screen ---
      // Assuming a widget with a key 'home_screen' exists after successful login
      expect(find.byKey(const Key('home_screen')), findsOneWidget);
      await tester.pumpAndSettle();

      // 2. Locate a specific product item in the catalog
      final productName = find.text('Asian Paints Apex Ultima');
      expect(productName, findsOneWidget);

      // 3. Find the 'Add to Cart' button (e.g., identified by a specific icon)
      // Search for the button within the parent card/tile of the product name
      final productCard = find.ancestor(of: productName, matching: find.byType(Card));
      final addToCartButton = find.descendant(
          of: productCard, matching: find.byIcon(Icons.add_shopping_cart));
      expect(addToCartButton, findsOneWidget);

      // 4. Verify initial cart count (Header icon badge)
      final cartIconBadge = find.byKey(const Key('cart_count_badge'));
      // Expect the badge to initially show 0 (or be absent if count is 0)

      // 5. Tap 'Add to Cart'
      await tester.tap(addToCartButton);
      await tester.pumpAndSettle(const Duration(seconds: 2)); // Wait for state change, network call,
and animation

      // 6. Verify success notification (e.g., Snackbar)
      expect(find.text('Product added to cart!'), findsOneWidget);

      // 7. Verify the cart count badge has incremented to '1'
      // This confirms the UI update and successful data write to the 'carts' collection in Firebase
      expect(find.descendant(of: cartIconBadge, matching: find.text('1')), findsOneWidget);

      // 8. Tap the cart icon to navigate to the Cart Summary screen
      await tester.tap(cartIconBadge);
      await tester.pumpAndSettle();

      // 9. Verify the Cart Summary screen loads and contains the product
      expect(find.text('Cart Summary'), findsOneWidget);
      expect(productName, findsOneWidget); // The product name should be visible on the cart page
    });
  });
```

}

Eg.Test Report

Project Name	**Smart Paint Shop: E-commerce and Visualization Platform**
Test Case ID	**SPS_TC_04**
Test Designed By	Akash Krishna
Test Priority	**High**
Test Designed Date	07-10-2025
Module Name	Product Catalog / Cart Management
Test Executed By	Akash krishna
Test Title	Verify Product Selection and Addition to Cart Flow
Test Execution Date	14-11-2025
Description	Validate that a registered user can select a product from the catalog, successfully add it to their shopping cart, and ensure the cart's item count updates correctly in the UI. This verifies the successful write operation to the carts collection in Firebase.
Condition	User is logged in as a **Customer**. The Product Catalog is visible and loaded with items from the products collection. Internet connection is active, and Firebase services are running.
Step	**Test Step**
1	Navigate to the Product Catalog screen.
2	Locate a visible product item.
3	Note the initial Cart Count in the header badge.
4	Tap the "Add to Cart" button for the selected product.
5	Verify the Cart Count badge update.
6	Tap the Cart icon to navigate to the Cart Summary.
Post-Condition	The selected product is successfully persisted in the user's carts document in Firebase, and the correct quantity is reflected in the Cart Summary screen.

Eg.Screenshot



CHAPTER 6
IMPLEMENTATION

6.1 INTRODUCTION

The Implementation Phase of the Smart Paint Shop Application involves transforming the detailed system design into a fully functional and deployable mobile e-commerce platform. This phase translates planned features and workflows into actual code and integrates the various components to create a seamless user experience. The implementation focuses on developing and integrating the primary modules — Customer, Manager, and Admin — and supporting components such as the AI Visualization tool, Product Catalog, and Order Management systems.

The development is executed using the selected Flutter + Firebase (Firestore, Authentication, Storage) technology stack. The Flutter frontend provides a responsive and user-friendly mobile interface for both customers and staff, while Firebase manages real-time database logic (Firestore), user authentication, and secure file storage (Storage) for images. Core functionalities developed during this phase include user registration and login, the AI-powered color suggestion and visualization, full e-commerce checkout flow, and data storage/retrieval.

Successful implementation results in a complete and deployable system, ready for rigorous testing and real-world use. This phase marks the realization of the Smart Paint Shop project's mission — to create a reliable, efficient, and intelligent mobile solution that simplifies the paint purchasing process and automates essential retail operations.

6.2 IMPLEMENTATION PROCEDURES

The implementation of the Smart Paint Shop project follows a systematic approach, ensuring the successful construction, integration, and deployment of all modules. The procedure covers software development, integration testing, user training, and maintenance planning for the Flutter–Firebase architecture.

6.2.1 User Training

User training ensures that all intended users can effectively operate the new system and fully utilize its features, catering to the three distinct user roles.

• **Training on Application Software:** For Customers, training emphasizes navigation through the Flutter interface, using the AI Color Suggestion tool, browsing the Product Catalog, and completing the secure E-commerce Checkout Flow.

For Managers, training includes managing the Point of Sale (POS) operations (if applicable), tracking daily inventory, and processing incoming orders.

For Admins, training covers managing the Product Inventory (adding/updating items), monitoring sales and order fulfillment, and generating Business Reports and Analytics.

• **Training Focus:** The sessions highlight how the Smart Paint Shop simplifies the

traditional paint buying process — emphasizing AI-driven accuracy, faster ordering, and ease of visualization. Users learn how the mobile application enhances efficiency in color selection, order tracking, and shop management.

6.2.2 Training on the Application Software

Training is tailored to match each user group's responsibilities within the Smart Paint Shop ecosystem, ensuring they can fully benefit from the app's capabilities.

Training for Admins and Managers

Training for Blood Bank personnel is vital, as they manage the system's core operations:
• Inventory and Product Management: Admins learn to use the dedicated forms to manage the products collection—adding new items, updating stock, and setting prices.
• Order and Fulfillment: Managers learn to monitor and update the orderStatus in the orders collection, process shipments, and handle customer communication.
• System Monitoring: Admins are guided on accessing administrative dashboards, viewing sales analytics, and managing the Painter Information module.

.**Training for Customers**

• Visualization Tools: Users are guided on uploading room images to the Firebase Storage and utilizing the AI Color Suggestion feature to select paints and textures.
• E-commerce Workflow: Training focuses on the end-to-end shopping process
• from adding items to the cart, applying discount codes, to completing secure payment via the integrated gateway.
• Profile and Order Tracking: Users are shown how to view their order history and track the delivery status of their active orders.
Overall, training ensures that all users can seamlessly interact with the app, understand its automation features, and adapt to its improved workflow compared to manual systems.

6.2.3 System Maintenance

System Maintenance is an ongoing activity aimed at preserving the reliability, performance, and scalability of the deployed Smart Paint Shop application. It ensures the system continues to function optimally in response to user feedback, security updates, and environmental changes.

6.2.4 Maintenance Procedures

• **Monitoring and Optimization:** Continuous observation of Flutter performance (e.g., widget build times) and Firebase services (Firestore read/write limits, Storage usage) to detect slow data loading or performance bottlenecks, ensuring a responsive mobile experience.
• **Security and Updates:** Routine updates to Flutter SDK dependencies, third-party libraries, and continuous management of Firebase Security Rules and Firebase Authentication to safeguard against vulnerabilities and maintain system stability.

• **Corrective Maintenance**: Timely debugging and resolution of user-reported issues or system anomalies, such as order processing errors or visualization tool bugs, to minimize downtime.
• **Adaptive Maintenance:** Modifying the system to stay compatible with new mobile OS versions (Android/iOS) and adapting to changes in integrated external services like payment gateways or AI libraries.
• **Perfective Maintenance:** Refining user experience, improving the interface for the AI Visualization tool, enhancing the order tracking feature, and optimizing data retrieval for faster catalog browsing.

By maintaining a proactive approach, the Smart Paint Shop ensures long-term sustainability, delivering a consistent, efficient, and secure e-commerce experience

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Smart Paint Shop – Integrated E-commerce and Visualization Platform project successfully completed its Mini Project Phase, proving the feasibility and effectiveness of its core objective: to create a smart, user-centric paint retail platform that promotes efficient buying, visualization, and inventory management through a seamless mobile experience. Developed using the Flutter and Firebase technology stack (Firestore, Authentication, Storage), the system ensures fast performance, real-time data synchronization, and scalability across users and staff devices.

The project has efficiently implemented its key modules—AI Color Visualization, Multi-Brand Product Catalog, secure E-commerce Checkout, Order Tracking, and Admin/Manager Dashboards—all working together to deliver an intuitive and comprehensive paint shopping solution. Through secure user management via Firebase Authentication and well-structured database collections (such as users, products, and orders), the Smart Paint Shop has laid a robust technical foundation for future enhancements.

The successful completion of system testing and validation confirms that all major features operate cohesively, meeting both functional and non-functional requirements. The app's design prioritizes usability, performance, and customer engagement—enabling users to confidently select and purchase products while providing managers and administrators with meaningful insights into stock levels and sales statistics.

In conclusion, the Smart Paint Shop has achieved its initial objectives by delivering a functional, reliable, and engaging e-commerce platform that redefines how users purchase paint and how retailers manage their inventory. The project establishes a strong base for future innovation, including advanced modules for AR-based Visual Simulation, automated Stock Tracking, and Advanced Sales Analytics, positioning the Smart Paint Shop as a modern and scalable digital retail ecosystem.

7.2 FUTURE SCOPE

The Smart Paint Shop – Integrated E-commerce and Visualization Mini Project has successfully delivered a stable Flutter and Firebase stack foundation and validated core operational workflows, but its full potential is realized in the planned expansion to a full-scale, intelligent paint retail automation system. The future scope is heavily defined by integrating advanced visualization technology, machine learning, and specialized retail management tools to enhance both customer experience and operational efficiency.

A critical planned addition is AR and AI Integration, which will enable the system to perform AR-based Visual Simulation. This allows customers to point their mobile device at a room wall and instantly preview selected colors or textures in real-time. Furthermore, the system will integrate a Smart Inventory Module that enables Stock Tracking with Supplier Alerts to proactively notify managers when stock levels drop below a pre-set threshold, ensuring

continuous product availability.

Finally, the scope includes comprehensive administrative and financial control, moving beyond simple order tracking. This involves implementing an Advanced Sales Analytics Module to track product performance, calculate profit margins, and generate detailed Business Reports for managerial decision-making. Technologically, the platform will introduce a dedicated Customer Service Chat Module to enable real-time messaging between users and the store staff, ensuring rapid support and issue resolution.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

Sharma, R., & Gupta, A. (2022). *AI-Enabled Fitness Tracking and Health Monitoring Systems: A Comprehensive Review*. *International Journal of Computer Applications in Health Technologies*, 15(2), 45-52.

WEBSITES:

Google Developers. (2024). *Flutter Documentation*. Retrieved from https://flutter.dev/docs

Pal, N. (2023). *Full Stack Mobile App Development with Flutter and Flask: A Complete Guide*. Packt Publishing.

Flask Project. (2024). *Flask: Web Development, One Drop at a Time*. Retrieved from https://flask.palletsprojects.com/

Firebase. (2024). *Firebase Authentication and Cloud Firestore Documentation*. Retrieved from https://firebase.google.com/docs

Stack Overflow Community. (2023). *Best Practices for Integrating Flutter with Flask Backend APIs*. Retrieved from https://stackoverflow.com

GeeksforGeeks. (2023). *Developing Cross-Platform Apps Using Flutter and REST APIs*. Retrieved from https://www.geeksforgeeks.org

CHAPTER 9
APPENDIX

9.1 Sample Code

```
import 'package:c_h_p/admin/add_product_page.dart';
import 'package:c_h_p/model/product_model.dart';
import 'package:c_h_p/product/edit_product_page.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:iconsax/iconsax.dart';
import 'package:cached_network_image/cached_network_image.dart';

// Ensure the class name matches your file name and usage
class ManageProductsPage extends StatefulWidget {
  const ManageProductsPage({super.key});

  @override
  State<ManageProductsPage> createState() => _ManageProductsPageState();
}

class _ManageProductsPageState extends State<ManageProductsPage> {
  final DatabaseReference _dbRef = FirebaseDatabase.instance.ref('products');
  final TextEditingController _searchController = TextEditingController();
  String _searchQuery = '';

  void _showDeleteDialog(String key, String name) {
    showDialog(
      context: context,
      builder: (ctx) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
        title: const Text('Confirm Deletion'),
        content: Text('Are you sure you want to delete "$name"? This action cannot be undone.'),
        actions: [
          TextButton(onPressed: () => Navigator.of(ctx).pop(), child: const Text('Cancel')),
          FilledButton(
            style: FilledButton.styleFrom(backgroundColor: Colors.red.shade700),
            onPressed: () {
              _dbRef.child(key).remove();
              Navigator.of(ctx).pop();
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text('"$name" has been deleted.'), backgroundColor: Colors.red),
              );
            },
            child: const Text('Delete'),
          ),
        ],
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey.shade100,
      appBar: AppBar(
```

```
      title: Text("Manage Products", style: GoogleFonts.poppins(fontWeight: FontWeight.bold, color: Colors.grey.shade800)),
      backgroundColor: Colors.white,
      elevation: 1,
      iconTheme: IconThemeData(color: Colors.grey.shade800),
    ),
    body: Column(
      children: [
        // --- SEARCH BAR ---
        Padding(
          padding: const EdgeInsets.all(16.0).copyWith(bottom: 0),
          child: TextField(
            controller: _searchController,
            decoration: InputDecoration(
              hintText: 'Search by product name...',
              prefixIcon: const Icon(Iconsax.search_normal_1),
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(12),
                borderSide: BorderSide.none,
              ),
              filled: true,
              fillColor: Colors.grey.shade200,
            ),
            onChanged: (value) => setState(() => _searchQuery = value.toLowerCase()),
          ),
        ),
        // --- PRODUCT LIST ---
        Expanded(
          child: StreamBuilder(
            // Order by name for consistency
            stream: _dbRef.orderByChild('name').onValue,
            builder: (context, AsyncSnapshot<DatabaseEvent> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const Center(child: CircularProgressIndicator(color: Colors.deepOrange));
              }
              if (snapshot.hasError) {
                return Center(child: Text("Error loading products: ${snapshot.error}"));
              }
              if (!snapshot.hasData || snapshot.data?.snapshot.value == null) {
                return const Center(child: Text("No products found. Add one to get started!"));
              }

              final productsMap = Map<String, dynamic>.from(snapshot.data!.snapshot.value as Map);


              final List<Product> allProducts = productsMap.entries.map((entry) {
                try {
                  return Product.fromMap(entry.key, Map<String, dynamic>.from(entry.value));
                } catch (e) {
                  debugPrint("Error parsing product ${entry.key} for manage page: $e");
                  // Return a dummy product or handle error appropriately
                  return Product(key: entry.key, name: "Error Parsing", description: "", stock: 0, mainImageUrl: "", backgroundImageUrl: "", benefits: [], packSizes: [], brochureUrl: "");
                }
              }).toList();
```

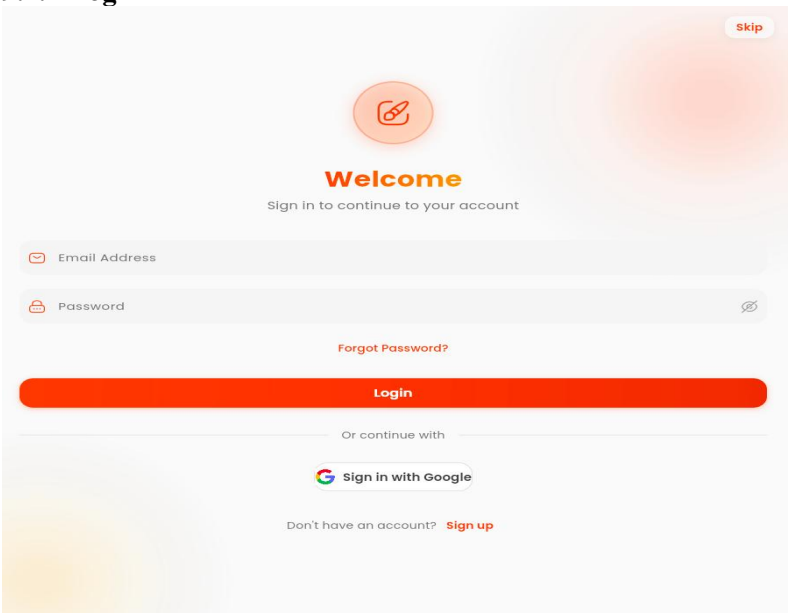
```
            // Filter based on the Product object's name
            final filteredProducts = allProducts.where((product) {
              return product.name.toLowerCase().contains(_searchQuery);
            }).toList();

            if (filteredProducts.isEmpty) {
              return const Center(child: Text("No products match your search."));
            }

            return ListView.builder(
              padding: const EdgeInsets.fromLTRB(16, 16, 16, 100), // Padding includes FAB space
              itemCount: filteredProducts.length,
              itemBuilder: (context, index) {
                final product = filteredProducts[index];
                // Still need the raw map for the potentially un-updated EditProductPage
                final productData = Map<String, dynamic>.from(productsMap[product.key] as Map);
                return _buildProductCard(product, productData); // Pass both product and raw data
              },
            );
          },
        ),
      ),
    ],
  ),
  floatingActionButton: FloatingActionButton.extended(
    onPressed: () {
      Navigator.push(context, MaterialPageRoute(builder: (context) => const AddProductPage()));
    },
    backgroundColor: Colors.deepOrange,
    icon: const Icon(Iconsax.add, color: Colors.white),
    label: Text("Add Product", style: GoogleFonts.poppins(color: Colors.white, fontWeight:
FontWeight.w600)),
  ),
 );
}

Widget _buildProductCard(Product product, Map<String, dynamic> rawProductData) {
 int stock = product.stock;
 Color stockColor;
 if (stock == 0) {
  stockColor = Colors.red.shade700;
 } else if (stock <= 10) {
  stockColor = Colors.orange.shade800;
 } else {
  stockColor = Colors.green.shade800;
 }

 // Get the price of the first pack size to display
 final priceToShow = product.packSizes.isNotEmpty ? product.packSizes.first.price : 'N/A';

 return Card(
  margin: const EdgeInsets.only(bottom: 16),
  elevation: 2,
  shadowColor: Colors.black.withValues(alpha: 0.1),
```

```
shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
child: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(
    children: [
      Row(
        children: [
          SizedBox(
            width: 70,
            height: 70,
            child: ClipRRect(
              borderRadius: BorderRadius.circular(10),
              child: Container(
                color: Colors.white,
                child: AspectRatio(
                  aspectRatio: 1,
                  child: CachedNetworkImage(
                    //      Use mainImageUrl from Product object
                    imageUrl: product.mainImageUrl,
                    fit: BoxFit.contain,
                    fadeInDuration: const Duration(milliseconds: 160),
                    memCacheWidth: 220,
                    memCacheHeight: 220,
                    placeholder: (context, url) => Container(color: Colors.grey.shade200),
                    errorWidget: (context, url, error) => const Center(child: Icon(Iconsax.gallery_slash)),
                  ),
                ),
              ),
            ),
          ),
          const SizedBox(width: 12),
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  //      Use name from Product object
                  product.name,
                  style: GoogleFonts.poppins(fontWeight: FontWeight.bold, fontSize: 16),
                ),
                const SizedBox(height: 4),
                Text(
                  //      Use brand from Product object
                  "Brand: ${product.brand ?? 'N/A'}",
                  style: TextStyle(color: Colors.grey.shade600, fontSize: 13),
                ),
              ],
            ),
          ),
        ],
      ),
      const Divider(height: 24),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

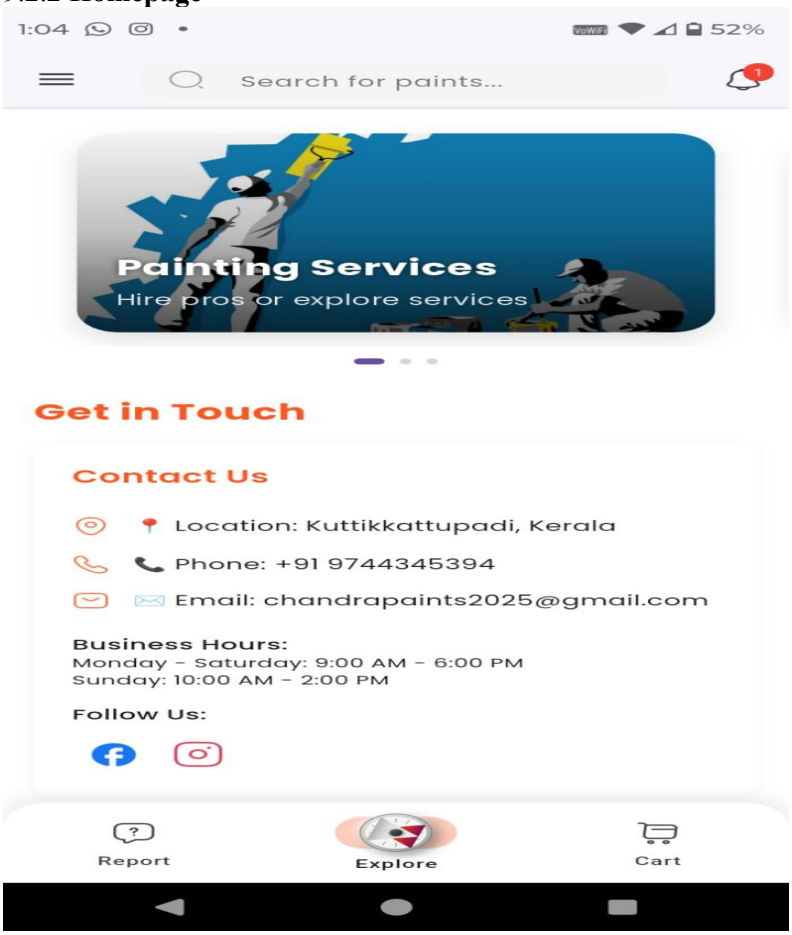
```
            children: [
              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text("Stock: $stock", style: TextStyle(fontWeight: FontWeight.bold, color: stockColor)),
                  //      Display starting price from pack sizes
                  Text("MRP : ₹$priceToShow", style: TextStyle(color: Colors.grey.shade700)),
                ],
              ),
              Row(
                children: [
                  IconButton(
                    icon: const Icon(Iconsax.edit, color: Colors.blue, size: 20),
                    onPressed: () => Navigator.push(
                      context,
                      // Pass the raw data map to EditProductPage
                      MaterialPageRoute(builder: (_) => EditProductPage(productKey: product.key,
productData: rawProductData)),
                    ),
                    tooltip: 'Edit Product',
                  ),
                  IconButton(
                    icon: const Icon(Iconsax.trash, color: Colors.red, size: 20),
                    //      Use key and name from Product object
                    onPressed: () => _showDeleteDialog(product.key, product.name),
                    tooltip: 'Delete Product',
                  ),
                ],
              ),
            ],
          ),
        ],
      ),
    ),
  );
 }
}
```

9.2 Screen Shots

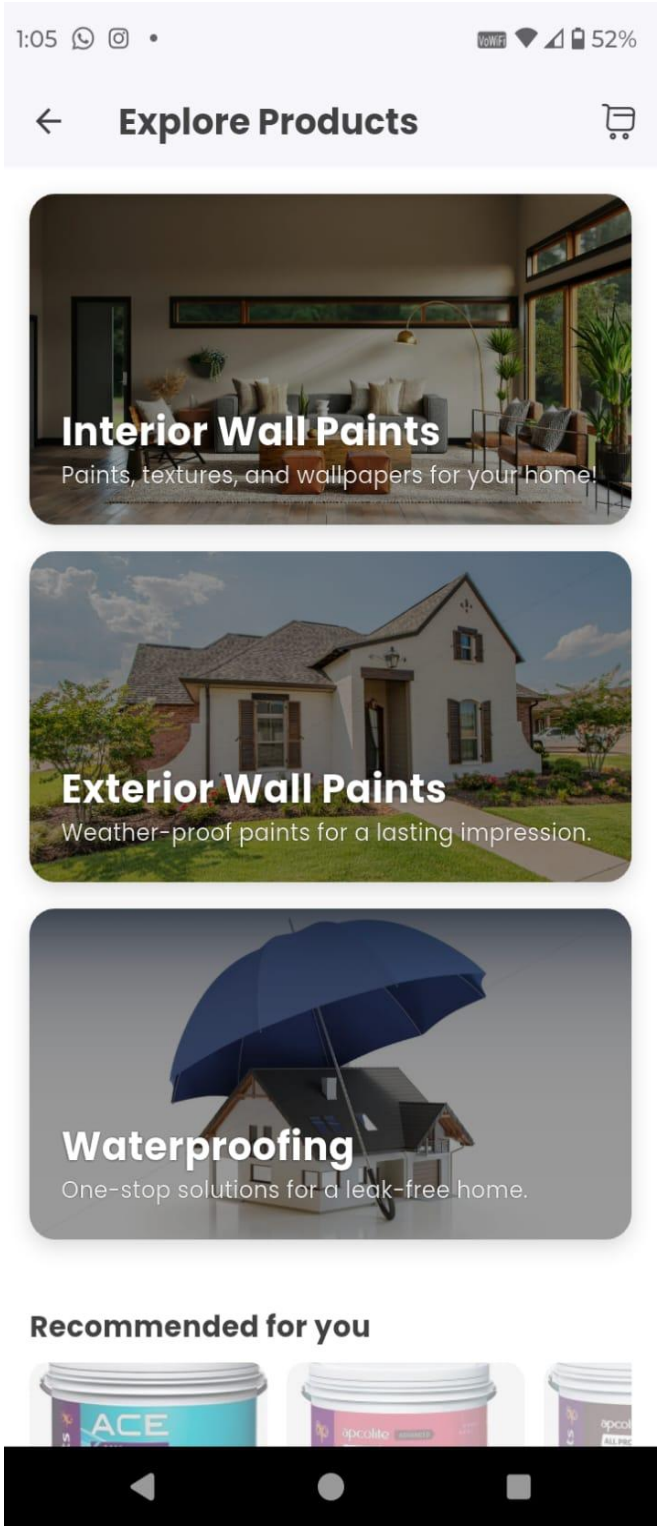
9.2.1 Login



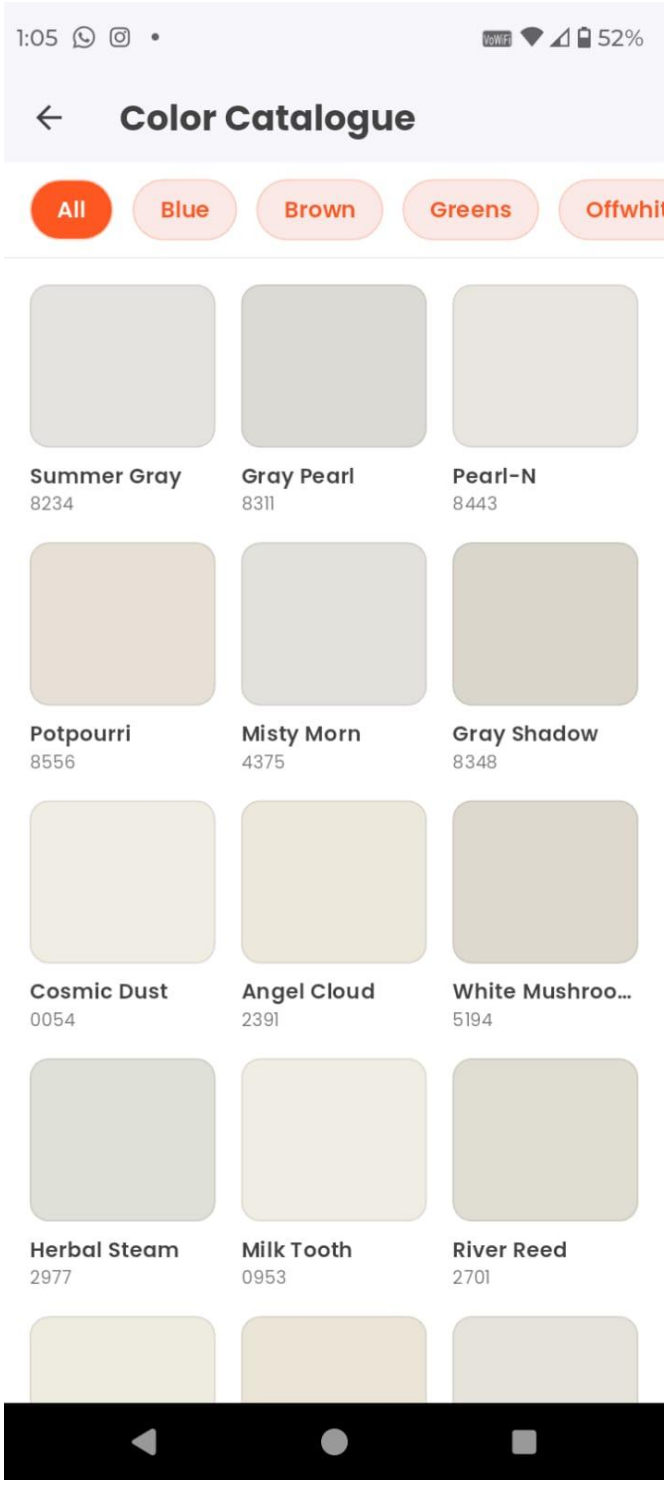
9.2.2 Homepage



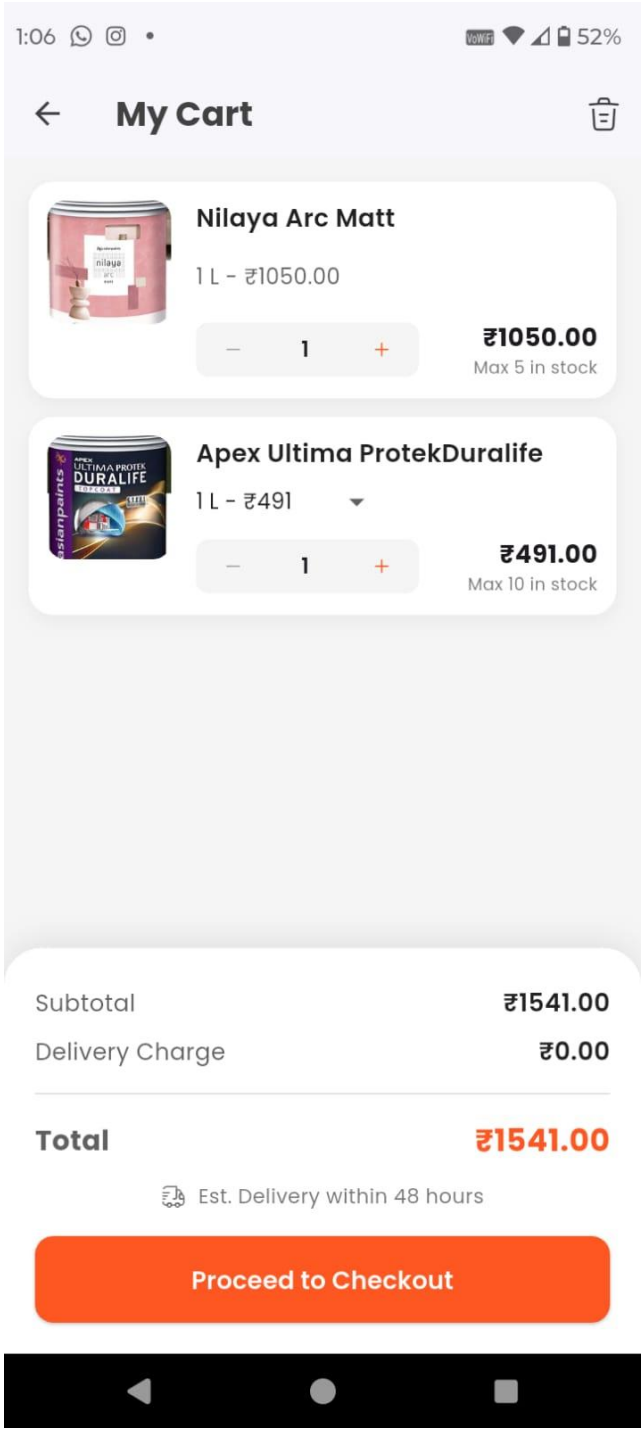
9.2.3 Explore Product



9.2.4 Color Catalogue Page



9.2.5 Cart Page



9.3 Git Log

akash@ASUS-TUF-A15:~/StudioProjects/c_h_p$ git log
commit b4d41207a593415feb83505e67317e3150856f96 (HEAD -> backup, origin/backup)
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Fri Oct 24 12:41:48 2025 +0530

new backup folders

commit ca3e31f5f168a108d09656724d5e9248c7376c18 (origin/main, main)
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Tue Oct 21 15:28:49 2025 +0530

Added textures and also updated cart

commit a4ab1e9002743c27a545006ff463200b322e635d
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Thu Sep 25 15:26:35 2025 +0530

major update on the product, added interior and exterior classes

commit 091abf5072bc4eed13e5066cc21a3501337cb3d8
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Thu Sep 18 08:57:32 2025 +0530

Aug 19, 2025, 2:04 PM

akash@ASUS-TUF-A15:~/Documents/Paint-Store-backup$ git log
commit cf948f9d8b104cbb6db42627e92f41a2dd54ec9e (HEAD -> backup, origin/master, origin/backup, master)
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Sun Oct 26 08:59:25 2025 +0530

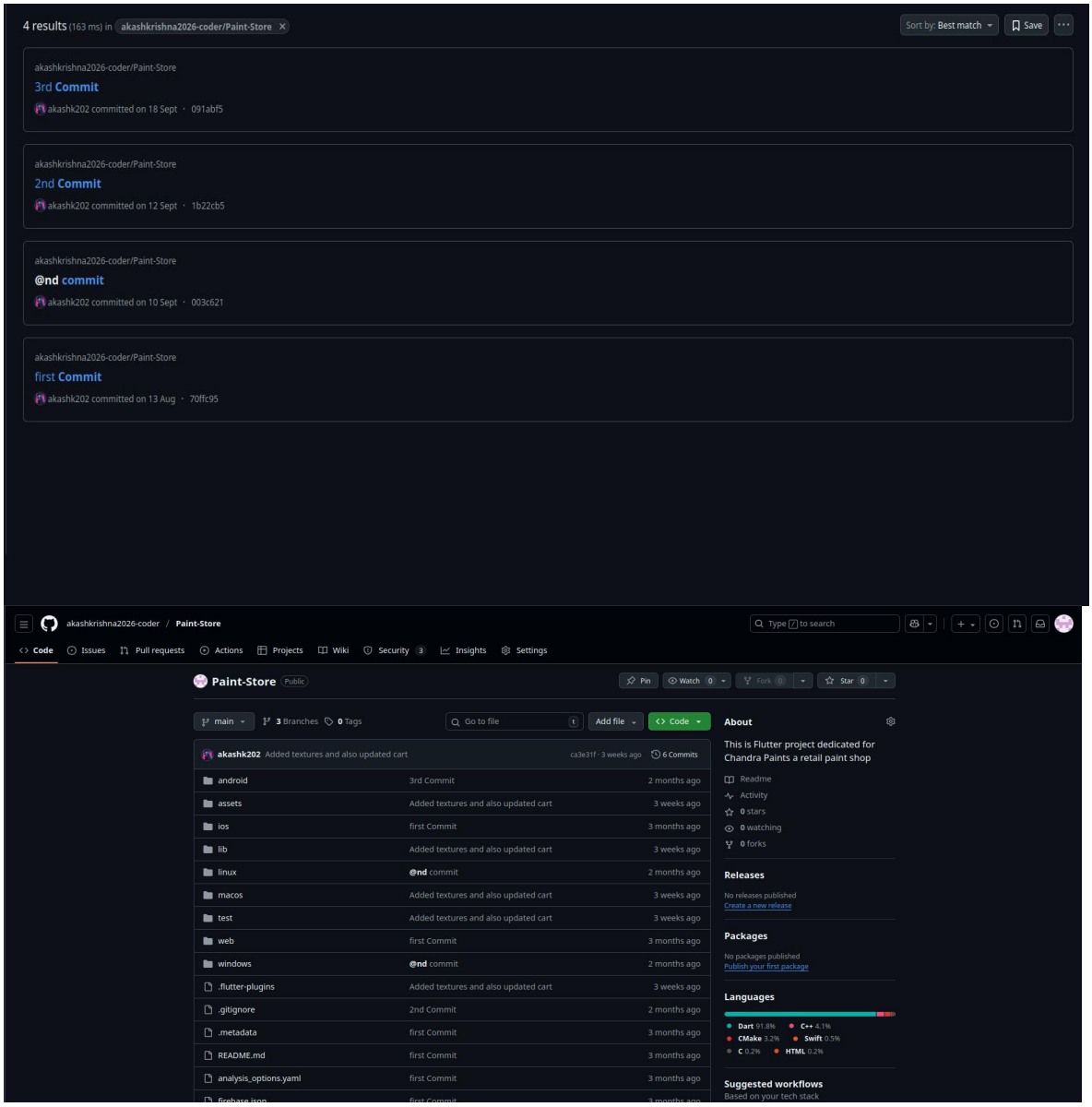
Backup till paint calculator

akash@ASUS-TUF-A15:~/Documents/Paint-Store-backup$ git log
commit c4f06ed838be8141b99da696385cf3976f8119a0 (HEAD -> main, origin/backup, backup)
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Mon Dec 1 11:51:32 2025 +0530

feat(product): Product pages is upgraded and new features is added

commit 321d70ca74360bcf0bad11f0323d2f318c231633
Author: Akash Krishna <akashkrishna389@gmail.com>
Date: Mon Dec 1 11:49:19 2025 +0530

build(deps): upgraded some dependencies


4 results (163 ms) in akashkrishna2026-coder/Paint-Store
Sort by: Best match
Save
akashkrishna2026-coder/Paint-Store
3rd Commit
akashk202 committed on 18 Sept · 091abf5
akashkrishna2026-coder/Paint-Store
2nd Commit
akashk202 committed on 12 Sept · 1b22cb5
akashkrishna2026-coder/Paint-Store
@nd commit
akashk202 committed on 10 Sept · 003c621
akashkrishna2026-coder/Paint-Store
first Commit
akashk202 committed on 13 Aug · 70ffc95
akashkrishna2026-coder / Paint-Store
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security 3
Insights
Settings
Paint-Store
Public
Pin
Watch 0
Fork 0
Star 0
main
3 Branches
0 Tags
Go to file
Add file
Code
akashk202 Added textures and also updated cart
ca3e31f · 3 weeks ago
6 Commits
android 3rd Commit 2 months ago
assets Added textures and also updated cart 3 weeks ago
ios first Commit 3 months ago
lib Added textures and also updated cart 3 weeks ago
linux @nd commit 2 months ago
macos Added textures and also updated cart 3 weeks ago
test Added textures and also updated cart 3 weeks ago
web first Commit 3 months ago
windows @nd commit 2 months ago
.flutter-plugins Added textures and also updated cart 3 weeks ago
.gitignore 2nd Commit 2 months ago
.metadata first Commit 3 months ago
README.md first Commit 3 months ago
analysis_options.yaml first Commit 3 months ago
About
This is Flutter project dedicated for Chandra Paints a retail paint shop
Readme
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
Dart 91.8%
C++ 4.1%
CMake 3.2%
Swift 0.5%
C 0.2%
HTML 0.2%
Suggested workflows
Based on your tech stack