Final Year Project

DreamSaver

Project Team:

Naqash Sachwani

2212230

Naveed Asad Ali

2212232

Project Supervisor:

Dr. Syed Samar Yazdani

August 5th, 2026

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science
in the

Faculty of Computing and Engineering Sciences

Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology University

(SZABIST University) Karachi Campus

Declaration of Authorship

We, the undersigned, hereby declare that the project titled *"DreamSaver"* is our original work and has been completed in accordance with the academic standards of SZABIST University under the supervision of *Dr Syed Samar Yazdani*.

We confirm that all sources and references used during the research and development of this project have been properly cited and acknowledged. The content of this report is original and has not been copied or plagiarized from any external source.

We further declare that this project has not been previously submitted for the award of any degree, diploma, or other qualification at any other university or institution.

Name: Naqash Sachwani

Signature: _____

Date: 5th August 2026

Name: Naveed Asad Ali

Signature: _____

Date: 5th August 2026

Project Description

The need of DreamSaver came from one common thing that everyone faces having to do in everyday life, saving enough money to purchase something big, but then finding out the price has increased by the time you have the money. We applied the proven layaway concept in a new, modern web app. The main aspect in the platform is the "Price Lock. The main aspect of the platform is the ‘Price Lock'. If an individual begins a savings objective for a particular item, the app "freezes" that market price. If inflation occurs or the store raises the price a month later, the customer can be assured the deal will keep the same price on day one.

The grinding of consumers money over the course of time was a huge priority for building trust. To deal with this, we've integrated Stripe to establish a safe escrow system. DreamSaver is a secure middleman in place of paying installment payments directly to the seller at once. All provisions remain in the platform's escrow environment until the client reaches 100% of savings goal. This eliminates the financial downside of the buyer, and ensures that the seller receives a paid-out sale once the sale is complete.

What is happening behind the scene is that the version of the app is a complete ecommerce stage with four different portals, one for the customers, one for the stores, one for the riders and one for the Admins. Customers can see where they are on their savings journey while store owners keep an eye on products inventory and manage incoming layaways from their dashboard. As soon as the customer's goal has been fully funded, the store can send the product to our logistics network. As part of the app, the map is used for tracking Riders.

Acknowledgement

In the name of ALLAH, the Most Beneficent, the Most Merciful Who endowed us with the capability, wisdom and strength to pursue and complete the "DreamSaver" project. We would like to thank Dr. Syed Samar Yazdani, our supervisor, from the Department of Computer Science at SZABIST for her tireless support, knowledge and encouragement over the course of our work. His expert guidance played a pivotal role in assisting us with various technical aspects, from conceptualizing the development of a full-fledged e-commerce and logistics platform to its final deployment.

We would also like to express our gratitude to all the esteemed teachers who have been great supporters of our development in our studies and who have helped us with this final year's project, with confidence and clarity. We also would like to thank our dear parents and relatives for your love, patience and prayers. It took numerous hours of hard work and development of a complex digital layaway and escrow system, but over the years we have found it to be a great source of strength and inspiration, in knowing that we have their backing and their support.

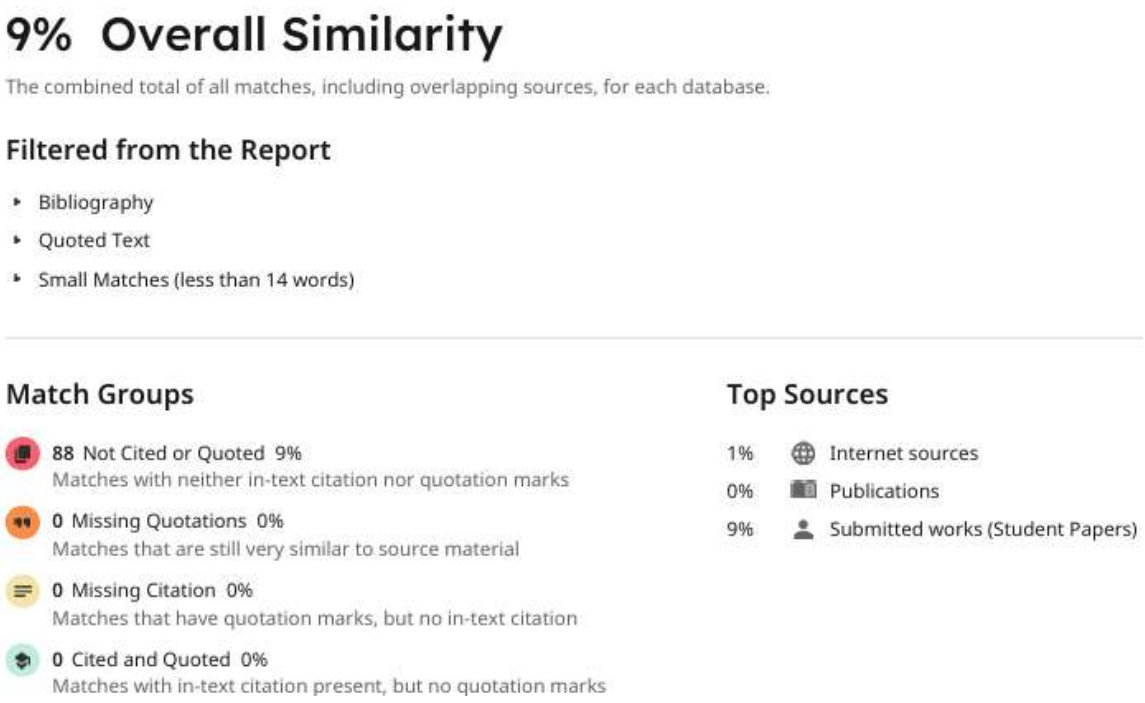
Last but not least, we are thankful to SZABIST for creating a supportive academic atmosphere and for lending us the opportunities we need to make this vision a reality. Special thanks to peers, mentors and early testers for their immensely valuable feedback on the price-lock, escrow, and multipurpose features on the platform. You were a key part to the development of DreamSaver being a safe, easy-to-use system; without your contributions this project would not have been able to happen.

Plagiarism-Free Certificate

This is to certify that the project titled **"DreamSaver"** has been successfully completed and submitted by the undersigned students. The project was carried out under the supervision of **Dr Syed Samar Yazdani** at **SZABIST University**

We hereby affirm that the content of this report is entirely original and has not been copied or reproduced from any external source. All references and materials used in the research and development of this project have been properly cited and acknowledged.

This project fully complies with the academic and ethical standards of **SZABIST University.** Furthermore, we declare that this work is our own and has not been submitted elsewhere for the award of any degree or qualification.



Name: Naqash Sachwani

Registration #: 2212230

Signature: _____

Name: Naveed Asad Ali

Registration #: 2212232

Signature: _____

Supervisor: Dr. Syed Samar Yazdani

Signature: _____

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Table of Contents

Table of Figures

Table of Tables

Project Proposal

1. Introduction

The effort to save up for a particular purpose is quite difficult in Pakistan wherein most people have no bank account with goal-oriented saving facilities or they save up in an informal manner at home. Many families start spending money too early, whether it's for a weddings, furniture's, education and so on, and this is why they are not ready when their time comes. It is a common practice in an informal way that many people have used for a long time of "layaway" whereby one party makes a deposit with a vendor until the item or products are purchased. But a digital, transparent and structured platform for this practice is not available in Pakistan.

Layaway is an attempt to digitize the layaway concept into a web app where the user registers their savings plan for acquiring a product, deposits a small amount of these savings (daily/weekly/monthly) and accesses the product only after all the money is collected. There is no debt, no interest and no product delivery until savings are done as they are not measured in installment systems. Stores are embedded into the system, which allows a direct linkage of their savings with a real product.

2. Objective

- To offer an online space for the users to set a saving plan for certain products.
- To let users top up with small amounts that can be made digitally to achieve their goals.
- To create a program for stores to list products.
- To build digital receipts for every operation to create trust and transparency.
- To provide dashboards to both consumers and retailers to see savings and progress.
- To help promote financial discipline from an impetuous spending.

3. Problem Description

In Pakistan, many have a lack of effective financial instruments of disciplined saving. There are tools for budgeting and expense tracking but they're not goal-focused or set into a store. Impulsive withdrawals from traditional home savings, and additional debt and financial stress added by installment plans. The Layaway platform will tackle these challenges by:

- Allowing users to set personalized savings goals (e.g., weddings education).
- Allowing for micro-deposits to save in an easy, small steps way.
- Standing up and providing dashboards, notifications to promote financial discipline.
- Integrating stores where saving is linked with real products.

This system incorporates financial management and individual consumer objectives to provide a safe, open and culturally responsive remedy to the issue of saving discipline. This system brings together financial management and the individual consumer objectives in a secure, transparent, and culturally relevant manner that mitigates saving discipline problems.

4. Target Industry and Application Areas

Therefore, the e-commerce industry in Pakistan is targeted considering physical products like electronics, household appliances, fashion, furniture, and lifestyle goods. Contrary to other

online stores like the Daraz, PriceOye and Telemart, who use the model of shopping and buying right away, our platform will be based on a layaway system where in which the customers put in money in smaller quantities and in turn receive the product right after the customers have reached their monetary target.

This method directly takes the halfway measure of the Pakistani consumers who are unwilling to take the intermediary of taking interest in the instalment system and are restricted in buying products based on inflation. The platform introduces a savings-to-purchase approach without charging interest, in adherence with the Sharia principles, to make e-commerce more accessible for the middle-class and unbanked population.

Weddings & Events:

- Jewelry, furniture, home appliances, decoration and electronics needed for marriages of bridal dresses.
- As families begin to save months ahead of time, they can also lock in products at a lower rate, rather than being impacted by inflation increases.

Electronics & Gadgets:

- Accessories, Mobile phones, Laptop and Smart TV.
- High-cost products like that can be obtained without a loan for young professionals and students.

Education & Work Essentials

- Laptops, Tablets, Study tables and office chairs.
- Assist students, freelancers and remote employees for purchases.

Sports, Fitness & Health Equipment

- Fitness machines, bicycles, and technology.
- Customers can make healthy buy without hurting their finances.

5. Methodology

To implement the development of the DreamSaver, an iterative model for web development known as Software Development Life Cycle (SDLC) will be used. The project will be executed over two phases (FYP-I and FYP-II), in both phases of which the new increment will be based on and expand the last phase. NextJs will be used to develop the backend to be sure that everything is scalable and transactions can be carried out efficiently. React will be used in the front end to provide dynamic and responsive user experience. PostgreSQL will be the key database for storage of transactions, user data and structured and secure records of Store.

Authentication will be provided via Clerk and secure role-based access to be provided for Users, Store, Riders and Administrators. The system will include the use of email for notifications and reminders, providing a mechanism for more users to be engaged and disciplined in their financial matters. Developed through payment gateway APIs, debit/credit card payment will be integrated.

Another key aspect of the methodology is to be gender transparent and trust-based, creating an invoice for every transaction. A series of short development cycles will be followed along with frequent testing; Agile principles will be used. By taking this approach, you can easily identify any emerging risks and address them early on, before they become an issue with the store, with pilot tests and utilizing secure APIs.

6. Project Scope

The project will aim to develop a platform for online layaway collections, where users can:

- Users to register, create saving goals, and make deposits.
- Register, list products and receive payments while storing.
- Admin users to be monitored based on users, stores, and transactions.
- It has dashboards, both for users, riders and stores, for monitoring progress.
- Secure and transparent record keeping thanks to digital invoice.
- Riders to receive delivery assignments

7. Comparative Analysis

Feature	Daraz	PriceOye	Telemart	DreamSaver
Payment Options	Cash, Credit, Debit, BNPL	Cash, Card, Installments	Cash, Bank Transfer	Layaway Plans (3–12 months)
Customer Risk	High (BNPL debt & impulse buying)	Medium (Price comparison focus)	Medium (Limited payment options)	Low (No debt & price locked)
Store Risk	High (15%+ commissions, returns)	Medium (Price competition)	High (Limited customer)	Low (5–8% commission, fewer returns)
Price Protection	No	No	No	Yes (Full price lock guarantee)
Interest Charges	Yes (BNPL, credit cards)	Sometimes (Instalment plans)	No	No (Interest-free)
Product Availability	Immediate delivery	Immediate delivery	Immediate delivery	Reserved after goal completion
Target Customer	Impulse buyers	Price-sensitive shoppers	Tech enthusiasts	Planned shoppers, budget-conscious
Return Policy	7–14 days easy returns	Varies by Store	Limited returns	Manufacturing defects only
Commission Fees	10–20%	8–15%	10–18%	5–8%
Cash Flow	Immediate Store payment	7–14 day payment cycle	Variable	No staged payments

Goal-based Savings	No	No	No	Yes (Customers save gradually towards purchase)
Store Integration	Basic Store onboarding system	Limited (focused on price listings)	Direct Store partnerships	Strong integration with layaway

Table 1: Comparative Analysis

8. Feasibility Study

- **Risks Involved:** Payment gateway integration, Store adoption, Trust issue.
- **Mitigation:** Use of secure APIs, pilot testing with select Stores, strong encryption.
- **Resource Requirements:** Web app, APIs for payment & notifications, database hosting, web hosting, domain name, laptop/PC.

9. Scientific Areas/ Applications Covered

- People who are planning to buy a house, send their children to school or go on a holiday.
- Revenue-generating financial institutions seeking to sell products that are not dependent on credit. Retailers interested in providing debt-free installments options.
- Ensuring data privacy and security.
- Creating a seamless user experience.

10. Emerging Technologies Used

- Frontend: ReactJS
- Backend: NextJs
- Database: PostgreSQL
- Payment Gateways: APIs
- Other Tools: GitHub, VS Code, Vercel, Inngest

11. CRUD Features

- User Management - Create accounts, view profiles, update details, delete accounts.
- Savings Goals - Create new goals, read/view goals, update goal details, delete goals.
- Transactions - Add deposits, view history.
- Stores – Register stores, update stores details, delete stores.
- Products – Add products, view catalog, update product details, delete inactive products.
- Rider Management – Record rider data, Check on delivery activity and earnings, Update rider data, Deactivate rider accounts..

12. Non-CRUD Features

- Progress Tracking – Report percentage of savings progress and deadline.
- Notifications – Notifications through email and in-application.
- Digital Receipts
- Reports – Saving reports.
- Security – User, Store, Rider and Admin role-based access.
- Refund Option – Get money back for cancelled goals.
- Map Integration – For tracking of deliveries.
- Support & Complaints

- Digital Wallet & Coupons

13. Correlation with Industrial Practices and Trends

The student's work will be checked for correlation with Industrial Practices and Trends (13). The platform is at par with the current global trends and practices of Ecommerce. Online layaway and goal-based saving apps are on the rise as ways consumers save to purchase items without debt and interest worldwide. Today, saving towards a goal is available on a number of FinTech apps, such as Chime and Revolut, where small savings can be set up and reminders emailed. Now Layaway has emulated and localized all these features for Pakistani people with notifications on the FinTech platforms.

The first consecutive entrance of digital payments, wallet and e-commerce platforms into Pakistan's industry marks a good change in the field of financial inclusion and trust-building. An SBP market report indicates that 60% of the digitally educated customer segment would be interested in interest-free payment options and hence the concept of a layaway system is aptly suited. Furthermore, with the Store integration model, you can have as many or as little risks as you want - and have the business pay for part or all of the stock before it is fulfilled. The system is additionally constructed with comparable technology to other prime e-commerce systems like PriceOye and Daraz via developing notifications, payment APIs, digital invoices and dashboards.

Overall, the suggested system is very micro-savings, cashless deposit, consumer protection, interest-free purchase, Store onboarding automation and price-lock - which makes the project highly relevant and ready for the future.

14. Expertise of the Team Members

All the team members are experienced in both database and website development. The team have attended courses on Database Systems and Web Programming, providing the necessary background. The project is of common interest to all the members.

15. Milestones

FYP-I:

- Project Setup & Requirement Analysis
- Admin and Store Management
- Product Management
- Savings Goals and Dashboards
- Progress Tracking and Reports

FYP-II:

- Transaction and Payment Integration
- Refund and Cancellation Requests
- User Management
- Notifications
- Digital Receipts and Map Integration
- Support & Complaints
- Digital Wallet & Coupons
- Rider Management

16. Project Schedule

Figure 1: Gantt Chart – FYP I

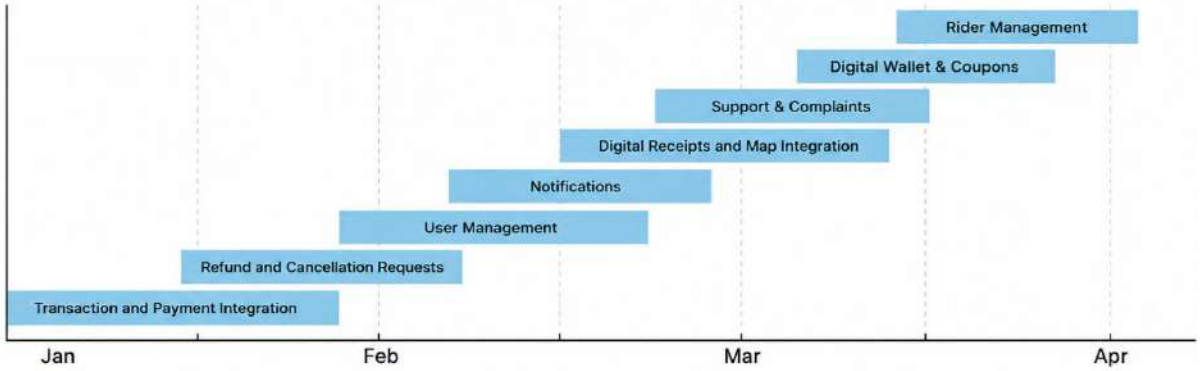


Figure 2: Gantt Chart – FYP II

17. Work Breakdown Structure

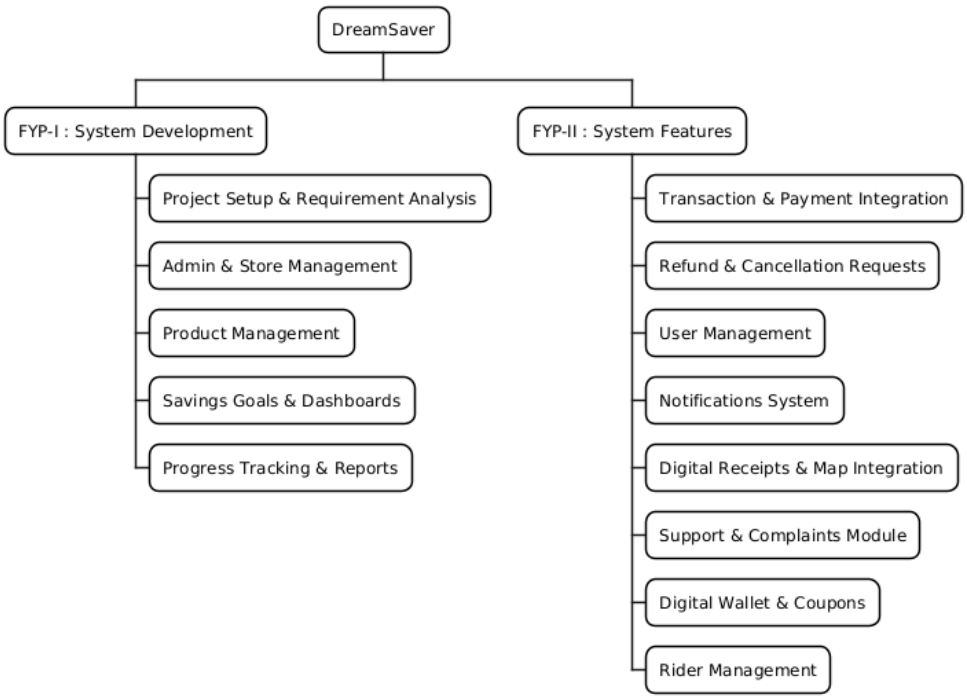


Figure 3: FYP I & FYP II – Work Breakdown

18. References

- Investopedia. “Layaway Plan – Definition and How It Works.” Last accessed September 2025. https://www.investopedia.com/terms/l/layaway-plan.asp
- Karandaaz Pakistan. (2023). “Digital Payments in Pakistan: Landscape & Opportunities.” https://karandaaz.com.pk/research/
- The Express Tribune. (2023, October 10). Daraz sees growth in Pakistani user base amid economic challenges.\https://tribune.com.pk/story/2441000/daraz-sees-growth-in-pakistani-user-base-amid-economic-challenges
- Daraz Group. (2024). About Daraz. https://www.daraz.pk/about/
- Telemart. (2024). *About Telemart*. https://telemart.pk/About
- PriceOye. (2024). *About PriceOye*. https://priceoye.pk/about

19. Business Model

Business Model Canvas	Designed for:	Designed by:	Date:	Version:
	DreamSaver – A Digital Layaway Savings Platform	Naqash Sachwani Naveed Asad Ali	May 2026	1.0

Key Partners
- Local vendors and retailers (furniture shops, bridal dress outlets, electronics stores).
- Payment service providers (Stripe)
- Email service (Nodemailer).
- Hosting providers (Vercel).
- Social Media campaigns.
- Delivery riders.

Key Activities
- Platform development (web).
- Store onboarding and product listing.
- Transaction management (digital deposits).
- User account & goal management.
- Generating invoices, reports, and reminders.
- Complaints and dispute resolution.
- Marketing and awareness campaigns to build trust.
- Dispatching orders, monitoring live tracking, and managing escrow payouts for riders.

Key Resources
- Web platform (React frontend, NextJs backend, PostgreSQL database).
- Skilled development team.
- Store product catalogs.
- Notification system (Email).
- Secure hosting and transaction infrastructure.
- Tracking infrastructure and a rider dashboard.

Value Propositions

For Users:
- A secure way to save gradually without risk of impulsive spending.
- No debt or interest compared to instalment systems.
- Transparency through digital receipts and progress dashboards.
- Ability to save for major life events (weddings, education, healthcare, travel).

For Stores:
- Attract more customers who otherwise cannot afford lump-sum purchases.
- Build long-term customer loyalty through pre-committed savings.
- Reduce risk of defaults compared to instalments.

For Riders:
- A cashless system with secure, escrow-backed earnings, removing the risks and hassles of handling Cash-on-Delivery.

For the Application:
- Provides a centralized ecosystem connecting users, stores and riders.
- Offers a scalable business model with multiple industries (retail, weddings, education, travel).
- Builds trust through secure transaction logs and transparent processes.
- Enables data-driven insights on user saving behaviour and vendor sales trends.
- Positions itself as a localized, interest-free alternative to BNPL (Buy Now, Pay Later) solutions in Pakistan.

Customer Relationships
- Personalized dashboards.
- Digital invoice for transparency.
- Customer support for refunds/disputes.
- Store relationship management through admin panel.
- Trust-building via progress reports and transaction history.
- Enhanced trust and peace of mind through real-time delivery visibility and secure package handoffs

Channels
- Web application.
- Social media for awareness and onboarding vendors.
- Store registration (partnership agreement).
- live map tracking page and a dedicated portal for riders.

Customer Segments

Primary:
Low- and middle-income individuals saving for weddings, furniture, healthcare, or education.

Secondary:
Stores/retailers seeking alternatives to instalment plans.
Families planning group contributions for savings goals.
Students saving for education fees.

Cost Structure
- Development costs (frontend, backend, database).
- Server hosting and maintenance costs.
- Payment gateway.
- Marketing, Store and Rider onboarding.
- Email notification.
- Support and maintenance team salaries.

Revenue Streams
- Store revenue share on completed goals.
- Charges on goal cancellation before completion.

Figure 4: Business Model Canvas

Software Requirements Specification

1. Introduction

1.1 Purpose

This document aims to describe the Software Requirements Specification (SRS) for DreamSaver – Layaway Saving Platform. This document describes all the function and non-functionality requirements of the platform. It will be utilized by developers, supervisors, system evaluators and testers to learn the system's working and action.

1.2 Document Conventions

- Font Style: Times New Roman
- Font Size: 12 (Body), 14 (Sub-Headings), 16 (Main Headings)
- Use Case format created using standard SRS table structure
- Requirements are written in simple and clear English

1.3 Intended Audience and Reading Suggestions

- Supervisors / Evaluation Panel: To discuss scope and feasibility.
- Developers: Understanding features and system behavior.
- Testers: To sign up for tests and either skip or test content.
- Project Members: To follow up the Project Roadmap.

1.4 Product Scope

DreamSaver is an online website that allows the consumer to strategically save toward a particular purchase (such as a wedding, furniture, electronics, or any other product) incrementally. Customers make little deposits; do not borrow or pay interest. Full Councilor(s) for products are listed on the stores and the delivery only takes place when the full savings are made. The platform overcome impulsive spending problem and encourage disciplined savings, by introducing the idea of discipline in accordance with goals in Pakistan.

1.5 References

- Investopedia – “Layaway Plan Definition”
- Daraz.pk, PriceOye.pk, Telemart.pk – Payment & Store policies
- SECP Pakistan – E-commerce Guidelines
- Karandaaz Pakistan. (2023). “Digital Payments in Pakistan: Landscape & Opportunities.” https://karandaaz.com.pk/research/

2. Overall Description

2.1 Product Perspective

DreamSaver is a new standalone Web-based platform—a solution that is not based on any existing system. To digitalize the concept of layaway, which was traditionally practiced in Pakistan which involved customers putting aside small amounts at numerous physical storefronts for later use. BFF is a saving-first, receive-later initiative to help people save instead of going into debt, whereas CostO is build on the idea of “buy now, pay now or installments”.

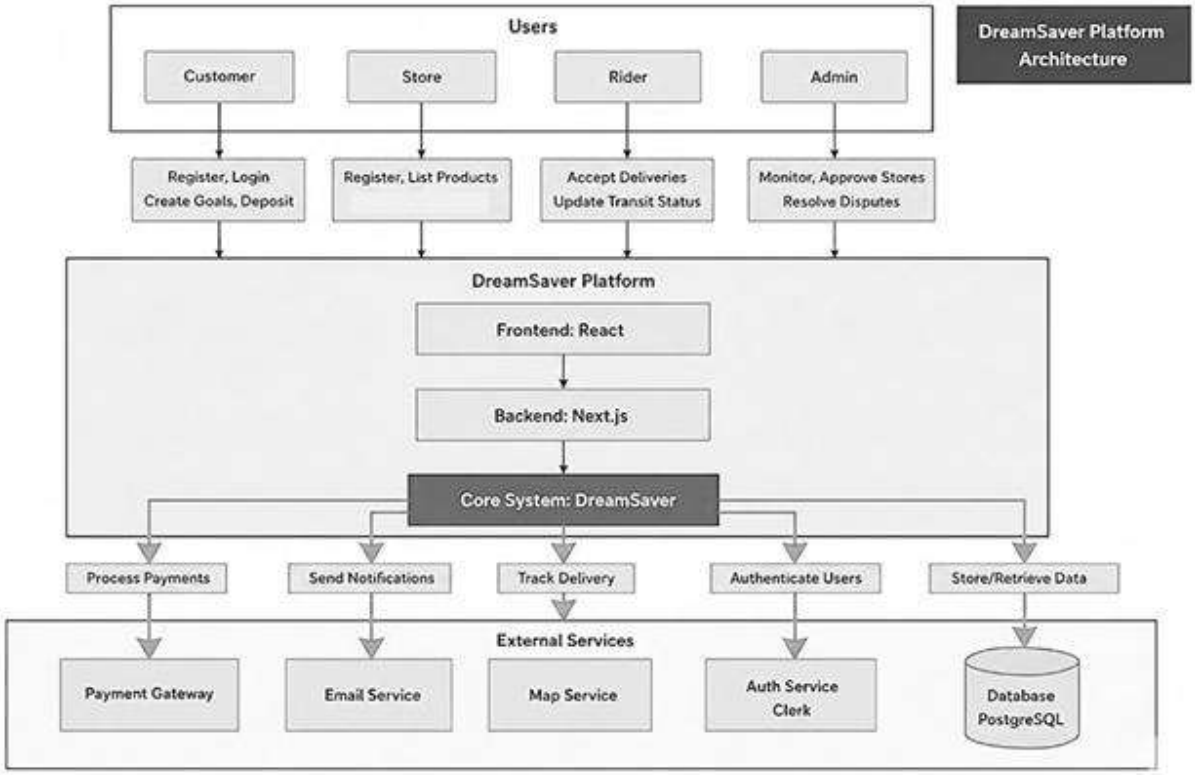


Figure 5: Product Perspective

2.2 Product Functions

The main functionalities of DreamSaver include:

- User & login the details of the customers / stores / admin.
- Goal-Based Saving Creation
- Deposits (Daily/Weekly/Monthly)
- Create a Product Listing & Price Lock
- Digital Receipts & Transaction History
- Coupons
- Support & Complaints
- Rider Management
- Digital Wallet
- Notifications
- Map Integration (Delivery Tracking)
- Refund & Goal Cancellation
- Admin Monitor I/O and restrict system access

2.3 User Classes and Characteristics

User Class	Characteristics
Customer/User	Basic Internet user, who saves money for definite objectives. Not particularly technical, will be user friendly.
Store	Small or Large business owner products are listed Trust in need, price lock, punctual payments.
Admin	Review Stores and Riders, resolves conflict, enforces policies.
Rider	Assisting in last mile delivery. Straightforward delivery status tracking, and a secure interface.

Table 2: User Classes and Characteristics

2.4 Operating Environment

- Frontend: React
- Backend: Next.js
- Database: PostgreSQL / Neon
- Internet Requirement: Stability of internet connection is required.
- Map & Notification Support: API with map & notification support to make them available.

2.5 Design and Implementation Constraints

- Needs to be following Pakistani E-commerce laws & SECP Regulations.
- Financial safety, data encryption.
- No Riba (interest) – for creating a Shariah compatible system.

2.6 User Documentation

- User Manual (How to Use Platform)

2.7 Assumptions and Dependencies

- Users need to have a basic Pc or mobile which is capable of connecting to the Internet.
- There is a price locking policy that is followed by stores – it is taken at product listing time.
- They should have a payment gateway (such as Stripe).
- All regulations of the SECP and e-commerce should be respected.

3. External Interface Requirements

3.1 User Interfaces

The application will have a simple and user friendly interface for both web and mobile. The screens that user would be seeing include Signup Screen which is capturing users' registers with basic details like name, e-mail, password; and Login Screen where the users are entering email and password. Once the user successfully logs in, he/she will get redirected to the Home

Screen where he/she will have topics to navigate such as About, Contact, Shop, My Goals etc.

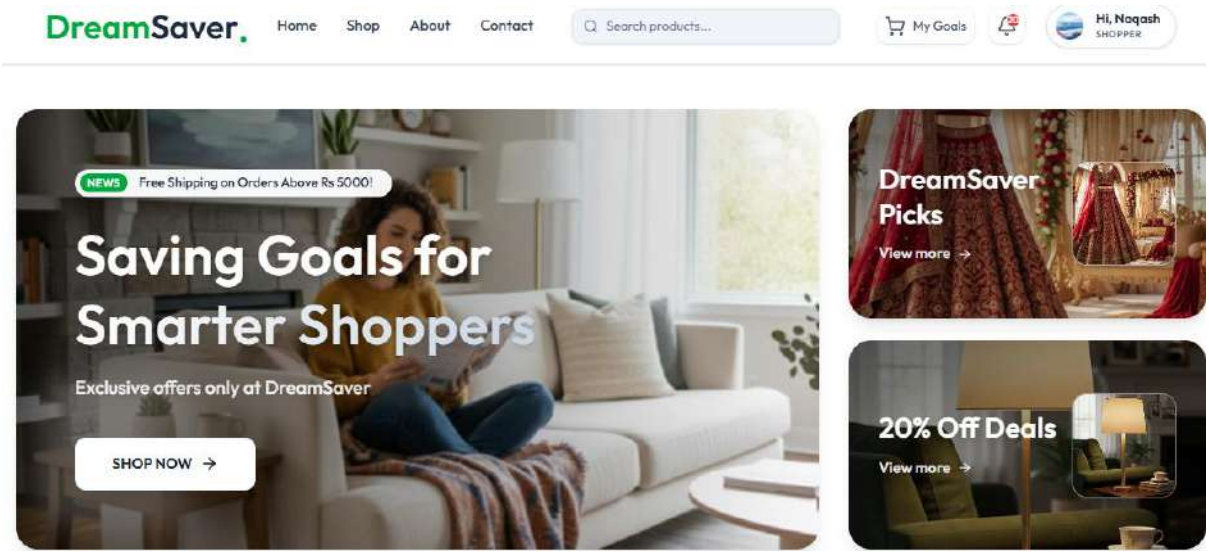


Figure 6: Home Screen

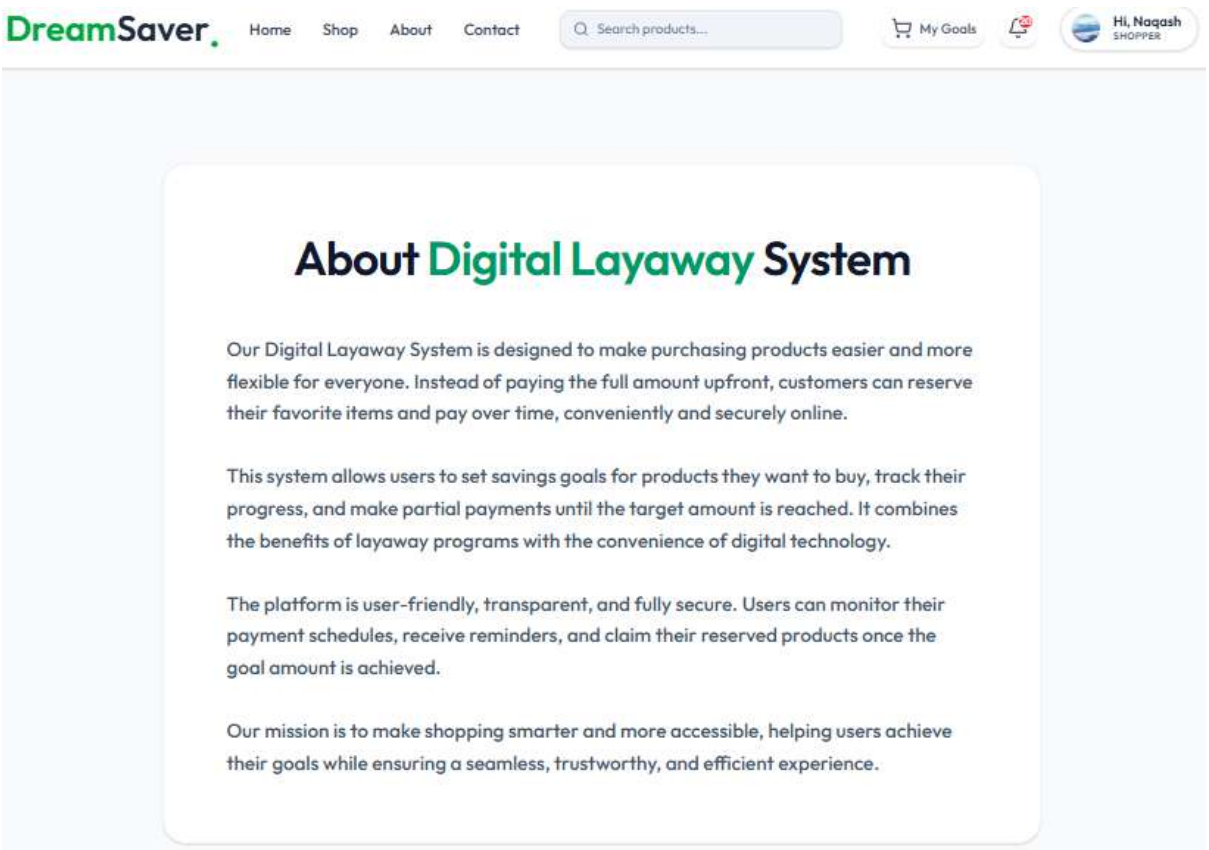


Figure 7: About Screen

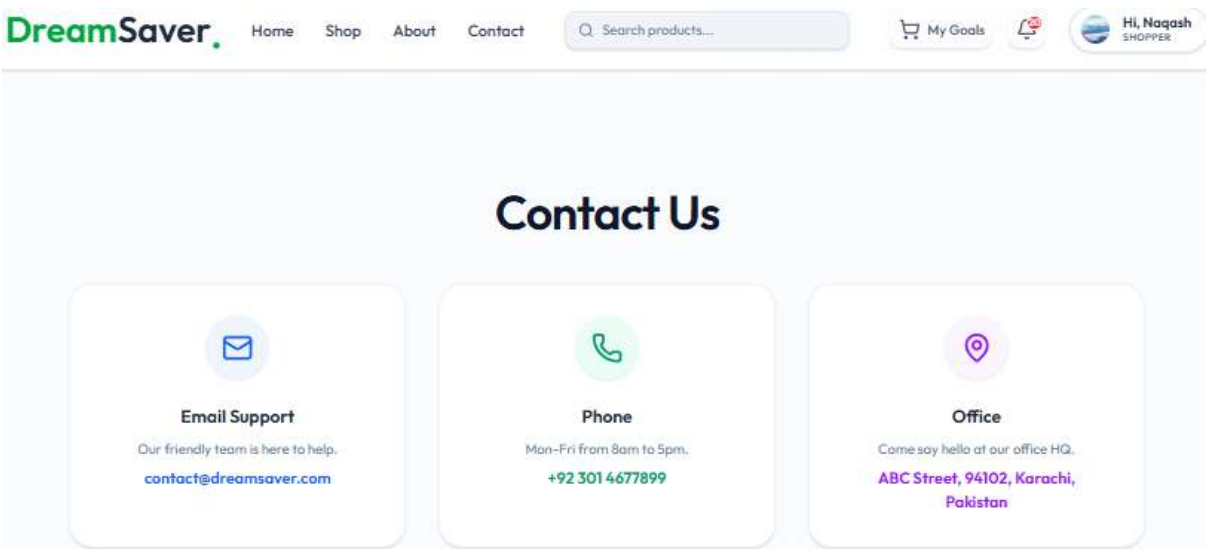


Figure 8: Contact Screen

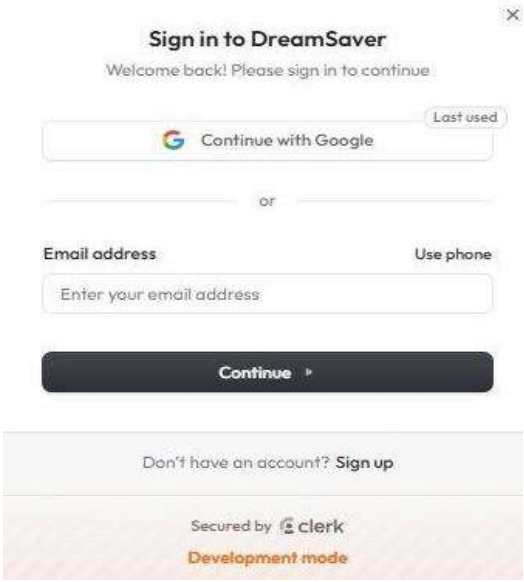


Figure 9: Sign-in Screen

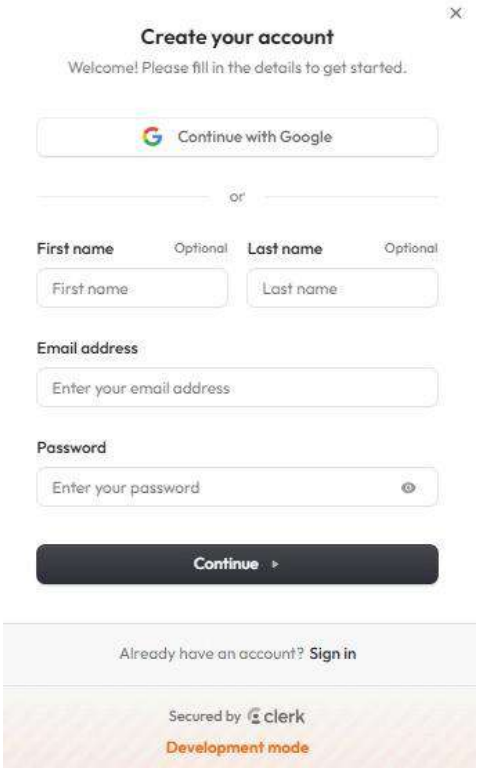


Figure 10: Sign up Screen

3.2 Hardware Interfaces

DreamSaver will work on:

Device	Interface Usage
Mobile Phone / Tablet	Touch interface for app/web access
Laptop / PC	Keyboard & mouse for admin and Store use
Server Machine	Hosting backend, database, and secure storage

Table 3: Hardware Interfaces

No special hardware is required other than standard devices with internet connectivity.

3.3 Software Interfaces

Software / Service	Purpose
Database (PostgreSQL / Neon)	Store users, Stores, goals, transactions
Payment Gateway (Stripe)	Online deposits
Email	Send notifications
Map (React Leaflet)	Delivery tracking
Authentication (Clerk)	Secure login, role-based access

Table 4: Software Interfaces

3.4 Communications Interfaces

- **Protocol:** HTTPS (Secure Web Communication)
- **Data Encryption:** SSL/TLS
- **Internet Required:** Real-time sync for deposits, goals, notifications
- **Email Communication:** Reset password, confirmations.
- **Delivery Tracking Communication:** Map tracking

4. System Features

The features offered by the DreamSaver Layaway Platform are geared toward helping customers highlight, manage and achieve product savings. Below, each feature is explained through separate use cases, and Core's functional behavior with customers, Stores, and administrators. These capabilities allow for safe registration, goal tracking, monetary deposits, and participation in stores, among other things.

4.1 User Registration (UC-01)

Use Case ID	**UC-01**
Use Case Name	User Registration
Actors	Customer
Description	This use case will allow a new customer to sign up on the DreamSaver platform with personal information like name, email and password. To assure authenticity and preventing duplicate or fraudulent accounts, the system verifies the email with verification email send to the email address he/she entered.
Preconditions	• User must have their email address. • User is not allowed to have an existing registered account.
Postconditions	• New user account is created and stored successfully in the system. • User can proceed to log in or set saving goals.
Main Flow	1. User visits the registration page. 2. User provides information (Name, e-mail address, password). 3. Verification e-mail is sent to the specified e-mail address. 4. User verify email. 5. System checks the email. 6. System stores the user data and creates an account. 7. A success message is displayed and redirected to login page.
Alternate Flow	A1. User types the wrong email address → System warns of the error and asks user to re-enter email. A2. E-mail already present → User is prompted to proceed to login or recover password.

Table 5: User Registration

4.2 User Login (UC-02)

Use Case ID	UC-02
Use Case Name	User Login
Actors	Customer
Description	This use case enables a registered user to login the platform using proper credentials. User is authenticated and allowed access to dashboards and saving goal features.
Preconditions	• Must be a registered User on the Platform. • User needs to have valid login credentials.
Postconditions	• User logs in and redirects to the dashboard. • Access through session is started for security.
Main Flow	1. User enters username and password. 2. Users logs in. 3. System validates credentials. 4. Access and opens dashboard.
Alternate Flow	A1. Wrong credentials → Error appears and asks for a re-entry. A2. Attempted to sign in multiple times→ Account locked, when account has been locked after multiple attempts, system will prompt for password recovery.

Table 6: User Login

4.3 Create Savings Goal (UC-03)

Use Case ID	UC-03
Use Case Name	Create Savings Goal
Actors	Customer
Description	This use case enables a customer to establish a savings objective for a particular product, by choosing a store product and duration. The product price is put in lock, the layaway plan is reset.
Preconditions	• Must be logged in as a user. • Registered stores must have a valid product.
Postconditions	• A new savings objective is set to current status. • The selected product is locked under the user's plan.

Main Flow	1. User clicks a product in the store listing. 2. User selects the length of time and agrees to the terms. 3. User clicks “Create Goal”. 4. Locks product and save goal. 5. Goal is displayed in the My Goals Page.
Alternate Flow	A1. System shows “Out of Stock” when the product is unavailable. A2. User inputs an invalid duration → System prints an error message and asks for a valid duration.

Table 7: Create Savings Goal

4.4 Deposit to Savings Goal (UC-04)

Use Case ID	**UC-04**
Use Case Name	Deposit to Savings Goal
Actors	Customer
Description	This use case allows the customer to save their money using digital payment methods like Stripe towards their savings target. A digital receipt is generated and the progress towards the goal is updated.
Preconditions	• Users need to have an active savings goal. • User must have a valid payment method.
Postconditions	• Deposit is recorded and Goal progress updated with notification.
Main Flow	1. User activates their active goal. 2. The user clicks “Deposit Now”. 3. User chooses a payment option. 4. System processes payment. 5. System produces digital receipt. 6. Updates progress bar for the savings goal in the system.
Alternate Flow	A1. Payment failed → System displays “Transaction Failed”. A2. Insufficient funds → System prompts either to try again.

Table 8: Deposit to Savings Goal

4.5 Track Savings Progress (UC-05)

Use Case ID	**UC-05**
Use Case Name	Track Savings Progress
Actors	Customer
Description	In this use case, the customer can see the progress of goals they have, deposit history, remaining amount and the expected date of completion.

Preconditions	• User's savings goal(s) should be active.
Postconditions	• User updates progress without changing the goal.
Main Flow	1. User clicks on the dashboard. 2. User selects a savings goal. 3. System displays total saved, total remaining, deadline, and deposit history.
Alternate Flow	A1. No active goal → System will show "No active savings goals found".

Table 9: Track Savings Progress

4.6 Request Goal Cancellation & Refund (UC-06)

Use Case ID	**UC-06**
Use Case Name	Request Goal Cancellation & Refund
Actors	Customer
Description	This use case enables the cancellation of an existing savings objective. The user requests for the return of the amount deposited. The system sends the request to the admin for processing refund based on the policy and then deducts the service or store charges.
Preconditions	• User must be logged in. • The savings goal need to be active. • Must accept refund policy by the user.
Postconditions	• Goal is marked as "Cancelled". • Forwarded to further review for refund.
Main Flow	1. User clicks on the goal details. 2. User clicks "Cancel Goal". 3. System displays cancellation notification and deductions. 4. User confirms cancellation. 5. The system requests a refund to the admin. 6. The status of the goal changes to "Pending Refund".
Alternate Flow	A1. User rejects the policy → The system aborts and terminates the cancellation. A2. Goal is already close to maturity → "Non-refundable stage" alert from the system.

Table 10: Request Goal Cancellation & Refund

4.7 Redeem Product After Goal Completion (UC-07)

Use Case ID	UC-07
Use Case Name	Redeem Product After Goal Completion
Actors	Customer, Store
Description	This use case enables the customer to ask for the product delivery after the complete payment of the saving goal. System sends a notification to the store and confirms that dispatch was made.
Preconditions	• User should have attained 100% of the goal amount. • Store must be able to have the product available.
Postconditions	• Product redemption initiated. • Final payment to store after product delivery.
Main Flow	1. Goal is complete, system detects. 2. User clicks on Redeem Now. 3. System notifies store. 4. Store assign rider. 5. Store confirms dispatch. 6. Once the successful delivery, the payment is released from the system to store. 7. Marks goal as “Completed”.
Alternate Flow	A1. Stock Issue reports → System displays new delivery date.. A2. Delay by customer in redemption → Product is reserved in system.

Table 11: Redeem Product After Goal Completion

4.8 Receive Notifications (UC-08)

Use Case ID	UC-08
Use Case Name	Receive Notifications
Actors	Customer, System
Description	This use case defines how the system automatically sends notification for after deposits, cancellation, delivery, and completion via email, or in-app notifications.
Preconditions	• Customer is required to register. • There is at least one active goal.
Postconditions	• Timely email sent to Customer. • Customer engagement/discipline is enhanced.

Main Flow	1. System checks deposit. 2. Notification is sent (email/ in-app). 3. Customer views and acknowledges notification.
Alternate Flow	A1. Notification service offline → System will try again to send later. A2. Customer unsubscribe → System disables non-essential notifications.

Table 12: Receive Notifications

4.9 Store Registration (UC-09)

Use Case ID	**UC-09**
Use Case Name	Store Registration
Actors	Store, Admin
Description	Allows a store to register by providing details like Name, Email, Description, Contact, Address, Banking details. The documents are administered, reviewed and verified before being approved.
Preconditions	• There is a requirement for store to have necessary information. • Agreeing to Platform Policies by Store.
Postconditions	• Store an account that was created with the "Pending Approval" flag. • Awaiting admin verification.
Main Flow	1. Opens registration page. 2. Store submits details. 3. Stores data in system and shows as "Pending". 4. Admin receives verification request. 5. Admin is notified of verification request.. 6. Admin approves/rejects Store.
Alternate Flow	A1. Missing details → System prompts re-upload. A2. Admin rejects → System notifies Store with reason.

Table 13: Store Registration

4.10 Product Listing by Store (UC-10)

Use Case ID	**UC-10**
Use Case Name	Product Listing
Actors	Store
Description	Allows stores to enter product name, price, category, and stock information to list products that can be purchased by layaway.
Preconditions	• Store must be approved and logged in. • Must be able to access dashboard.

Postconditions	• Products can be seen by customers for goal creation.
Main Flow	1. Store opens dashboard. 2. Click on “Add Product”. 3. Stores add details of product and images. 4. System validates details. 5. Product is placed on the marketplace by system.
Alternate Flow	A1. Missing mandatory fields → System will request that they be filled correctly. A2. Price conflicts with lock policy → System alerts Store.

Table 14: Product Listing by Store

4.11 Lock Product for Layaway (UC-11)

Use Case ID	**UC-11**
Use Case Name	Lock Product for Layaway
Actors	Store, User
Description	The objective of this use case is to lock the product for a particular customer when a savings objective is created. During the lock period, the system reserves the product and other customers are not able to buy the product.
Preconditions	• Must have a savings goal set up by customer. • Store must have product listed and in Stock.
Postconditions	• Product will be retained by the customer until layaway contract is fulfilled. • Price is fixed at the time of completion or when cancelled.
Main Flow	1. Customer sets them a goal when it comes to purchasing Store product. 2. System indicates that product is “Locked”. 3. A period and price freeze on the systems.
Alternate Flow	A1. Store declines lock due to no stock → System informs customer. A2. Price Lock Policy validation by System – Product Price changed before Goal.

Table 15: Lock Product for Layaway

4.12 Confirm Delivery & Final Payment Release (UC-12)

Use Case ID	**UC-12**
Use Case Name	Confirm Delivery & Final Payment Release
Actors	Store, Customer, Admin, Rider

Description	This is an example use case for post goal delivery of final product. Rider/User confirms delivery and then admin releases payment to Store.
Preconditions	• Customer is required to save the entire amount. • Store must ensure that product is available for deliver.
Postconditions	• Product has been delivered to the customer. • Full payment is received to the Store.
Main Flow	1. Customer clicks Redeem Product. 2. System notifies Store. 3. Rider/User/Store confirms delivery. 4. System releases payment to Store. 5. System marks order as "Completed".
Alternate Flow	A1. Customer reports non-delivery → System keeps payment and notifies admin. A2. Damaged product reported → Refund/Replacement process is started.

Table 16: Confirm Delivery & Final Payment Release

4.13 View Store Dashboard & Earnings (UC-13)

Use Case ID	**UC-13**
Use Case Name	View Store Dashboard & Earnings
Actors	Store
Description	In this use case, Stores can see real time data such as completed orders, and total earnings.
Preconditions	• Must be logged in and approved as a Store. • A minimum of one product is listed.
Postconditions	• Store can track layaway sales and make stock plan.
Main Flow	1. Store logs to the dashboard. 2. Click on "Dashboard Overview". 3. System displays completed orders, products and earnings. 4. Keep in track the views of detailed analytics.
Alternate Flow	A1. No active goals → Show "No Active Layaway Plans" in the system. A2. Store dashboard error → System displays maintenance message.

Table 17: View Store Dashboard & Earnings

4.14 Approve or Reject Store Applications (UC-14)

Use Case ID	**UC-14**
Use Case Name	Approve/Reject Store Applications
Actors	Admin

Description	Through this use case, an admin can check whether a store is registered or not and either approve or disapprove the marketplace access according to the authenticity of the store.
Preconditions	• Must fill out necessary information for store. • The user must have an administrator login with permissions.
Postconditions	• Record result of status. • Stores are informed.
Main Flow	1. Admin opens pending Store list. 2. Admin reviews set up information. 3. Admin approves or rejects application. 4. System notifies Store.
Alternate Flow	A1. Not enough documents → Admin asks for a repeat submission. A2. Fraud detected → Admin blocks the Store account.

Table 18: Approve or Reject Store Applications

4.15 Monitor Transactions & Resolve Disputes (UC-15)

Use Case ID	**UC-15**
Use Case Name	Monitor Transactions & Resolve Disputes
Actors	Admin
Description	This use case enables an Admin to track deposits, refunds, disputes between Stores & customers, complaints regarding delivery and act on it.
Preconditions	• Admin must be logged in. • There must be transactions in system.
Postconditions	• Disputes settled and actions recorded. • Payments held/released accordingly.
Main Flow	1. Admin opens up dispute panel. 2. Flagged issues shown in the system. 3. Admin checks the details of the dispute. 4. Admin has taken corrective measures. 5. System records final decisions.
Alternate Flow	A1. Incomplete evidence → Admin requests more info. A2. Fraud detected → User/Store will be permanently banned by Admin.

Table 19: Monitor Transactions & Resolve Dispute

4.16 Manage Price Lock & Inflation Policy (UC-16)

Use Case ID	**UC-16**
Use Case Name	Manage Price Lock & Inflation Adjustments

Actors	Admin, Store
Description	This use case gives the admin the ability to enforce and tweak price lock policies to safeguard customers and Stores from inflation.
Preconditions	• It is necessary to have active savings goals. • Product Lock for Customer Must be in Store.
Postconditions	• Price lock is still in effect or is adjusted on mutual agreement.
Main Flow	1. Admin approves/rejects price fluctuation requests from Stores. 2. Admin review's goal timelines. 3. New price lock terms are applied by admin. 4. System notifies customers and Stores.
Alternate Flow	A1. Customer disagrees → Admin initiates refund or alternative product.

Table 20: Manage Price Lock & Inflation Policy

4.17 Generate System Reports & Audit Logs (UC-17)

Use Case ID	**UC-17**
Use Case Name	Generate System Reports & Audit Logs
Actors	Admin
Description	This use case will allow admin to create financial reports and escrow reports for transparency, audit and adherence to SECP guidelines.
Preconditions	• Admin should have reporting privileges. • There also needs to be a history of transactions.
Postconditions	• Outline of the entire system can be generated as pdf reports for record keeping.
Main Flow	1. Admin view dashboard. 2. Admin downloads report. 3. System compiles data. 4. System exports report.
Alternate Flow	A1. No data available → Report shows "No Data".

Table 21: Generate System Reports & Audit Logs

4.18 Automated Notifications (UC-18)

Use Case ID	**UC-18**
Use Case Name	Automated Notifications
Actors	System

Description	This use case is for the system to automatically send an e-Mail or in-app notification for deposit, goals etc.
Preconditions	• Must have an active account or goal. • Background notification needs to be enabled.
Postconditions	• Customers are notified timely. • Logs are kept of notifications.
Main Flow	1. When a notification trigger occurs (increase in deposits, goal update.). 2. System prepares notification message. 3. Notification is sent (Email/Application). 4. Customer views notification. 5. System logs notification in history.
Alternate Flow	A1. Notification service down → System will retry later. A2. Customer unsubscribed → Pauses non-critical alerts.

Table 22: Automated Notifications

4.19 Track Delivery via Map (UC-19)

Use Case ID	**UC-19**
Use Case Name	Track Delivery via Map
Actors	Customer, Store, System
Description	This use case allows a customer to be notified of where their product delivery is after the goal is completed. The Store labels the product as "In transit" and the system includes a live map view with route tracking with the customer being able to see the status of their delivery.
Preconditions	• Customer must have fulfilled the objective of the savings. • The dispatch of the product shall be confirmed by store. • Delivery service should be able to track live maps.
Postconditions	• Customer is notified of delivery status. • Marks product as delivered once product delivery is completed.
Main Flow	1. Product is marked as ‘in-transit'. 2. The customer clicks ‘Track Delivery' option. 3. System shows current position on map of the Store route. 4. Customer keeps an eye on the delivery until completion.
Alternate Flow	A1. Application cannot display a map → System will indicate the last time the map was updated.

Table 23: Track Delivery via Map

4.20 Apply Coupon (UC-20)

Use Case ID	UC-20
Use Case Name	Apply Coupon
Actors	Customer
Description	The use case allows the customer to input a promotional coupon code in the creation of a goal, to get a discount on the amount of the goal. The system checks code validity, usage restrictions, and user eligibility before applying the discount.
Preconditions	• User should have their account logged in. • User must have a valid coupon code with them.
Postconditions	• The total amount is correctly calculated using the coupon discount. • The system keeps track of the user's coupon usage.
Main Flow	1. User is redirected to the goal creation page. 2. The user types in the coupon code in the input box. 3. User clicks on the "Apply" button. 4. System checks the validity of the coupon code based on expiry dates, usage limits and coupon status. 5. System calculates the discounted target amount. 6. System updates the screen with discount applied and the new final price. 7. System displays a success message.
Alternate Flow	A1. Coupon code is incorrect or expired → System returns an error message ("Invalid or expired coupon code") and requests for the coupon code to be re-entered. A2. User's maximum number of uses for the coupon is reached → An error message is displayed (Coupon limit reached). A3. Coupon is strictly for new users, but the user has previous goals → System displays an error ("This coupon is strictly for new users").

Table 24: Apply Coupon

4.21 Manage Digital Wallet - Withdraw Funds (UC-21)

Use Case ID	UC-21
Use Case Name	Withdraw Funds from Digital Wallet
Actors	Customer
Description	In this use case, a customer can withdraw money from his/her digital wallet, DreamSaver, to his/her bank account.
Preconditions	• User should be signed in to their account. • User should have a positive account balance in his/her digital wallet, which should be at least equal to the amount for which he/she wants to withdraw.
Postconditions	• The requested amount is deducted from the user's digital wallet. • A withdrawal is recorded.

Main Flow	1. User access the "My Wallet" page. 2. User selects the "Withdraw" option. 3. The user types the amount to be withdrawn, account name and the account number. 4. User clicks the "Withdraw". 5. System checks the wallet balance to see if it has enough funds to complete the request. 6. System takes the amount from the wallet's balance. 7. System generates a user associated WITHDRAWAL transaction record. 8. System alerts user with an e-mail and in-app message for the withdrawal. 9. System displays a success message on the screen.
Alternate Flow	A1. User requests more than they have in their wallet → System shows an error message ("Insufficient wallet balance") and does not allow for submission. A2. User leaves blank fields for bank/payout information, system alerts user to complete.

Table 25: Manage Digital Wallet - Withdraw Funds

4.22 File a Complaint (UC-22)

Use Case ID	**UC-22**
Use Case Name	File a Complaint
Actors	Customer, Store
Description	The use case allows users to make complaints about products, delivery, or the behavior of the store. The system is able to correlate the complaint with the appropriate goal/order, make sure that the time limits are followed, and queue the ticket for Admin resolution.
Preconditions	• User must be logged into their account.. • If the complaint is about a specific order, the user must have an existing goal/order on the platform.
Postconditions	• A new complaint ticket is successfully created with a status. • The appropriate Store and Admin are informed of the problem.
Main Flow	1. The user logs in to the Support & Complaints dashboard. 2. User clicks "File a New Complaint". 3. User types a title, a description and chooses the category of the complaint (such as Delivery, Product Issue). 4. User chooses the relevant Goal/Order from the drop down menu to connect the complaint. 5. User clicks the ‘Submit Complaint' button. 6. System checks the eligibility of delivered items for the seven-day rule. 7. System creates a new complaint record in the database. 8. System sends an automatic acceptance email to user. 9. System shows a success message and takes the user to the list of their opened tickets.

Alternate Flow	A1. Users who try to submit a 'Product Issue' complaint for an item delivered more than 7 days ago → System blocks the submission and shows an error message 'Complaints regarding delivered products must be made within 7 days of delivery'. A2. User does not specify a Title and/or Description → System tells the user to fill in the required text boxes before continuing.

Table 26: File a Complaint

4.23 Rider Registration and Verification (UC-23)

Use Case ID	**UC-23**
Use Case Name	Rider Registration and Verification
Actors	Rider, Admin
Description	This use case allows a rider to register on the platform providing valid information and upload of required identification documents. The application is retained in the system until it is reviewed by an Admin to ensure all deliveries are secure.
Preconditions	• User must have registered with a basic account in the DreamSaver platform. • User must be able to present valid identification as image files.
Postconditions	• Rider profile is successfully created into a Database. • Rider account is activated and marked as 'Approved' by the Admin.
Main Flow	1. Rider clicks on the 'Register as Rider' portal. 2. The rider fills in personal information, vehicle description and license plate. 3. Rider uploads images. 4. System checks the entered data and stores the profile as 'Pending Approval'. 5. Admin reviews the submitted documents, and sets the status to 'Approved'.
Alternate Flow	A1. Invalid document format uploaded → System shows an error and asks the rider's to upload supported image files (JPG/PNG). A2. Admin rejects application → System status becomes 'Rejected' and the rider is not able to accept deliveries.

Table 27: Rider Registration and Verification

4.24 Accept Delivery Request (UC-24)

Use Case ID	**UC-24**
Use Case Name	Accept Delivery Request
Actors	Rider, Store
Description	Description A use case for a delivery being moved from 'Pending' to 'Accepted'. If there is a Rider available, the Store acknowledges the delivery request when it is accepted by that Rider. The system then automatically updates the status to 'Accepted', and the customer immediately sees the change on his/her live tracking page from 'Pending' to ' Accepted'.

Preconditions	• Customer has reached their savings target and has asked for a delivery. • A delivery record has been created in the system that is in the 'Pending' state. • Rider is logged in, verified and can see available delivery requests.
Postconditions	• Rider is officially assigned delivery order. • In the database, the delivery status changes to ‘Accepted'. • Upcoming changes to Customer tracking dashboard – 'Accepted' status and Riders details.
Main Flow	1. System sends Pending delivery requests to available Riders. 2. Rider checks the details of delivery and clicks on 'Accept Request'. 3. The accepted request will be displayed on their dashboard, and the Rider will be confirmed. 4. The delivery status changes from 'Pending' to ' Accepted'. 5. System pushes update to the Customer's tracking page, showing the new status and the assigned Rider's information.
Alternate Flow	A1. Vehicle type mismatch: Store rejects the Rider assignment → delivery status reverts to 'Pending' and it is sent as a broadcast message to other available Riders. A2. The other Rider accepts the request milliseconds beforehand → System shows the other Rider "Request no longer available".

Table 28: Accept Delivery Request

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- All pages, such as Dashboard, Goal Creation, Store Panel and Notifications, should require a short time to load when the internet connection is normal.
- Deposit/ Payment confirmations shall be processed within 20 – 30 seconds via API (Stripe).
- Track the real-time delivery location with map shall refresh the location updates.
- Reports, deposits and goal progress shall take a shorter time to run in the database.

5.2 Safety Requirements

- All financial transactions and goal deposits shall be backed up automatically every day and shall be protected against loss of data.
- System should gracefully handle errors and show a safe message, such as “Service is temporarily unavailable”, rather than technical errors.
- If there is a change in the policy, system maintenance or risk of delivery, Users shall be notified.
- Delivery tracking shall not provide driver personal information for the purpose of physical security.

5.3 Security Requirements

- Users (Customers & Stores) should access using secure log-in with encrypted log-in credentials (passwords).
- Two factor authentication shall be used for registering.
- Role Based Access Control (RBAC)
- Customer is only able to view/manage their goals.
- Only listed products and orders can be managed by the stores.
- Admin is able to track every transaction, dispute and Store verification.
- The system shall be in line with SECP & Digital Payment Security Standards.
- If there is “fraudulent activity”, this shall be automatically blocked and reported to admin.
- Rider identity documents be securely uploaded, encrypted and then manually verified by an Admin prior to account activation.

5.4 Software Quality Attributes

- Usability: Interface should be mobile friendly, easy and should lead the user through financial tasks (goal creation, deposits, refunds).
- Reliability: System availability – 99% (excluding scheduled maintenance). System should be scalable to accommodate future additions such as referring partner banks, and referral programs.
- Maintainability: Easy to add new features, such as payment gateways or map tracking.
- Portability: Application should be available for web browsers and mobile devices (be responsive).

5.5 Business Rules

- If a product is set in a layaway goal, the product price cannot be increased or decreased without the consent of both Store and admin.
- Admin can approve/reject a Store registration according to the information given.
- Reserved products must be kept in stock and are not available for sale to other stores.
- Only delve into the payment once 100% of the goal has been paid.
- Include Dispute cases (damaged product, non-delivery): If admin does not review and resolve within 7 working days, the dispute will be closed as "Unresolved".
- Refund deductions (service charges, Store penalties) shall be in accordance with the terms agreed when the goal was created in accordance with the DreamSaver policy.
- Rider delivery payouts will be automatically created when a delivery is made with success, but will be held in escrow until the final payout release is approved by the Admin.

Other Requirements

At this stage there are no further requirements. The previous sections already meet all the functional and non-functional requirements. Any future needs or updates will be added in further revisions of this document.

Software Design Specification

1. Introduction:

1.1 Purpose of this Document

The goal of the Software Design Specification (SDS) is to represent the design and development of the DreamSaver platform. The SRS documents the "what" of the system, while the SDS documents the "how" of the system; it documents the architectural, structural and component-level design decisions that are made to realize the system requirements defined in the SRS.

This document outlines the system architecture, data design, user interface design, description of the components, design trade-offs, considerations for reusing components, and the pseudocode for each component. It provides clarity of understanding for the development team during implementation.

1.2 Scope of the Development Project

DreamSaver is an online digital layaway service that allows consumers to accumulate their purchases over a period of time. Products are listed in Stores, price is set for the period of the saving, and products are delivered after the user has achieved the goal.

The system supports:

- User registration & login
- Goal creation & deposits
- Store registration & product listing
- The price, track, dashboard and delivery function are all built in
- Refund & cancellation workflows
- Notifications and digital receipts
- Store management, dispute, and system reports administration
- Coupons, Wallet, Support & Complaint
- Rider Management

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DB	Database
ERD	Entity Relationship Diagram
HTTP/HTTPS	Hypertext Transfer Protocol / Secure HTTP
UI/UX	User Interface / User Experience
RBAC	Role-Based Access Control
SDS	Software Design Specification
SRS	Software Requirements Specification

Table 29: Definitions, Acronyms, and Abbreviations

1.4 References

- Investopedia. “Layaway Plan – Definition and How It Works.” https://www.investopedia.com/terms/l/layaway-plan.asp
- Stripe. “Stripe API Reference – Payments, Refunds, and Webhooks.” https://stripe.com/docs
- Clerk. “Clerk Authentication Documentation.” https://clerk.com/docs
- React.js. “React Official Documentation.” https://react.dev
- Next.js. “Next.js App Router Documentation.” https://nextjs.org/docs
- PostgreSQL. “PostgreSQL Database Official Documentation.” https://www.postgresql.org/docs
- Prisma. “Prisma ORM Documentation.” https://www.prisma.io/docs

1.5 Overview of Document

This Software Design Specification (SDS) document gives a detailed and organized account of the system's structure, parts, interfaces and actions. Describes the overall design approach, key modules and the relationship between user, store, rider and administrative functions. The document starts with an introduction to the system goals, followed by architectural decisions, component-level specification, and designs of the user interface. Every section describes the way the features are realised, why they were designed and the limitations which were taken into account when designing the system. The SDS also contains pseudocode to help implement and make the code easier for developers to understand. This document provides a comprehensive and visible design specification, which is used as a reference for development, testing, maintenance and future enhancements.

2. System Architecture Description

2.1 Section Overview

This section covers architectural design aspects of DreamSaver: the selected architecture pattern (Next.js App with API Routes); data flow within the application; constraints for the application; internal/external interfaces; and rationale.

2.2 General Constraints

Hardware & Software Constraints

- Runs on cloud platform (Vercel / Neon for PostgreSQL).
- Should be compatible with current browsers (Chrome, Edge).
- Mobile responsive design.

Interface Constraints

- REST-based API communication.
- Clerk integration is also included with a built-in payment integration for Stripe, Inngest, and ImageKit.

- Delivery tracking.

Security Constraints

- Must follow SECP e-commerce guidelines.
- Any user and transaction data should be encrypted.
- RBAC is applied as a whole across all API endpoints.

Performance Constraints

- Payment processing <30 seconds.
- Dashboard and goals load <5 seconds under normal conditions.

2.3 Data Design

All of the application data is going to be stored on a neon database where different tables are created according to need so data can be found in a sorted manner, the application will fetch data from online neon database.

Internal Data Structures

- Objects representing user sessions
- The JSON payloads for API routes
- React state objects for UI components

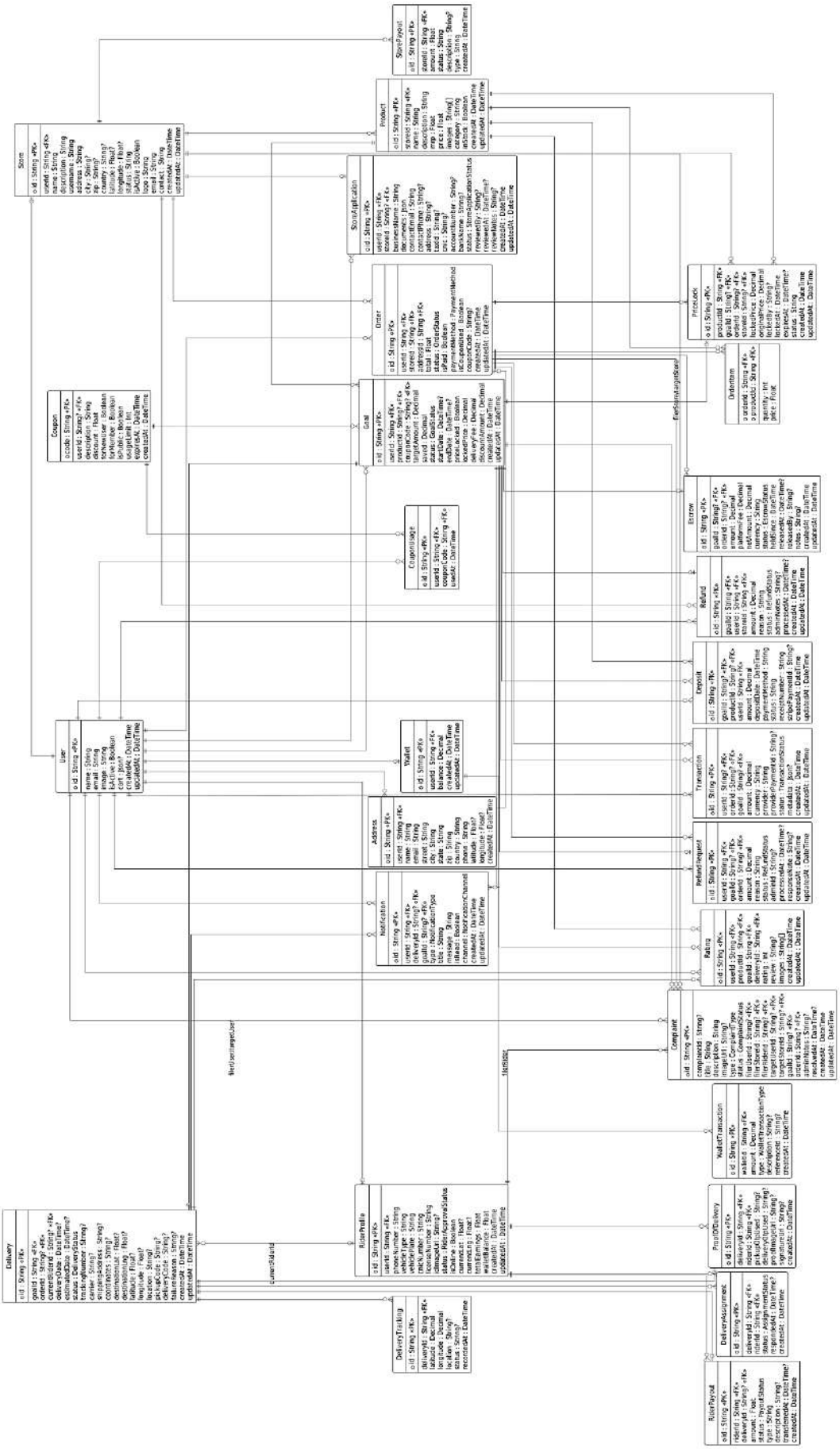


Figure 11: ERD

2.4 Program Structure

DreamSaver follows a **3-layer architecture**:

1. Presentation Layer

- Pages: Home, Login, Dashboard, My Goals, Store Panel, Admin Panel, Rider Panel
- Components: Forms, Cards, Charts, Maps

2. Application/Logic Layer

Handles:

- Goal management
- Store system
- Payment processing
- Refund workflows
- Notifications
- Coupon
- Refund
- Rider Management

3. Data Layer (PostgreSQL)

System Architectural Diagram

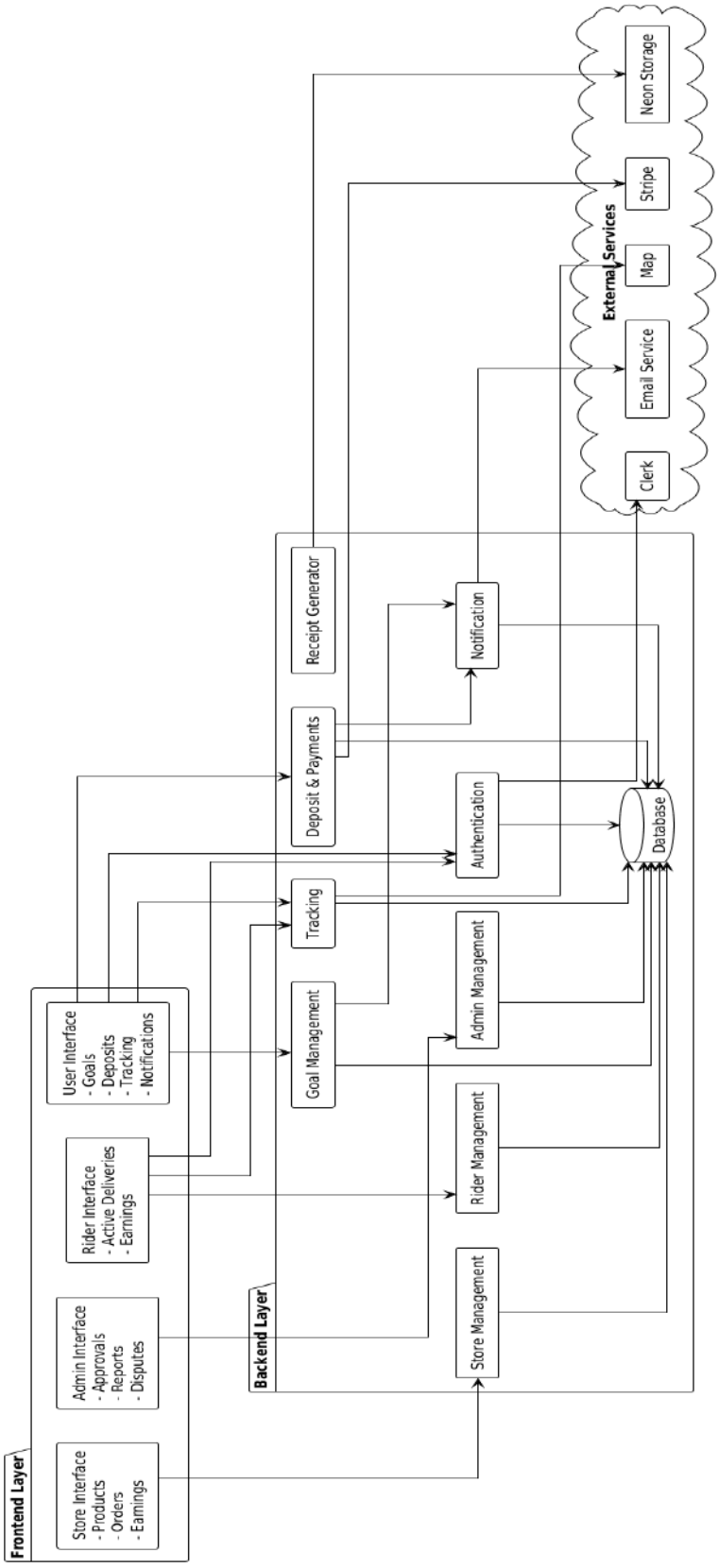


Figure 12: System Architectural Diagram

2.5 Alternatives Considered

Architecture	Pros	Cons	Decision
Microservices	Scalability	Too complex for FYP	Rejected
Serverless-only	Fast, cheap	Hard for continuous DB sessions	Rejected
Modular Monolithic (Next.js)	Balanced, easy, scalable	Requires disciplined module separation	Selected

Table 30: Alternatives Considered

3. Detailed Description of Components

3.1 Section Overview

The detailed design of each major component is given in this section, following the component template: Identification, Type, Purpose, Functions, Subordinates, Dependencies, Interfaces, Resources, Processing, Internal Data.

3.2 Component Descriptions

3.2.1 User Registration

Identification	User Sign Up
Type	**Module:** The user module is responsible for handling user account creation.
Purpose	Initiates a new customer, checks email, creates account.
Function	**Handles:** • Email validation • Password validation • Role = "user" assignment • Account creation **Processes:** • Input validation • Duplicate email checking **Outputs:** • Success message • Error messages
Subordinates	**Consists of subcomponents:** • Email Validator • Password Validator • Clerk Registration Handler
Dependencies	**Relies on:** • Clerk Authentication • Users Table **Interaction:** • DB read/write for user creation

Interfaces	**External Interface:** Clerk as verification emails. **Internal Interface:** Database (Neon) to save and retrieve user information. **User Interface:** • Name, email and password fields. • Selects which role to play (default: user). • Error messages for invalid email, weak password or existing account. • Confirmations via email (when registration is confirmed).
Resources	**Requires:** • Number of seconds it takes to execute the system for validation and encryption. • Email delivery & database queries via network. • Developer Data Storage Database / Admin Approval Status.
Processing	**Following these steps:** 1. User submits form 2. Validate fields 3. Check duplicate 4. Create Clerk user 5. Insert into DB
Data	**Handles:** The name is gathered during registration, email for account creation and verification, and the password is securely stored for account authentication.

Table 31: User Registration - Component

3.2.2 User Login

Identification	User Login
Type	**Module:** The module is responsible for handling user authentication and managing user sessions.
Purpose	Validates user identity and creates sessions.
Function	**Handles:** • Credential validation • Session creation **Processes:** • Token generation/validation via Clerk • RBAC check **Outputs:** • Session token/cookie • Error/success responses
Subordinates	**Consists of subcomponents:** • Session Manager • Token Verifier

Dependencies	**Relies on:** • Clerk Authentication • Sessions storage **Interaction:** • Auth middleware checks on protected routes
Interfaces	**External Interface:** Clerk authentication endpoints for sign-in. **Internal Interface:** Session management, token validation middleware. **User Interface:** • Each field for entering an email address and password. • "Forgot password" link. • Incorrect password and/or unverified account error messages. • Redirect to user dashboard if successful.
Resources	**Requires:** • Clerk API keys • Secure session DB
Processing	**Following these steps:** 1. Receive credentials 2. Call Clerk for authentication. 3. Make a session record / cookie 4. Redirect to dashboard
Data	**Handles:** The login process includes checking for the existence of user, obtaining a token (either verifying or generating), fetching userId and fetching roles for access control.

Table 32: User Login - Component

3.2.3 Create Savings Goal

Identification	Create Savings Goal
Type	**Module:** Creating goals and locking the product.
Purpose	Set up a savings goal for a purchase from a store and fix the price of the product.
Function	**Handles:** • Goal input (duration, terms acceptance) • Goal lock **Processes:** • Product availability check • Deposit schedule calculation • Goal creation **Outputs:** • New goal record • Show confirmation or failure message to the user.
Subordinates	**Consists of subcomponents:** • Time Period

	• Terms Acceptance
Dependencies	**Relies on:** • Products table, Store module, DB transactions **Interaction:** • Update data and create goal automically
Interfaces	**External Interface:** None (Internal Only). **Internal Interface:** Product service, Goal service, Database tables (Goal, Product). **User Interface:** • Product selector. • Duration Selection. • Terms Acceptance. • Messages for out-of-stock.
Resources	**Requires:** • Supported by a DB transaction (Prisma) • Scheduler for reminders
Processing	**Following these steps:** 1. Validate input 2. Verify product availability 3. Lock product 4. Prepare the plan and implementation of goals 5. Inform user that it was successful
Data	**Handles:** It consists of these fields which represent a goal: goalId is the goal, productId is a reference to a product to which the goal refers, target_amount is the savings target, deposits are payments made against the goal, startDate is the starting date of the goal, and endDate is the end date of the goal.

Table 33: Create Saving Goals - Component

3.2.4 Deposit to Savings Goal

Identification	Deposit to Savings Goal
Type	**Module:** User authentication and session management
Purpose	Make deposits to goals, document transactions and generate receipts.
Function	**Handles:** • Payment intent creation • Receipt generation **Processes:** • Make Stripe payment intent, confirm payment using Webhook, update DB. • To create a payment intent for the Wallet, confirm it through the webhook and update the DB. **Outputs:** • Attachment of records to the original deposit, updated goal progress, receipt.

Subordinates	**Consists of subcomponents:** • Receipt Generator • Deposit Recorder
Dependencies	**Relies on:** • Stripe API, Deposit table, Goal module, Wallet **Interaction:** • Reworked Goal Progress and Webhook updates.
Interfaces	**External Interface:** Stripe API and Wallet for payments. **Internal Interface:** Payment system, Receipt generator, Database (Deposit, Goal, Wallet). **User Interface:** • Deposit amount. • Payment progress indicator. • Errors for cards that were declined and/or payments failed. • Activity success confirmation with receipt and new progress.
Resources	**Requires:** • Stripe API keys, Wallet • DB write capacity
Processing	**Following these steps:** • Add payment as Stripe or wallet • User completes payment • Webhook confirms payment • Add deposit record and record progress towards goal
Data	**Handles:** For transaction history, each deposit has a unique id, the amount of the deposit (depositAmount), the productId and goalId of the deposit, and the date of the deposit (depositDate).

Table 34: Deposit to Savings Goals - Component

3.2.5 Track Savings Progress

Identification	Track Savings Progress
Type	**Module:** The module is responsible for aggregating and displaying the progress towards the goal.
Purpose	Calculate and show total saved, remaining deposit, and deposit history.
Function	**Handles:** • Aggregation of deposits • Progress percentage calculation **Processes:** • Fetch deposit history **Outputs:** • Create progress UI data and history list.

Subordinates	**Consists of subcomponents:** • Progress Calculator • History Aggregator
Dependencies	**Relies on:** • Deposit table, Goal table **Interaction:** • Add deposits and goals to DB
Interfaces	**External Interface:** None. **Internal Interface:** Database queries, Goal service. **User Interface:** • Progress bar or progress circle. • Transaction history including timing and quantity.
Resources	**Requires:** • DB read throughput • Charting library (Recharts)
Processing	**Following these steps:** • Query goal and deposits • Sum totals and compute percent • Create UI that displays charts and lists
Data	**Handles:** The deposit list enables progress tracking to be performed, which includes the following: Total saved deposits amount, percentage of deposits completed, and the next due deposit date.

Table 35: Track Savings Progress - Component

3.2.6 Request Goal Cancellation & Refund

Identification	Request Goal Cancellation & Refund
Type	Module: Handles refund and admin notification
Purpose	Allow users to request cancellation and initiate refund processing.
Function	**Handles:** • Eligibility checks • Refund request creation **Processes:** • If the goal is met, then the policy will be enforced as well as a mark goal pending refund and admin notification. **Outputs:** • Maintain a record and status of requests for refunds.
Subordinates	**Consists of subcomponents:** • Refund Validator • Admin Notifier
Dependencies	**Relies on:** • Refund policy configs, Deposit table, Admin module

	Interaction: • Inform admin and prevent any more deposits.
Interfaces	**External Interface:** None. **Internal Interface:** Refund service, Admin service, Database (refunds, goals), Wallet. **User Interface:** • Refund reason input. • Warning when goal is cancelled with some amount of money. • On submission, confirm modal. • Pending, approved and rejected status updates.
Resources	**Requires:** • Admin review.
Processing	**Following these steps:** • Validate refund eligibility • Create refund record, set goal to pending_refund • Notify admin for manual review
Data	**Handles:** Refunds store the refundId, the amount_requested, the user's reason, and the refund status (approved/rejected).

Table 36: Request Goal Cancellation & Refund – Component

3.2.7 Redeem Product After Goal Completion

Identification	Redeem Product
Type	**Module:** The module is responsible for handling the initiation of redemption and notification to the store.
Purpose	Give user choice of redemption, inform store and arrange dispatch.
Function	**Handles:** • Redeem validation • Store notification **Processes:** • The stock is checked, record is created so people can redeem the stock, and the store is notified. **Outputs:** • Request for confirmation and dispatch.
Subordinates	**Consists of subcomponents:** • Redemption Recorder • Store Notifier
Dependencies	**Relies on:** • Products table, Store module, DB updates **Interaction:** • Sets product.locked, creates redemption row
Interfaces	**External Interface:** None. **Internal Interface:** Redemption service, Store service, Database

	(redemptions, products). **User Interface:** • To redeem button (when goal is 100%). • Confirmation dialog. • Error if product unavailable. • Success with redemption ID and dispatch time.
Resources	**Requires:** DB writes, notification engine
Processing	**Following these steps:** 1. Verify that goal status = completed 2. Confirm store stock 3. Enter redemption record and inform the store.
Data	**Handles:** Redemption relies on goalId for goal identification, userId for ownership, status for progress, and dispatch_info for updates on dispatching.

Table 37: Redeem Product After Goal Completion – Component

3.2.8 Receive Notifications

Identification	Notification Engine
Type	**Module:** Responsible for event-based notifications.
Purpose	Send deposit confirmation, goal completion via email and in-app notifications.
Function	**Handles:** • Notifications • Sending emails and in-app messages **Processes:** • triggers, log deliveries **Outputs:** • Email sends, in-app notification entries
Subordinates	**Consists of subcomponents:** • Email Adapter • In-app Notification Handler • Scheduler
Dependencies	**Relies on:** • The library and the Notification table, along with the Goal table. **Interaction:** • The goals are being read by the Cron Jobs and enqueued.
Interfaces	**External Interface:** Email Library. **Internal Interface:** Notification module, Database (notifications), background jobs. **User Interface:** • In-app notifications. • Error logs of failed sends by admin.
Resources	**Requires:** • Quotas for emails sent with cron job.

Processing	**Following these steps:** • Scheduler queries due reminders • Compose notification content • Send via Email library
Data	**Handles:** The notificationId, notification type, and delivered_at timestamp are all recorded in notifications.

Table 38: Receive Notifications - Component

3.2.9 Track Delivery Using Map

Identification	Delivery Tracking
Type	**Module:** This module provides integration of maps services and tracking of couriers.
Purpose	Show delivery address and delivery status to the user.
Function	**Handles:** • Map rendering and location updates **Processes:** • Update DB **Outputs:** • Map view with location
Subordinates	**Consists of subcomponents:** • Map Renderer
Dependencies	**Relies on:** • Maps **Interaction:** • Add tracking information to delivery tracking table.
Interfaces	**External Interface:** Map **Internal Interface:** Delivery service, Database (delivery tracking). **User Interface:** • Display the map with item location and route. • Delivery statuses. • Estimated arrival time. • Errors of tracking when no tracking is available.
Resources	**Requires:** • Maps library (React leaflet)
Processing	**Following these steps:** • Receive tracking update • Update DB • Update client map
Data	**Handles:** ProductId, GoalId, DeliveryDate, DeliveryStatus, TrackingNumber and ShippingAddress are tracked.

Table 39: Track Delivery Using Map - Component

3.2.10 Store Registration

Identification	Store Registration
Type	**Module:** Handles Store onboarding
Purpose	Let Store submit application with necessary information for admin approval.
Function	**Handles:** • Upload info **Processes:** • Save info to storage, create store pending record, notify admin **Outputs:** • **store** record with status = pending, admin notification
Subordinates	**Consists of subcomponents:** • Validator
Dependencies	**Relies on:** • Storage (Neon), Store table **Interaction:** • Admin approval workflow
Interfaces	**External Interface:** Storage API for details. **Internal Interface:** Store service, Database (Store). **User Interface:** • Required info. • Validation errors. • A success message and pending-approval notice are displayed when submitting.
Resources	**Requires:** • Storage credentials
Processing	**Following these steps:** • Store submits details • Enter store record and status pending and notify admin.
Data	**Handles:** The application status and store profile fields are stored in store registration.

Table 40: Store Registration - Component

3.2.11 Store Login

Identification	Store Login
Type	**Module:** The module is responsible for authentication and access control for Handles.
Purpose	Verify Store account and allow access to features only within the store.
Function	**Handles:** • Credential verification, Approval status check **Processes:** • Ensure that it is authenticated by Clerk and that store status is approved. **Outputs:** • Access to session token/cookie and access to store dashboard.

Subordinates	**Consists of subcomponents:** • Role Checker, Session Manager
Dependencies	**Relies on:** • Clerk, Store table
Interfaces	**External Interface:** Auth service endpoints. **Internal Interface:** Role-based access system, Database (approval status). **User Interface:** • Email/password. • Pending/rejected account messages. • Alert to store dashboard.
Resources	**Requires:** • Clerk config, secure session store
Processing	**Following these steps:** • Store submits credentials • Authenticate via Clerk • Check if Store approved in DB • Create session & redirect
Data	**Handles:** The login uses storeId as an identifier, verifies approval status and fetches roles for dashboard access.

Table 41: Store Login - Component

3.2.12 Product Listing

Identification	Product Listing
Type	**Module:** This module is responsible for creating and managing products.
Purpose	Allow store to add and edit a product listing for layaway.
Function	**Handles:** • Handle product information, product images and stock management. **Processes:** • Validate inputs, upload images, insert/update product row **Outputs:** • Product in the marketplace – visible to users
Subordinates	**Consists of subcomponents:** • Image Uploader, Product Validator
Dependencies	**Relies on:** • Storage, Product table
Interfaces	**External Interface:** Storage (for images). **Internal Interface:** Product service, Database (product). **User Interface:** • Inputs: Price, Stock, Category, Description. • Renders a preview of the image in the image uploader. • Validation errors. • Success message with the Product ID.

Resources	**Requires:** • Storage space is provided and optimized for images via the library.
Processing	**Following these steps:** • Adds product information and uploads photos. • Review and upload photos. • Return a success when creating product DB record.
Data	**Handles:** All the products have the same properties: productId, name, price, images, stock, category.

Table 42: Product Listing - Component

3.2.13 Lock Product for Layaway

Identification	Lock Product
Type	**Module:** Provides the creation and management of products for goal.
Purpose	Lock in a price and use a product for a user's objective.
Function	**Handles:** • Product locking and lock prize. **Processes:** • Acquire DB **Outputs:** • Lock/goal confirmation or failure
Subordinates	**Consists of subcomponents:** • Stock Validator, Transaction Handler
Dependencies	**Relies on:** • Prisma transactions, Product & Goal tables
Interfaces	**External Interface:** None. **Internal Interface:** Product service, Goal service, Database transactions. **User Interface:** • Locked indicator on goal page. • Error for lock failure.
Resources	**Requires:** • DB transaction support
Processing	**Following these steps:** • Select product for lock • If available, locked=true, locked_by=userId, lock_expires_at, lock_conditions. • Fail, commit or rollback
Data	**Handles:** A lock/goal has the following: productId, userId, targetAmount, createdAt timestamp, endDate, inStock status.

Table 43: Lock Product for Layaway - Component

3.2.14 Confirm Delivery & Final Payment Release

Identification	Confirm Delivery & Payout
Type	**Module:** Module is responsible for confirm delivery in the store.
Purpose	On completion of delivery, release funds to stores.
Function	**Handles:** • Ability to confirm and initiate payouts. Delivery confirmation/payout initiation. **Processes:** • Verify Delivered, Compute Platform fees, Create Payout, Update DB **Outputs:** • Payout record, status update, notifications
Subordinates	**Consists of subcomponents:** • Delivery Verifier, Payout
Dependencies	**Relies on:** • Redemptions, Store account
Interfaces	**External Interface:** Map. **Internal Interface:** Payout service, Receipt service. **User Interface:** • Store buttons: dispatch, delivered updates. • Admin payout controls. • Notifications (user/store). • Errors for unsuccessful payouts.
Resources	**Requires:** • Secure account setup details for payment gateways, secure payout details.
Processing	**Following these steps:** • Store marks delivered in system receives confirmation • Check if payout criteria are met (hold period criteria met). • Starting payout and updating DB
Data	**Handles:** The userId, amount to be targeted and/or the delivery/payout status is stored in the module.

Table 44: Confirm Delivery & Final Payment Release - Component

3.2.15 View Store Dashboard & Earnings

Identification	Store Dashboard
Type	**Module:** The Metrics and Reports are stored in the Aggregates module.
Purpose	Show sales, product management and analysis.
Function	**Handles:** • Data consolidation and graphing. **Processes:** • Answer queries on transactions and redemptions, calculate KPIs.

	Outputs: • Adjust the size of the Dashboard panels, and export to PDF.
Subordinates	**Consists of subcomponents:** • Aggregator, Table Data Provider
Dependencies	**Relies on:** •Deposit table, Goal, Product
Interfaces	**External Interface:** None **Internal Interface:** Reporting service, Database (Deposit, Product, Goal). **User Interface:** • Charts, filters. • PDF export controls. • Empty state and error views.
Resources	**Requires:** • DB aggregation performance
Processing	**Following these steps:** • Query relevant tables for storeId • Compute totals and trends • Create charts and export as needed
Data	**Handles:** The product management and revenue are used to create reports from the dashboard.

Table 45: View Store Dashboard & Earnings - Component

3.2.16 Approve or Reject Store Applications

Identification	Store Approval
Type	**Module:** The module is responsible for admin review and approval.
Purpose	Let Admins view and approve or deny the store details.
Function	**Handles:** • Preview details and make decisions. **Processes:** • Update store.status, notify store **Outputs:** • The Status change (approved/rejected) and notification are provided.
Subordinates	**Consists of subcomponents:** • Details Viewer, Approver UI
Dependencies	**Relies on:** • Store details, Store table
Interfaces	**External Interface:** Neon storage viewer, e-mail library. **Internal Interface:** Admin service, Notification system, Database (Store). **User Interface:** • Details viewer. • Approve/Reject.

	• Confirmation modal. • The status messages are sent to the store.
Resources	**Requires:** • Admin authentication
Processing	**Following these steps:** • The Admin opens up the store record that has been pending. • Review details • Allow, deny and inform store
Data	**Handles:** Admin updates the storeId application and its status.

Table 46: Approve or Reject Store Applications – Component

3.2.17 Monitor Transactions & Resolve Disputes

Identification	Dispute Management
Type	**Module:** Handles dispute intake and resolution
Purpose	Allow admin to see transactions disputed and undertake refunds or other corrective measures.
Function	**Handles:** • The use of evidence is the basis for the investigation process and procedures of getting the refund. **Processes:** • Validate evidence, compute refund amounts **Outputs:** • Provide dispute resolution information and notifications.
Subordinates	**Consists of subcomponents:** • Evidence Validator, Refund Executor
Dependencies	**Relies on:** • Deposit table, Transactions
Interfaces	**External Interface:** Payment Gateway API. **Internal Interface:** Dispute service, Database, Notification system, Wallet. **User Interface:** • Upload evidence to dispute the form. • Admin case review. • Notifications for status changes. • Errors for invalid evidence or failed refunds.
Resources	**Requires:** • Admin access, logs
Processing	**Following these steps:** • Admin examines the evidence and dispute • Decide action (refund/deny) • If refund, invoke payment gateway refund and update DB

Data	**Handles:** Disputes track disputeId, evidence submitted, original_payment_id and resolution notes

Table 47: Monitor Transactions & Resolve Disputes – Component

3.2.18 Manage Price Lock & Inflation Policy

Identification	Price Lock & Policy
Type	**Module:** Handles policy decisions about price locks and adjustments
Purpose	Allow admin to manage and apply price lock adjustments in case of inflation or store requests.
Function	**Handles:** • Review store requests for price change, apply policy rules **Processes:** • Analyze timelines, propose adjustments, apply agreed changes **Outputs:** • Updated lock terms, notifications to affected users.
Subordinates	**Consists of subcomponents:** • Policy Evaluator, Negotiation Helper
Dependencies	**Relies on:** • Goal table, Store request, Admin approval
Interfaces	**External Interface:** None. **Internal Interface:** Policy management, Goal, Database (goal, product). **User Interface:** • Admin policy controls. • Notifications for affected users. • Error messages for conflicting rules.
Resources	**Requires:** • The policy config files and admin UI.
Processing	**Following these steps:** • Admin's view: store request or inflation event. • Analyze the impacted goals and users. • Apply adjustment or schedule refunds and notify parties.
Data	**Handles:** Policy updates include the lock_terms, adjustment_notes, and affected affected_goal_ids.

Table 48: Manage Price Lock & Inflation Policy - Component

3.2.19 Generate System Reports

Identification	Reports
Type	**Module:** The module is responsible for aggregate and export of system reports.
Purpose	Produce PDF reports for financials.

Function	**Handles:** • Data aggregation, filtering, export formatting **Processes:** • Compile queries, format to PDF **Outputs:** • Downloadable report files
Subordinates	**Consists of subcomponents:** • Query Engine, Export Builder
Dependencies	**Relies on:** • All DB tables records.
Interfaces	**External Interface:** Cloud/System storage **Internal Interface:** Reporting service, Database **User Interface:** • Filters (date, store, type). • Generate/download PDF. • Errors for failed or oversized reports.
Resources	**Requires:** • DB read throughput
Processing	**Following these steps:** • Admin requests report with filters • Query DB and generate file
Data	**Handles:** Reports use database record for exports.

Table 49: Generate System Reports & Audit Logs - Component

3.2.20 Payment Processor

Identification	Payments Integration
Type	**Module:** Internal and External service integration for payments.
Purpose	Interface with Payment gateway and wallet for payment intents, webhooks, refunds and payouts.
Function	**Handles:** • Create payment intents, handle webhooks, initiate refunds/payouts. **Processes:** • Validate webhooks, update DB transactions on success/failure. **Outputs:** • Transaction IDs, payment statuses
Subordinates	**Consists of subcomponents:** • Webhook Listener, Payout Manager
Dependencies	**Relies on:** • API keys, connected store accounts and wallet.
Interfaces	**External Interface:** Payment Gateway API **Internal Interface:** Deposit service, Payout service, Database (deposits, payouts), wallet.

	User Interface: • Payment form. • Webhook-driven updates. • Error messages for declines. • Receipt links.
Resources	**Requires:** • Secure webhook endpoints, API keys, wallet
Processing	**Following these steps:** • Create payment intent on deposit initiation • Confirm via client or webhook callback • Record transaction and generate receipt
Data	**Handles:** Payment logs the depositId, transactionId and payment status.

Table 50: Payment Processor – Component

3.2.21 Notification Engine

Identification	Notification Engine
Type	**Module:** System service for messaging
Purpose	Central service to create and deliver email and in-app notifications.
Function	**Handles:** • Compose messages, queue, send via email, and log deliveries **Processes:** • If the event does not occur, the retry logic on failure is not implemented. **Outputs:** • Delivered emails and logs
Subordinates	**Consists of subcomponents:** • Email Adapter
Dependencies	**Relies on:** • Email provider (library), DB for notification records
Interfaces	**External Interface:** None **Internal Interface:** Notification service (Nodemailer), Database (notification logs), scheduling engine. **User Interface:** • Notification center. • Preference settings for email.
Resources	**Requires:** • Email quotas
Processing	**Following these steps:** • Receive event • Build message content and recipients • Dispatch and log result

Data	**Handles:** Notifications store notification logs, the type, recipients, and delivered_at timestamp.

Table 51: Notification Engine - Component

3.2.22 Receipt Generator

Identification	Receipt Generator
Type	**Module:** Document generation service
Purpose	Create PDF receipts for deposits.
Function	**Handles:** • Generate receipts from transactions and render to pdf. **Processes:** • Both rendering template and PDF generation. **Outputs:** • Receipt PDF
Subordinates	**Consists of subcomponents:** • PDF Engine, Template Manager
Dependencies	**Relies on:** • DB transaction records
Interfaces	**External Interface:** Neon API **Internal Interface:** Receipt generator, Payment system, Database (Deposit, Order). **User Interface:** • View/download receipts. • Print option. • Errors for generation or upload failures.
Resources	**Requires:** • A PDF rendering library, storage credentials
Processing	**Following these steps:** • Load transaction data • Render template and generate PDF • Download PDF
Data	**Handles:** The receipt generator receives data from transactions to build the receipt, saves the receipt to be able to retrieve it later, and stores generated_at as the time the receipt was generated.

Table 52: Receipt Generator - Component

3.2.23 Apply Coupon

Identification	**Apply Coupon**
Type	**Module:** Handles validation and application of promotional codes to goals.

Purpose	Verify coupon validity and apply discounts to the user's target amount.
Function	**Handles:** • Coupon code parsing and rule validating. **Processes:** • Verify expiry, usage limits, and new user constraints. • Calculate discount amount. **Outputs:** • Discounted target amount and success confirmation, or validation error.
Subordinates	**Consists of subcomponents:** • There is a Coupon Validator, Discount Calculator and Usage Tracker.
Dependencies	**Relies on:** • Prisma transactions, Coupon & CouponUsage tables, Goal table.
Interfaces	**External Interface:** None. **Internal Interface:** Goal service, Database transactions. **User Interface:** • Coupon input field on goal creation. • Discounted price display. • Error messages for invalid/expired codes.
Resources	**Requires:** • The DB transaction support, Server Date/Time for Expiry checks.
Processing	**Following these steps:** • Retrieve coupon from DB matching the entered code. • Check if expired, usage limit reached, or restricted to new users. • If valid, calculate discount against base price and Output final amount. • Upon goal creation, create a CouponUsage record.
Data	**Handles:** A code is stored in a coupon that contains the code, description, discount value, usageLimit, forNewUser flag, isPublic flag, userId (when it's a private/apology) and expiresAt timestamp.

Table 53: Apply Coupon - Component

3.2.24 Digital Wallet

Identification	**Digital Wallet Management**
Type	**Module:** Handles user balances, deposits, and withdrawal.
Purpose	Handle the user's saved money and manage the withdrawal to external bank accounts.
Function	**Handles:** • Balance checks, fund deductions, withdrawal logging. **Processes:** • Wallet update, transaction ledger creation, triggering of notifications. **Outputs:** • Success confirmation, transaction history, updated balance.
Subordinates	**Consists of subcomponents:** • Balance Validator, Transaction Ledger, Notification Engine
Dependencies	**Relies on:** • Prisma transactions, Wallet & WalletTransaction tables, Notification Service
Interfaces	**External Interface:** None. **Internal Interface:** Authentication service, Admin payout queue.

	User Interface: • Wallet balance display. • Withdrawal form. • Transaction history list.
Resources	**Requires:** • DB transaction support
Processing	**Following these steps:** • Validate user's withdrawal request amount against current wallet balance. • If sufficient, deduct amount from wallet balance. • Create a "withdrawal" transaction record containing user's bank details. • Trigger confirmation email/in-app notification to user. • Commit or rollback on failure.
Data	**Handles:** In a wallet there are two things: userId and balance. The wallet transactions include walletId, amount, type (withdrawal, refund_credit, etc.), description, referenceId and createdAt timestamp.

Table 54: Digital Wallet - Component

3.2.25 Support & Complaints

Identification	**Support & Complaints**
Type	**Module:** Handles the filing, tracking, and resolution of user and store complaints.
Purpose	Offer a proper system for complaints about products, deliveries or store behavior.
Function	**Handles:** • Ticket creation, 7-day rule validation, store/goal linking. **Processes:** • Rule Enforcement, DB Insertion, Automated Notifications. **Outputs:** • Created complaint ticket, alerts to Admins/Users.
Subordinates	**Consists of subcomponents:** • Notification Engine, Ticket Manager
Dependencies	**Relies on:** • Prisma transactions, Complaint, Goal, Delivery, and Store tables
Interfaces	**External Interface:** None. **Internal Interface:** Goal service, Admin dashboard. **User Interface:** • Complaint submission form (title, type, description, linked goal). • Ticket status dashboard for users/stores.
Resources	**Requires:** • DB transaction support
Processing	**Following these steps:** • Receive complaint details and optional linked goalId. If the goal is connected to a delivered goal, check that it's been delivered within the 7-day period. • Create complaint record with status "OPEN" and associate with target store/user.

	• Send Acknowledgement e-mail to filer. • Rollback or commit on failure.
Data	**Handles:** A complaint contains AdminNotes, createdAt, targetStoreId, targetUserId, type, description, goalId, and status (OPEN, RESOLVED), and the filerUserId or filerStoreId.

Table 55: Support & Complaints – Component

3.2.26 Rider Profile & Status Management

Identification	**Rider Profile & Status Management**
Type	**Module:** Register and verify riders, update rider profiles, and mark riders as active/inactive.
Purpose	For accurate rider credentials, only approved riders should be allowed to operate and availability should be tracked in real time for dispatch.
Function	**Handles:** • Registration requests, document uploads. **Processes:** • Admin approval workflows, availability broadcasting to the dispatch engine. **Outputs:** • Verified rider profile, active status signals for the system.
Subordinates	**Consists of subcomponents:** • Profile Manager, Verification Engine, Status Broadcaster
Dependencies	**Relies on:** • Prisma transactions, User, and RiderProfile tables.
Interfaces	**External Interface:** Cloud storage (Neon Storage) for ID/License images. **Internal Interface:** Admin Dashboard. **User Interface:** • Registration form (CNIC, Vehicle Plate, License). • Status toggle switch (Go Online / Go Offline).
Resources	**Requires:** • DB transaction support, File storage infrastructure.
Processing	**Following these steps:** • Receive rider details and ID/License images. • Create RiderProfile record with status "PENDING_APPROVAL". • Admin reviews documents and updates status to "APPROVED". • Rider logs in and toggles isOnline to true. • Initializes tracking and permits routing.
Data	**Handles:** Rider attributes including phoneNumber, vehicleType, vehiclePlate, cnicNumber, licenseNumber, idImageUrl, status (PENDING_APPROVAL, APPROVED, SUSPENDED), and isOnline boolean.

Table 56: Rider Profile & Status Management - Component

3.2.27 Rider Earnings & Payouts

Identification	Rider Earnings & Payouts
Type	**Module:** Processes fees calculations, wallet ledger, and withdrawal requests from riders.
Purpose	To accurately track rider compensation per successful delivery and facilitate secure fund transfers to their bank accounts.
Function	**Handles:** • Fee calculation post-delivery, withdrawal requests. **Processes:** • Balance incrementing, payout record creation, ledger balancing. **Outputs:** • Updated wallet balance, generated payout receipts.
Subordinates	**Consists of subcomponents:** • Earnings Calculator, Payout Manager
Dependencies	**Relies on:** • Prisma transactions, RiderProfile, RiderPayout and Delivery tables.
Interfaces	**External Interface:** Payment Gateway (e.g., Stripe Connect). **Internal Interface:** Admin Finance Dashboard. **User Interface:** • Earnings summary dashboard. • Withdrawal Request Form and transaction history list.
Resources	**Requires:** • DB transaction support, Secure financial processing API.
Processing	**Following these steps:** • Upon a Delivery reaching "DELIVERED" status, calculate the delivery fee. • Increment totalEarnings and walletBalance in the Rider's profile. • Rider requests a withdrawal specifying an amount. • System checks for enough balance, subtracts it, and generates a RiderPayout set to "PENDING" status. • The system processes are passed through the gateway and set to "TRANSFERRED".
Data	**Handles:** Financial records include the amount, status (PENDING, TRANSFERRED), type (EARNING, WITHDRAWAL), walletBalance, totalEarnings, and transferredAt timestamps.

Table 57: Rider Earnings & Payouts - Component

3.2.28 Delivery Execution & Proof of Delivery (PoD)

Identification	Delivery Execution & Proof of Delivery (PoD)
Type	Module: Handles real-time location tracking, store pickup verification, and customer drop-off validation.
Purpose	To enable clear visibility of the delivery life-cycle.
Function	**Handles:** • Live updates, photo proof uploads. **Processes:** • Location logging, Status transitions. **Outputs:**

	• Live map updates, confirmed Proof of Delivery records.
Subordinates	**Consists of subcomponents:** • Location Tracker,
Dependencies	**Relies on:** • Prisma transactions, Delivery, DeliveryTracking table.
Interfaces	**External Interface:** Map. **Internal Interface:** Notification Engine (to alert customer). **User Interface:** • Live map view.
Resources	**Requires:** • Device access, DB transaction support, File storage.
Processing	**Following these steps:** • Rider arrives at Store. Store verifies and updates status to "DISPATCED". • Show live tracking of rider travel. • Rider arrives at Customer and upload photo. • System verifies, creates ProofOfDelivery, and marks Delivery as "DELIVERED".
Data	**Handles:** Tracking coordinates (latitude, longitude), proofImageUrl, and transitions Delivery status.

Table 58: Delivery Execution & Proof of Delivery (PoD) - Component

4. User Interface Design

4.1 Section Overview

This section provides detailed descriptions of the user interface design: overall design principles, visual conventions, GUI components and detailed descriptions of the screens. It describes how the users will interact, move around and do things like setting up a savings objective, making deposits, looking at receipts and tracking deliveries, handling products in the store and doing admin approvals.

This section is structured into four parts:

1. Rules of Interface Design – The standards and UI principles adopted.
2. GUI Components – These are the basic elements for creating the GUI system.
3. Detailed Screen Descriptions – Description of all user, store and admin screens.
4. Screen Flow Overview – Navigation structure and transitions.

4.2 Interface Design Rules

The interface design is based on best practices, which ensures the user-friendly and intuitive experience:

- Consistent color theme
- The financial transactions process is as simple as possible.
- Mobile-first responsive layout
- These accessibility features are included: readable fonts & color contrast.

4.3 GUI Components

These are the components of the visual system used throughout the system.

4.3.1 Buttons

- Primary Buttons – These are used for primary actions, such as “Set Goal”, “Deposit”.
- Secondary Buttons – used for navigation.
- Danger Buttons –used for cancellation/refund actions.

4.3.2 Input Fields

- Standardized Material UI text fields.
- Dropdowns for:
    - Shop
    - About
    - Contact Us

4.3.3 Cards

Used for:

- Goal overview
- Product listings
- Store dashboard analytics

Each card includes:

- Title
- Preview image
- Action buttons
- Status indicators

4.3.4 Navigation Components

- **navigation bar** (mobile)
- **Side navigation menu** (store/admin dashboards)

4.3.5 Modal Dialogs

Used for:

- Deposit confirmations
- Cancel goal
- Refund request
- Store approval

4.3.6 Charts

- Line charts
- Bar chart

4.4 Detailed Screen Descriptions

4.4.1 Sign Up

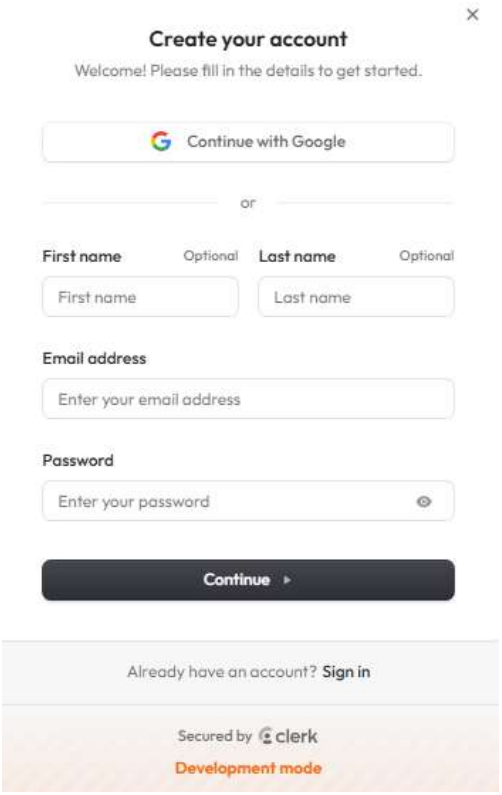


Figure 13: Sign Up

4.4.2 Sign In

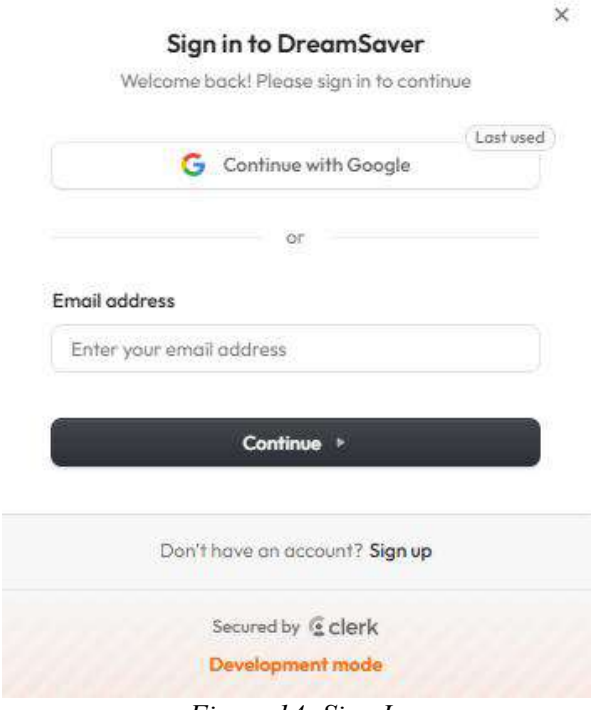


Figure 14: Sign In

4.4.3 Home Page 1

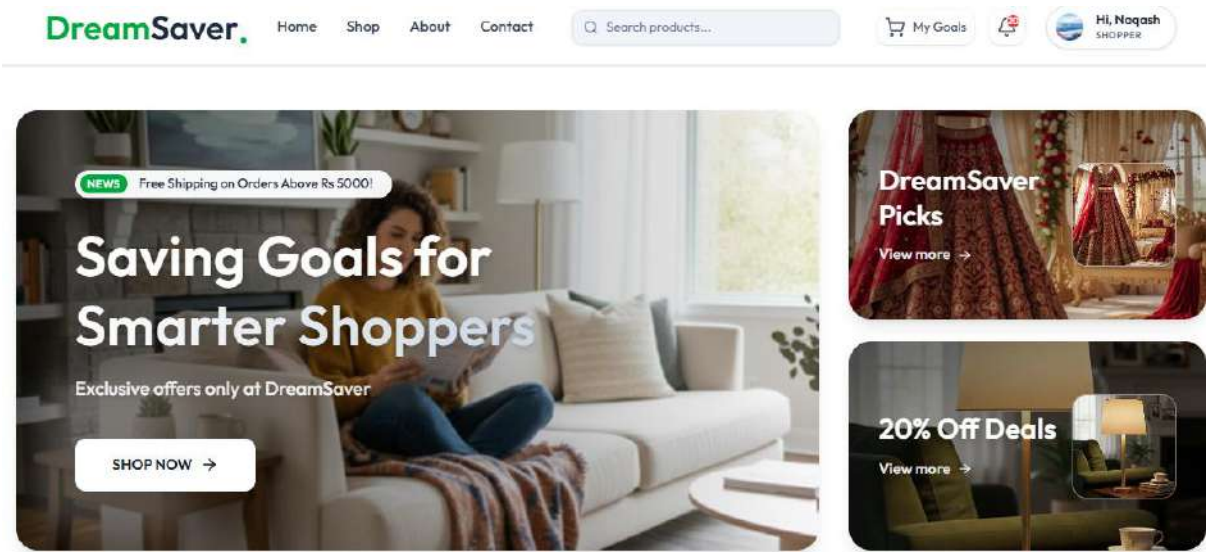


Figure 15: Home Page 1

4.4.4 Home Page 2

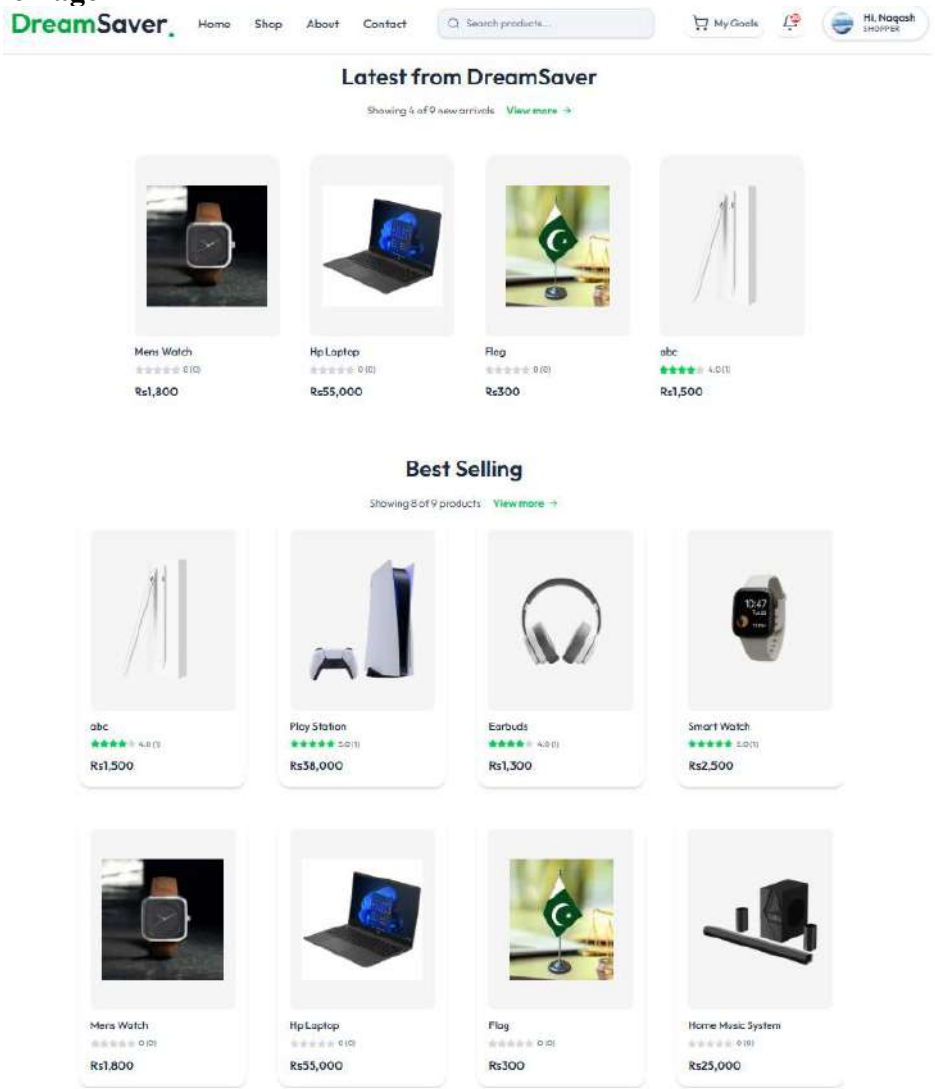


Figure 16: Home Page 2

4.4.5 Home Page 3

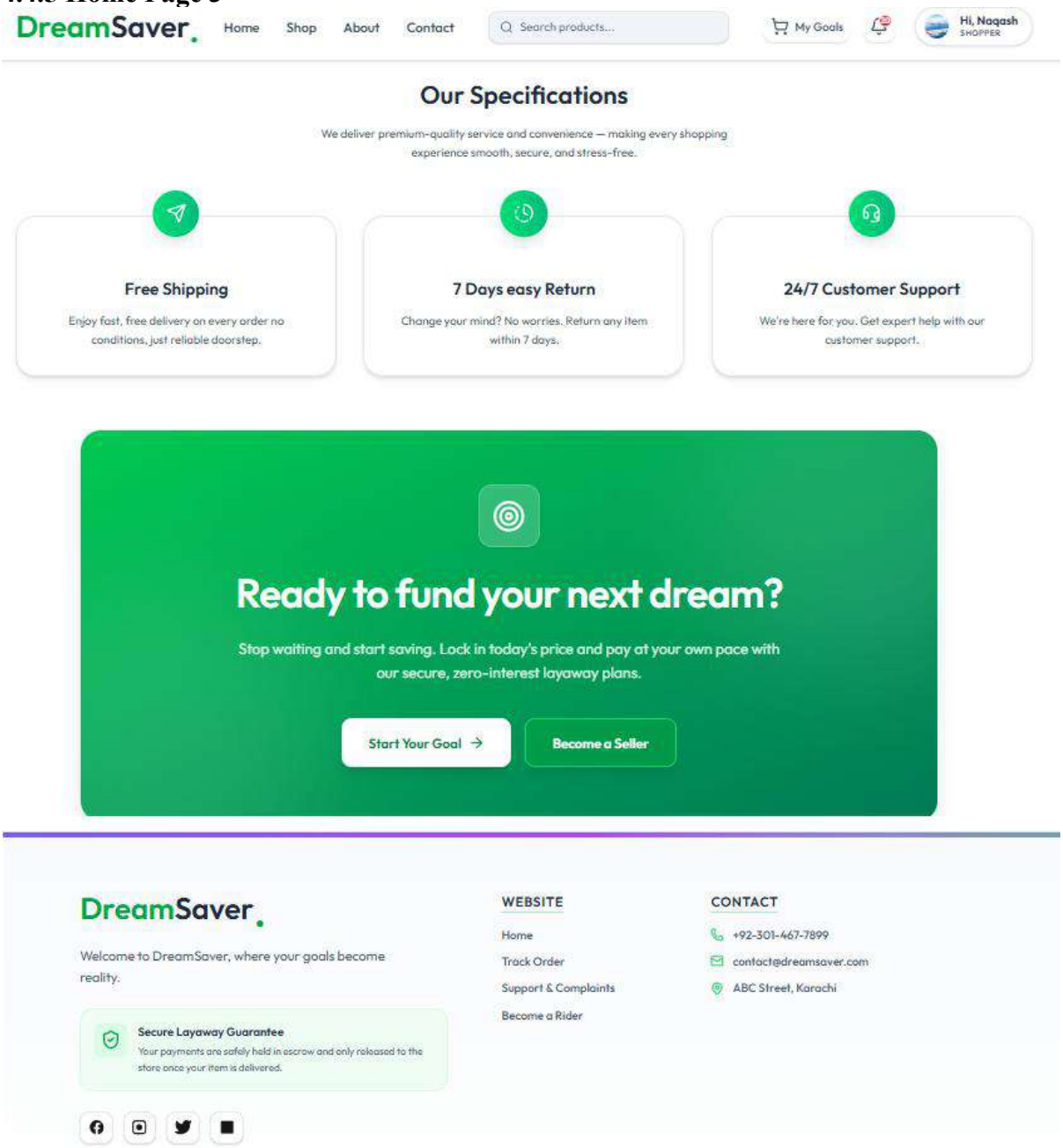


Figure 17: Home Page 3

4.4.6 Shop

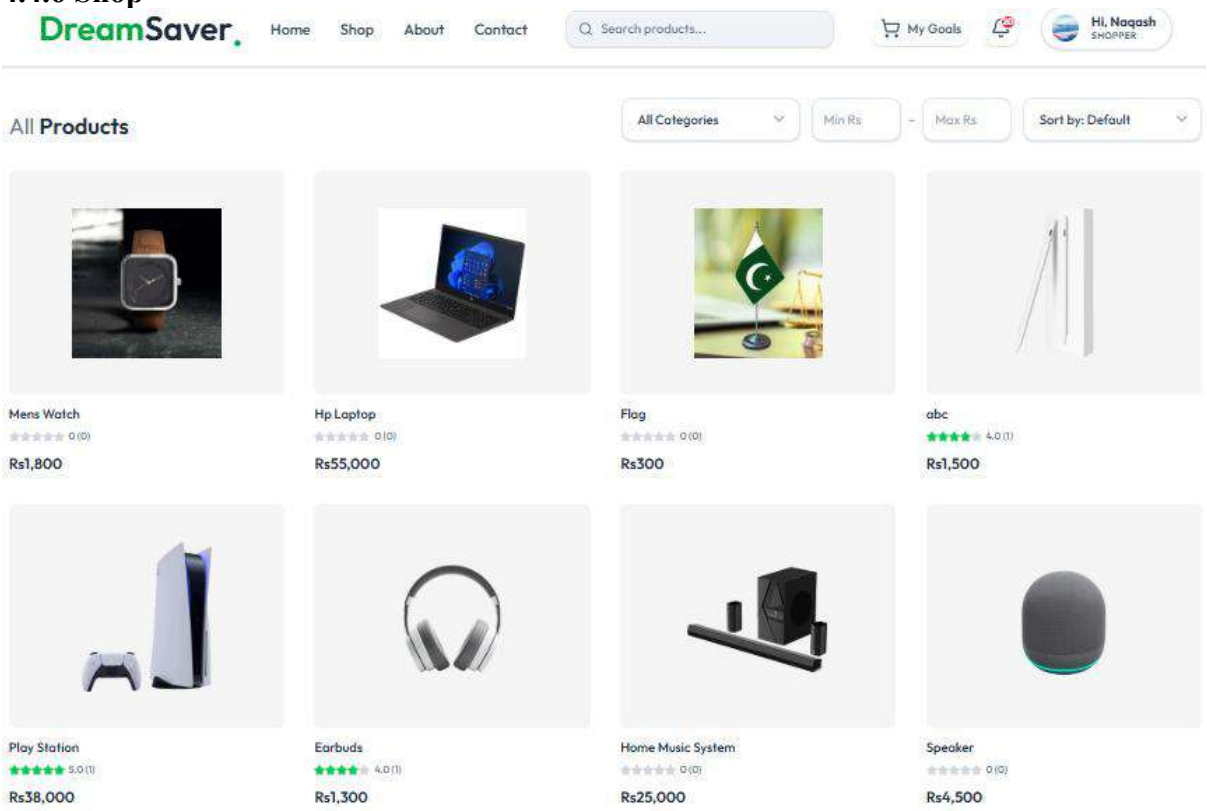


Figure 18: Shop Page

4.4.7 About

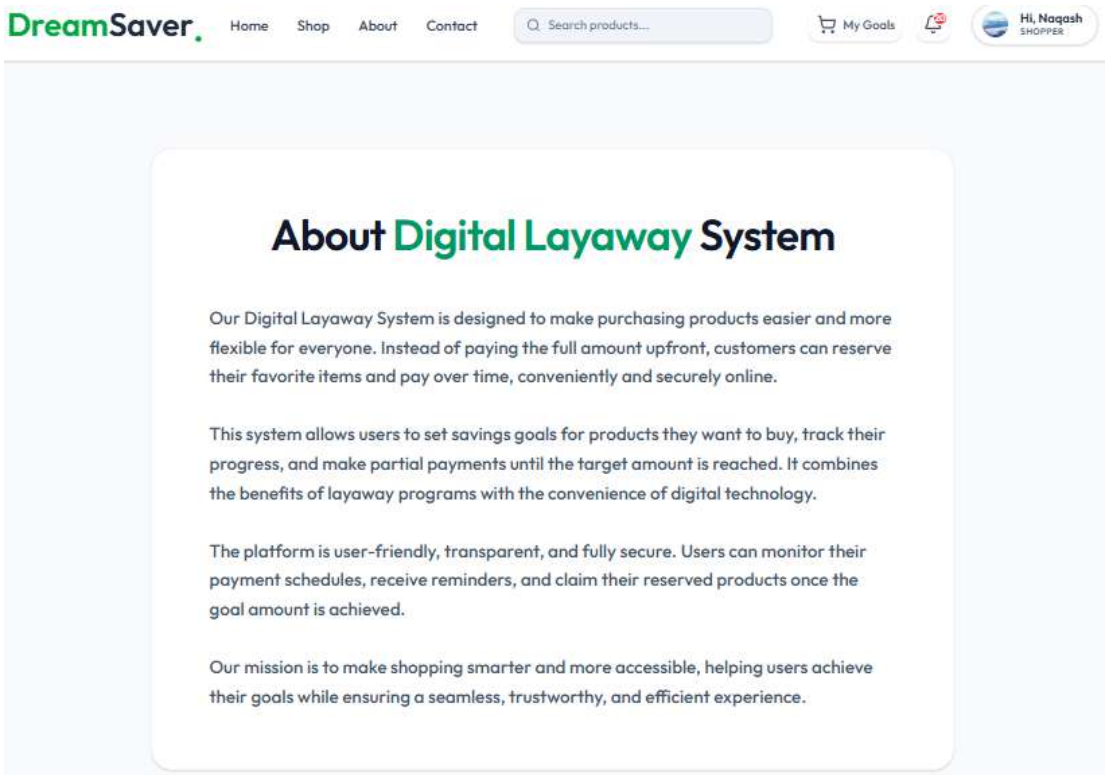


Figure 19: About

4.4.8 Contact us

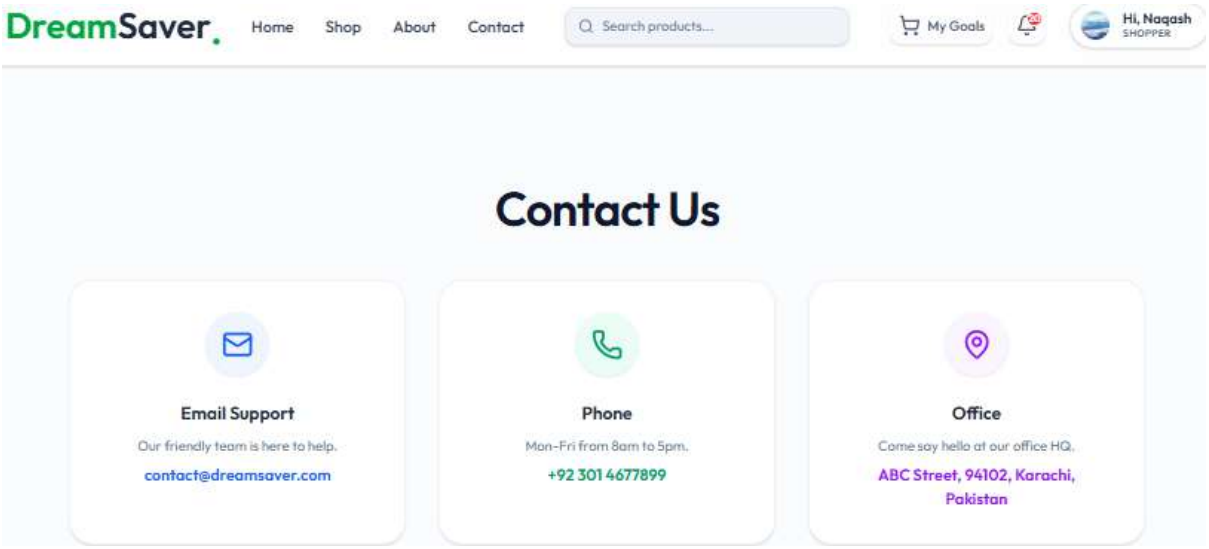


Figure 20: Contact Us

4.4.9 Product

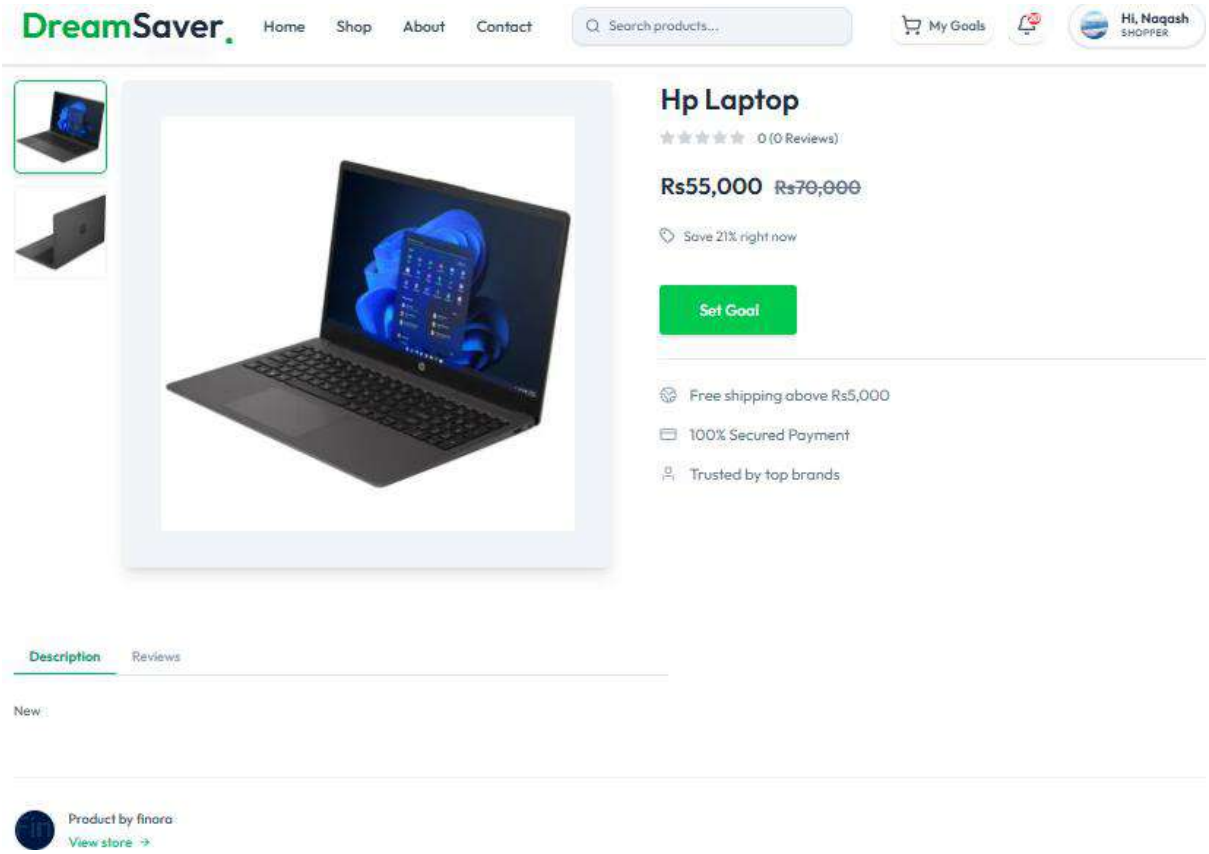


Figure 21: Product

4.4.10 Set Goal

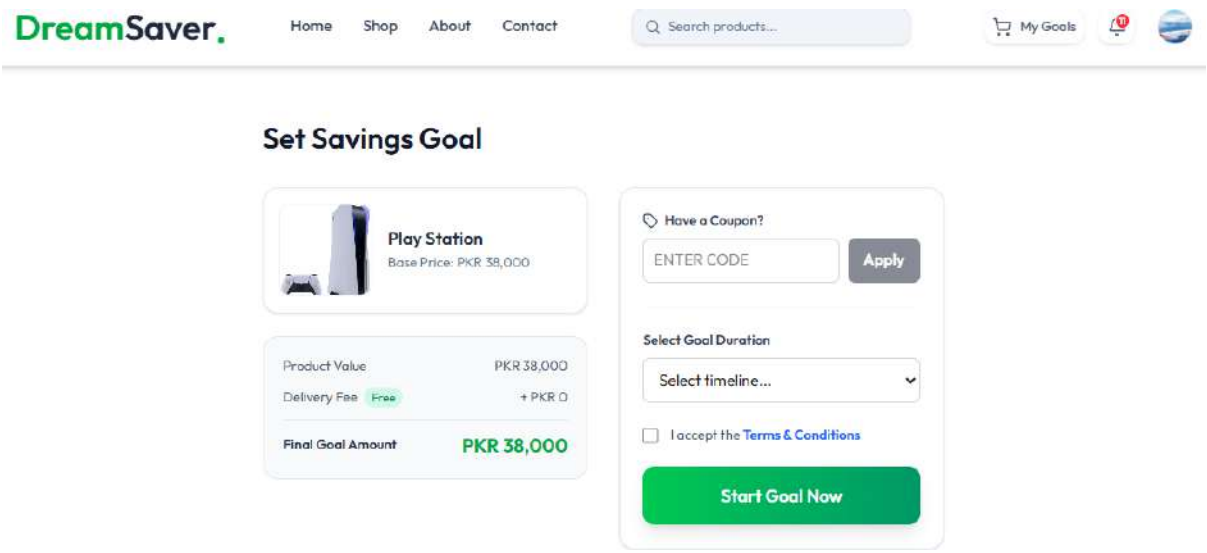


Figure 22: Set Goal

4.4.11 Goal Progress Page

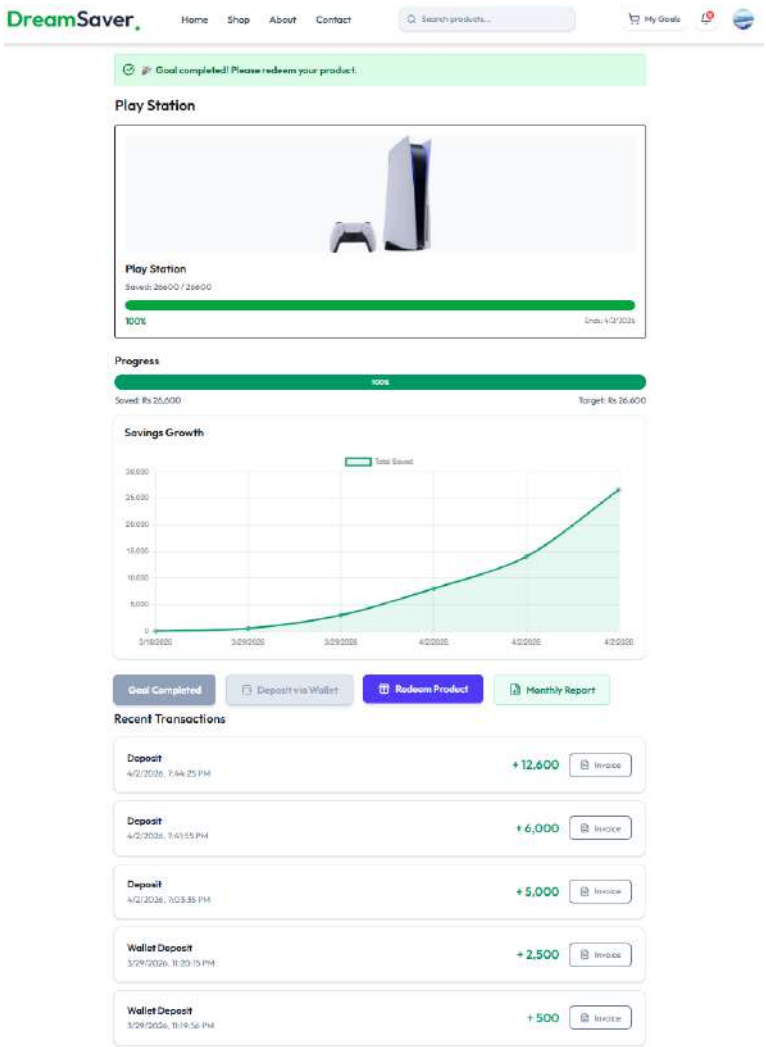


Figure 23: Goal Progress

4.4.12 Store Dashboard

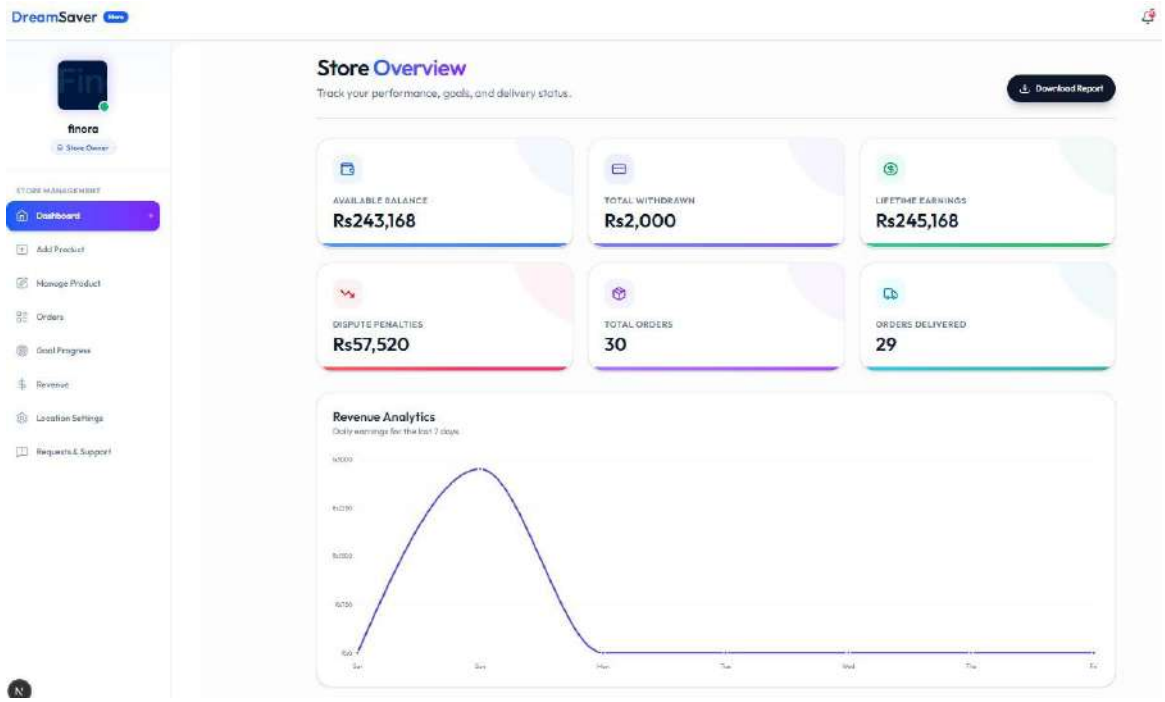


Figure 24: Store Dashboard

4.4.13 Store Add Product

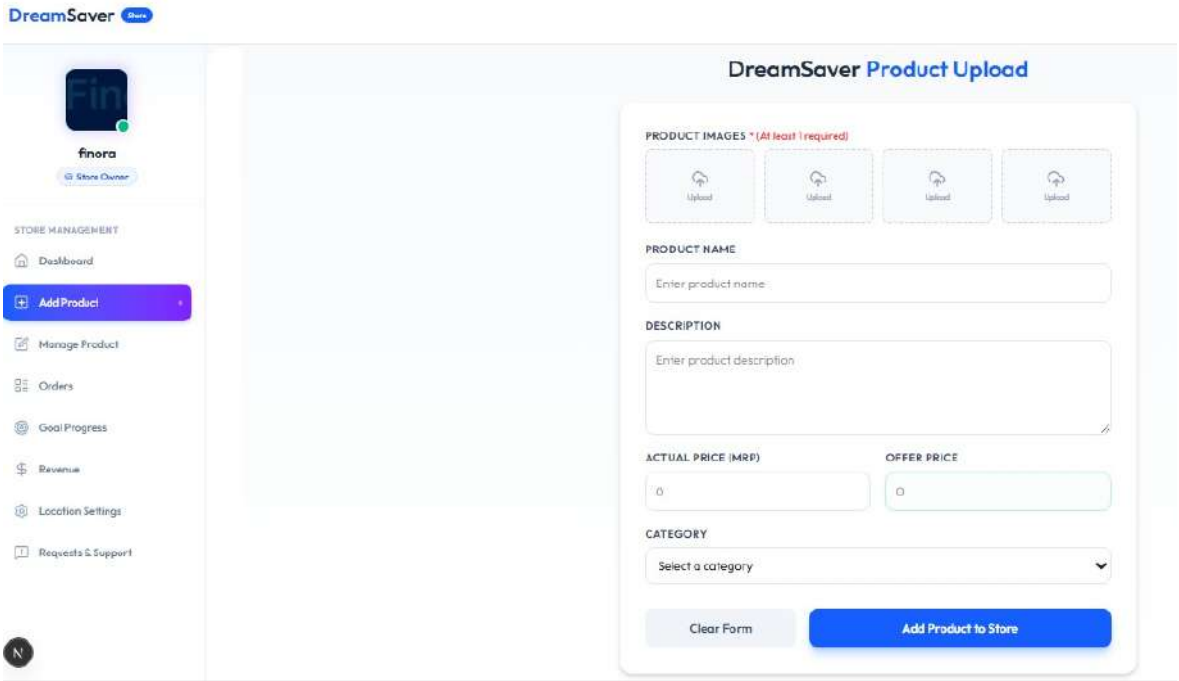


Figure 25: Store Add Product

4.4.14 Store Manage Product

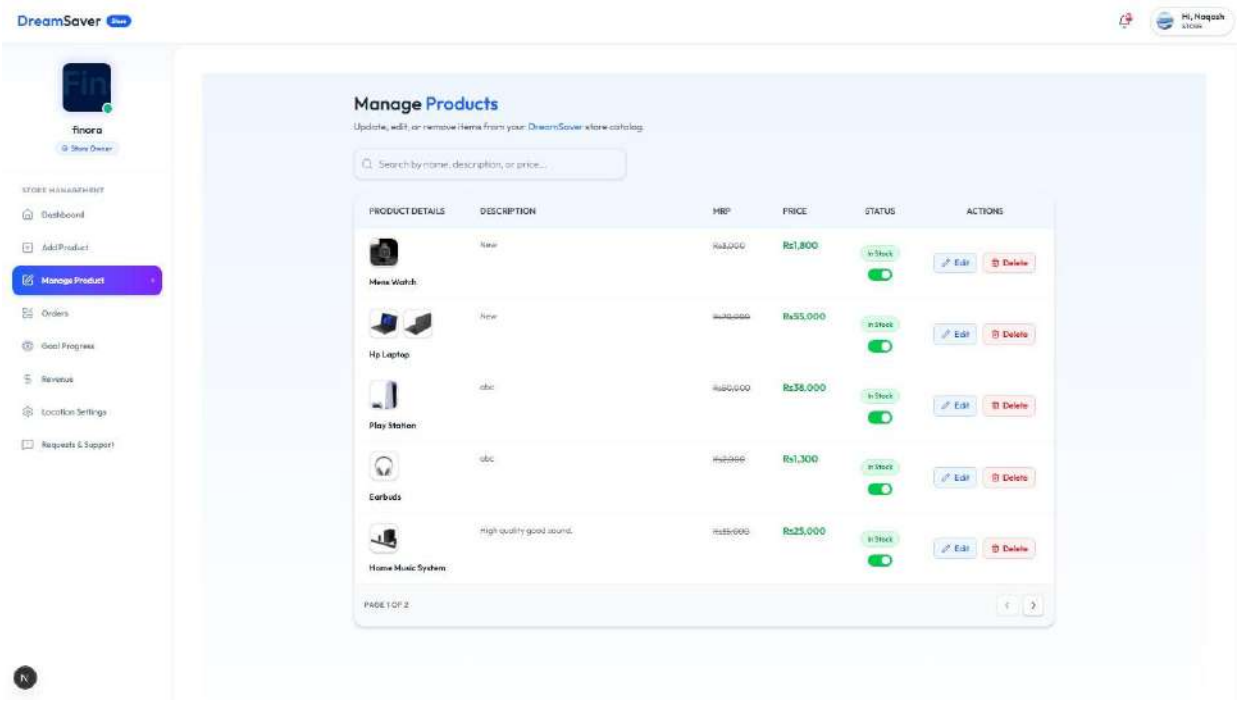


Figure 26: Store Manage Product

4.4.15 Store Order

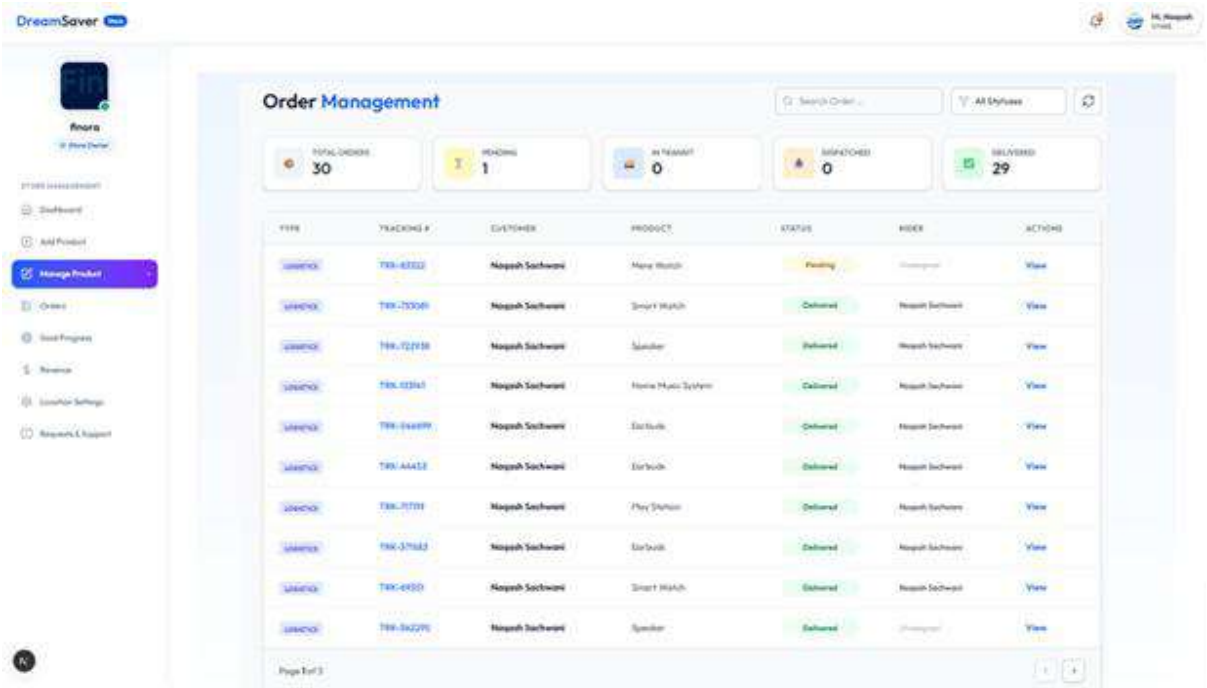


Figure 27: Store Order

4.4.16 Admin Dashboard

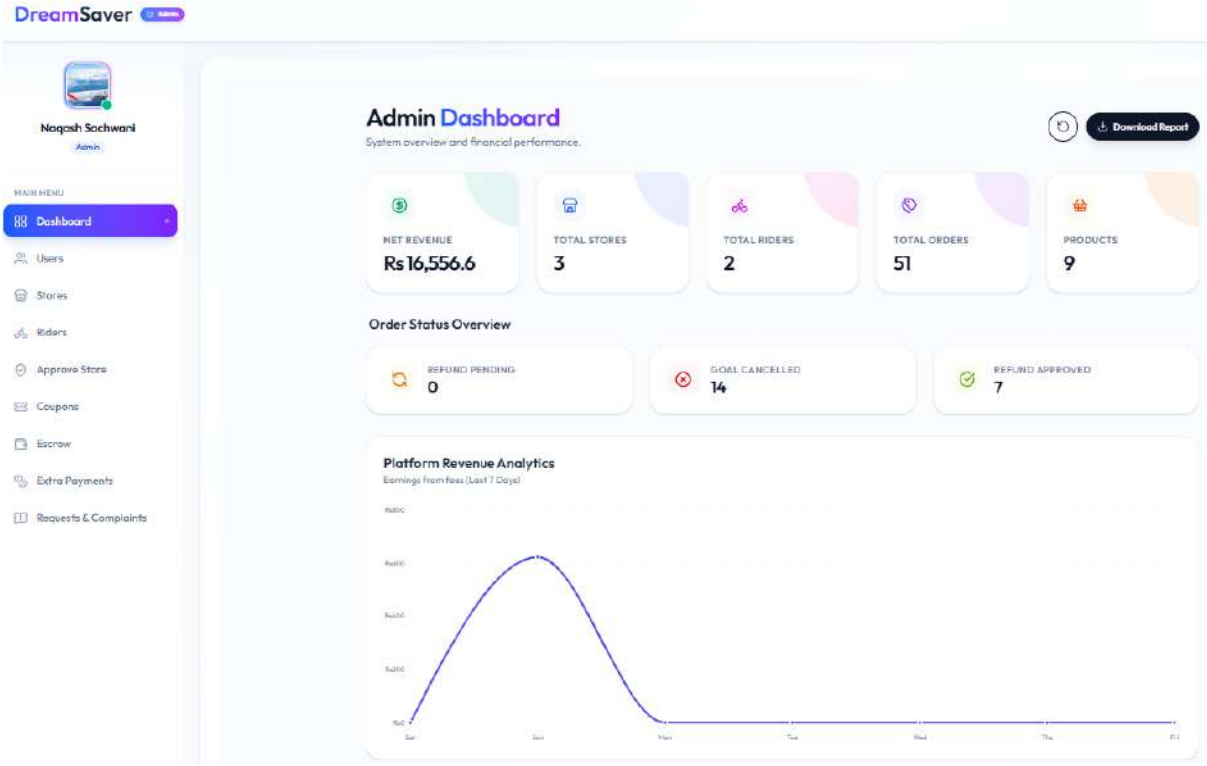


Figure 28: Admin Dashboard

4.4.17 Admin Store Management

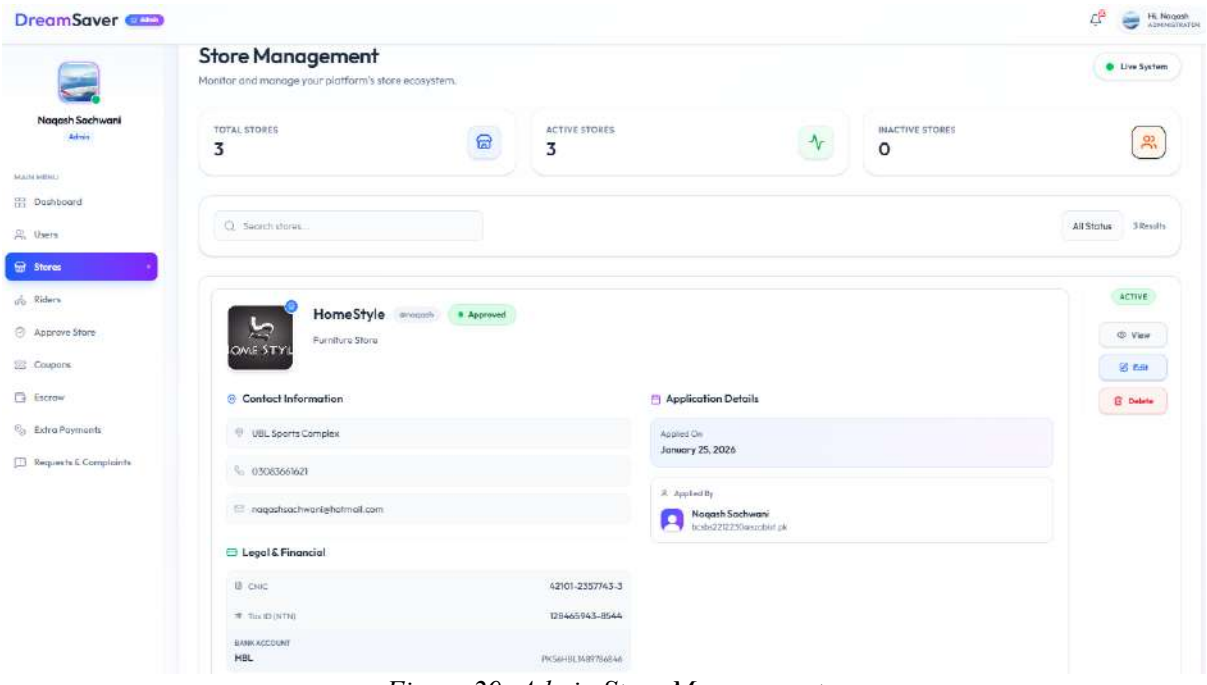


Figure 29: Admin Store Management

4.4.18 Admin Store Approval

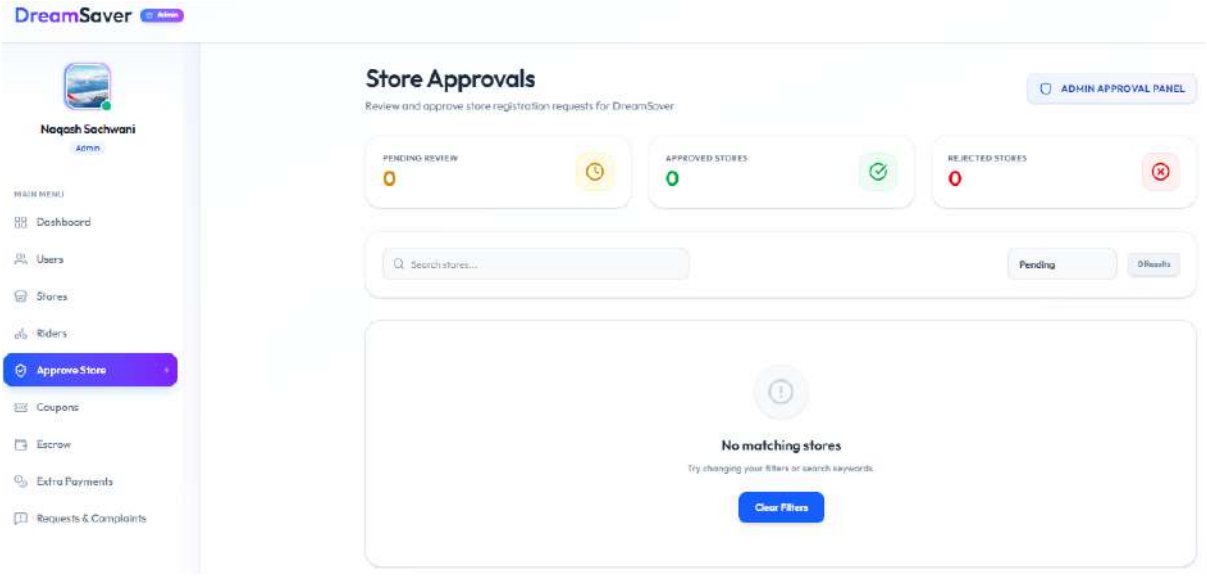


Figure 30: Admin Store Approval

4.4.19 Coupon Creation

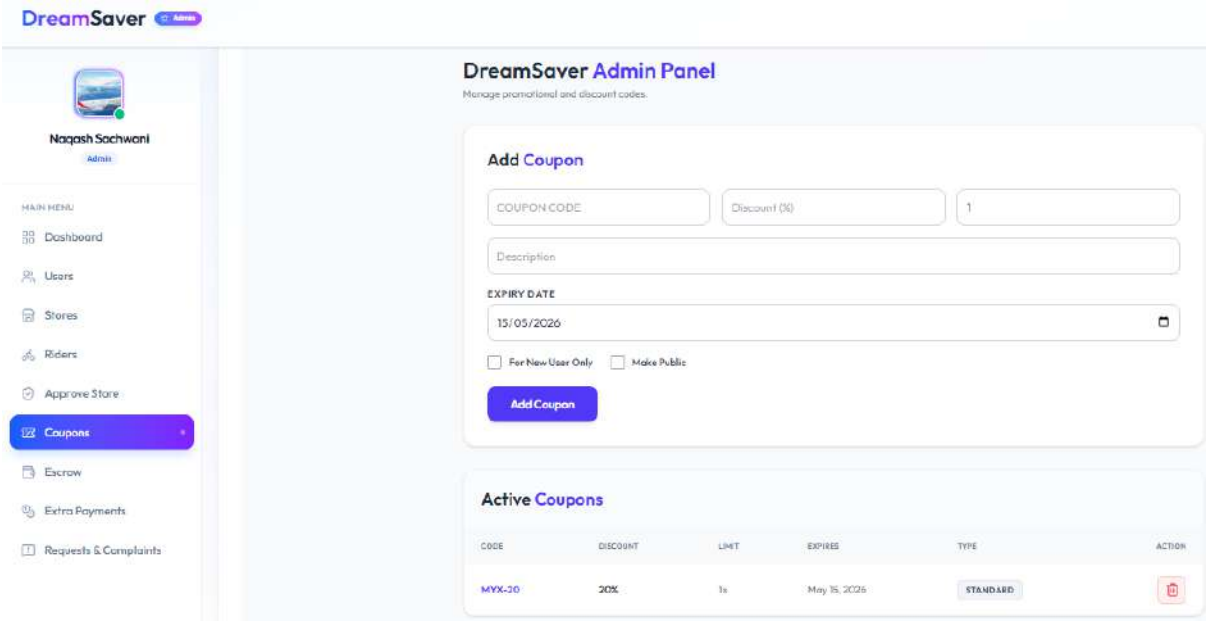


Figure 31: Coupon Creation

4.4.20 Goal Cart

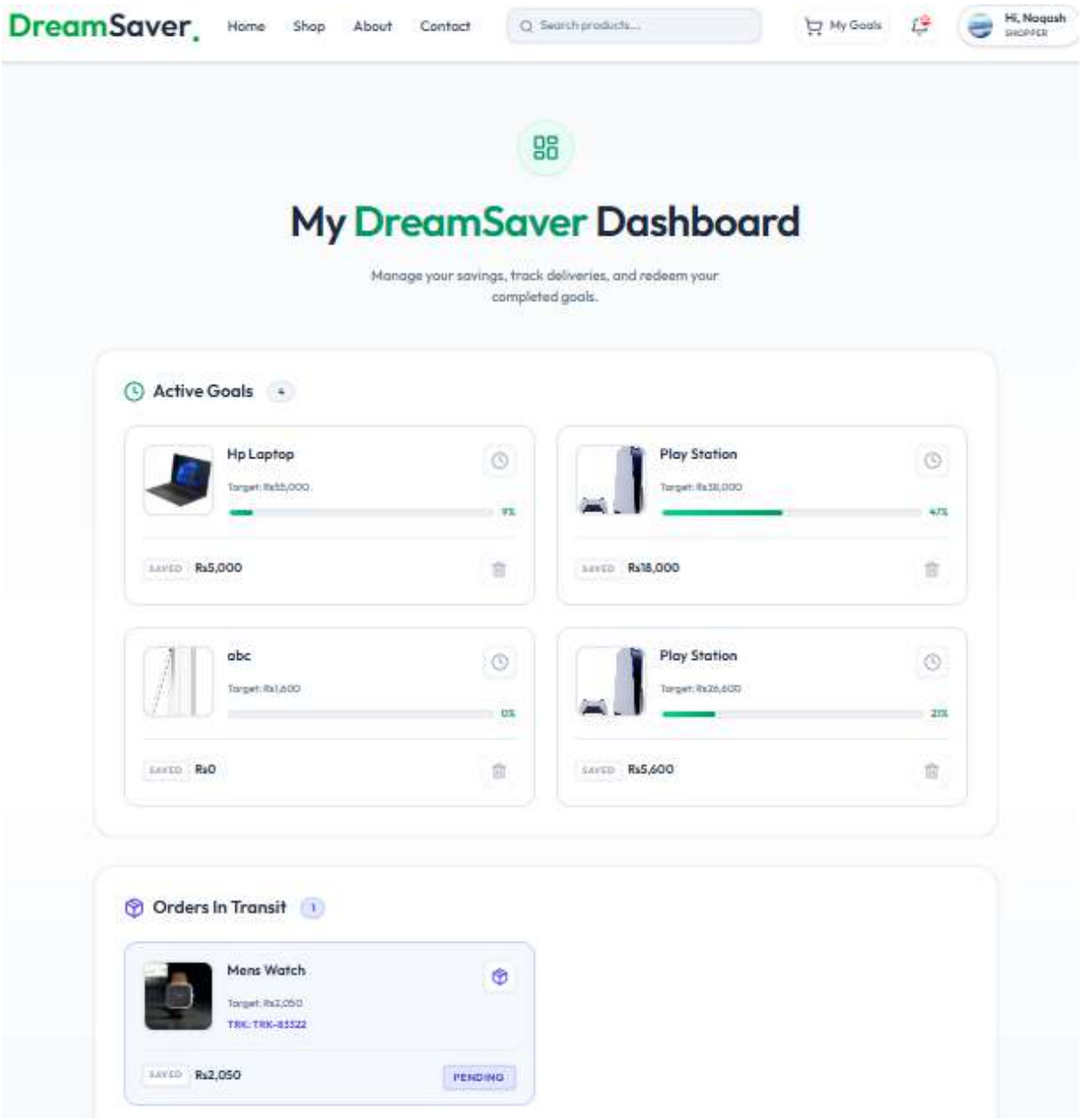


Figure 32: Goal Cart

4.4.21 Payment Initialization

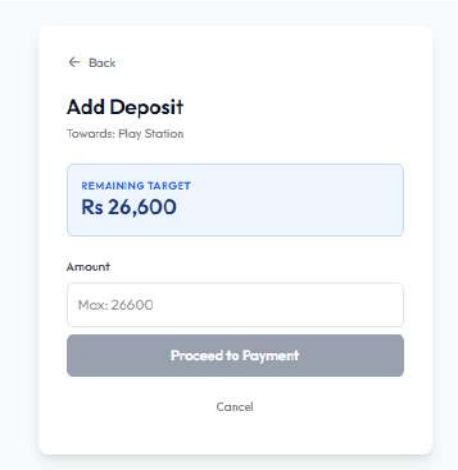


Figure 33: Payment Initialization

4.4.22 Stripe Gateway

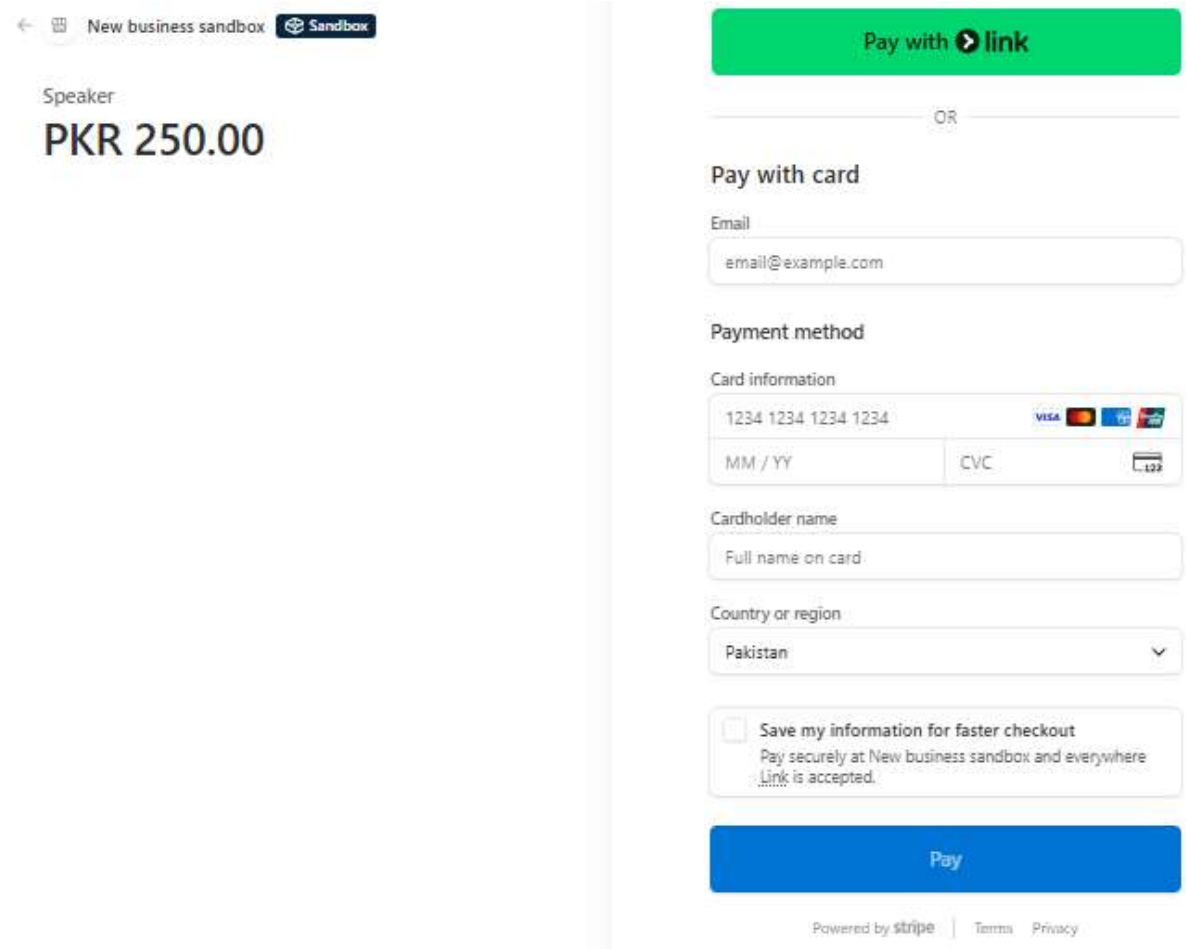


Figure 34: Stripe Gateway

4.4.23 Store Creation Form

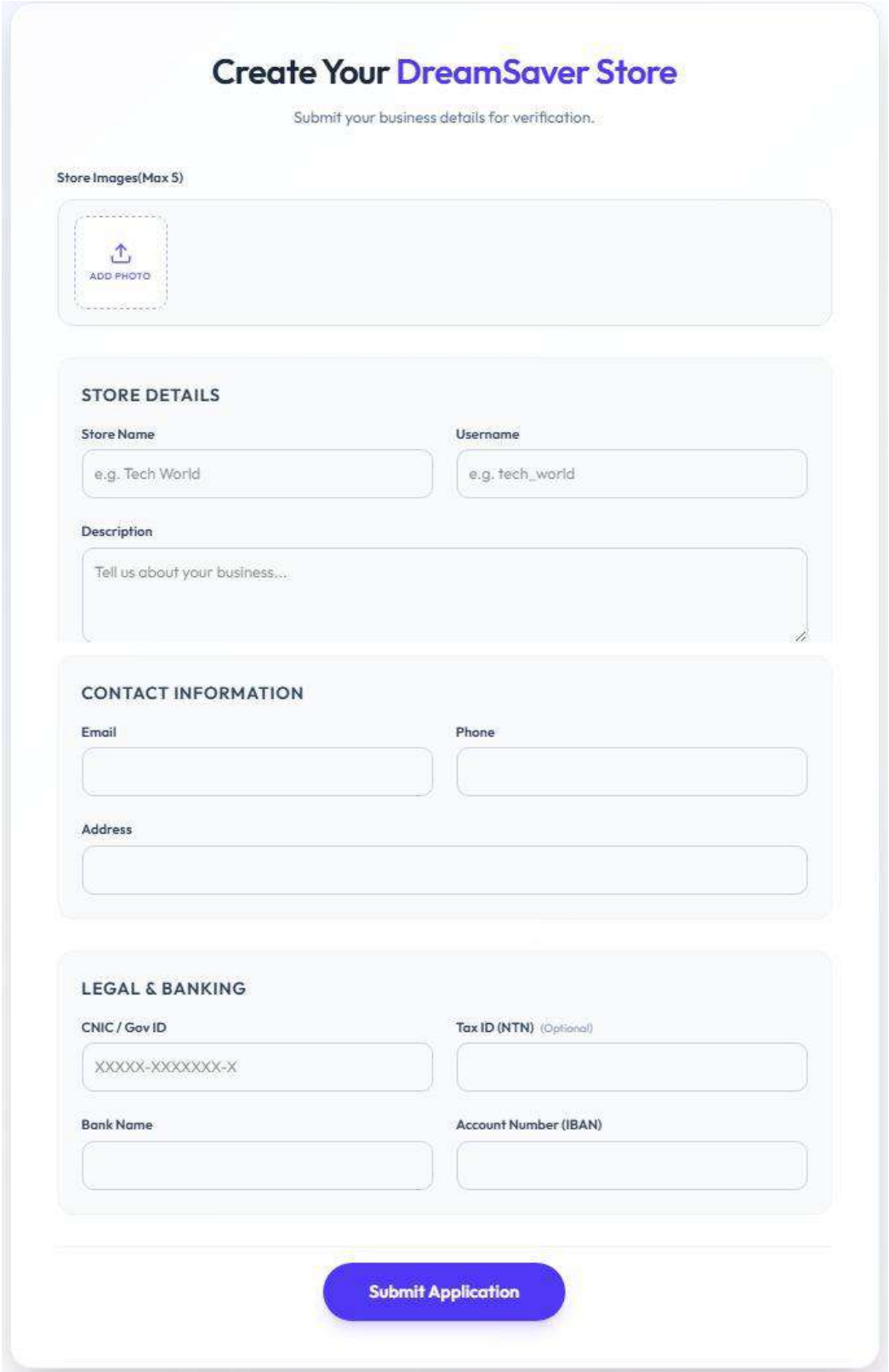


Figure 35: Store Creation Form

4.4.24 Wallet

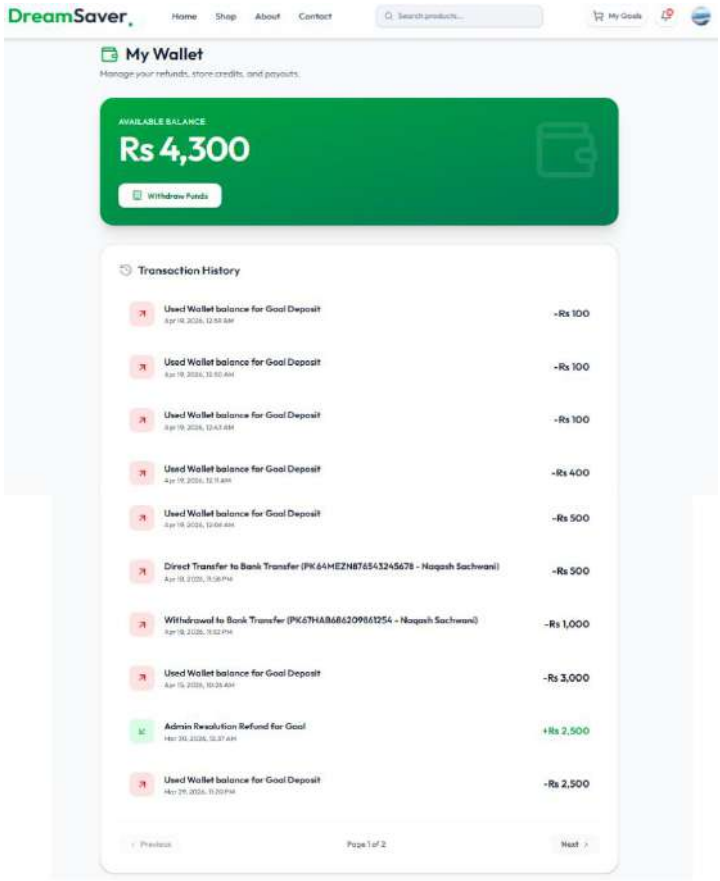


Figure 36: Wallet

4.4.25 Support & Complaints

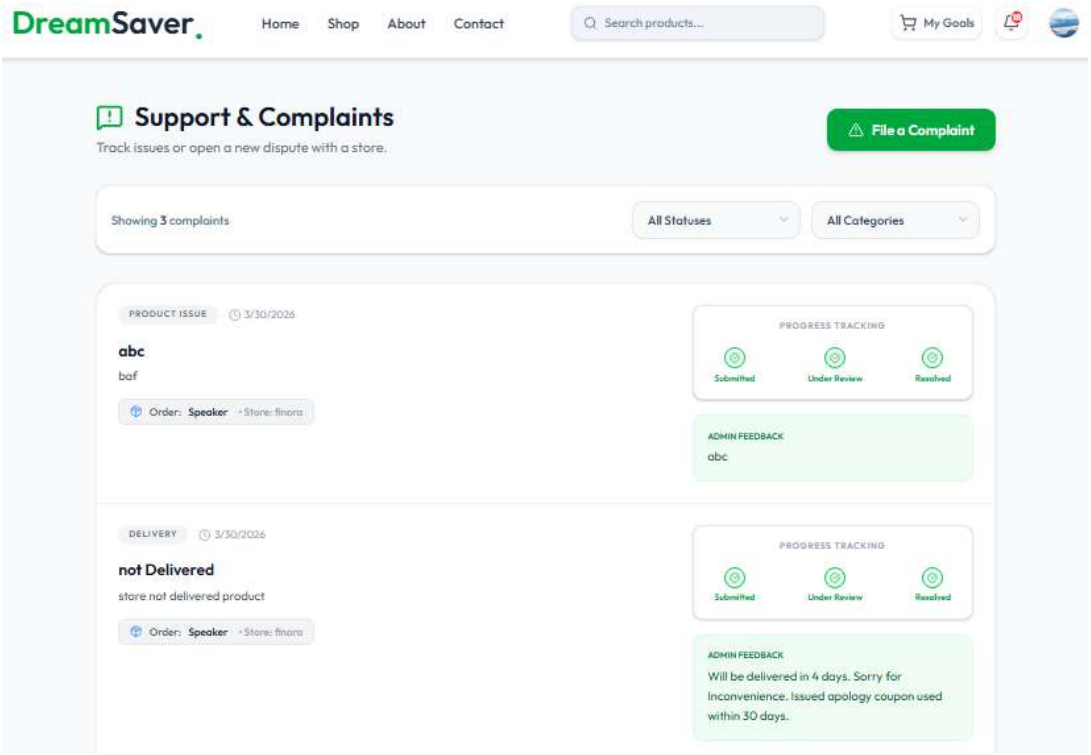


Figure 37: Support & Complaints

4.4.26 Goal History

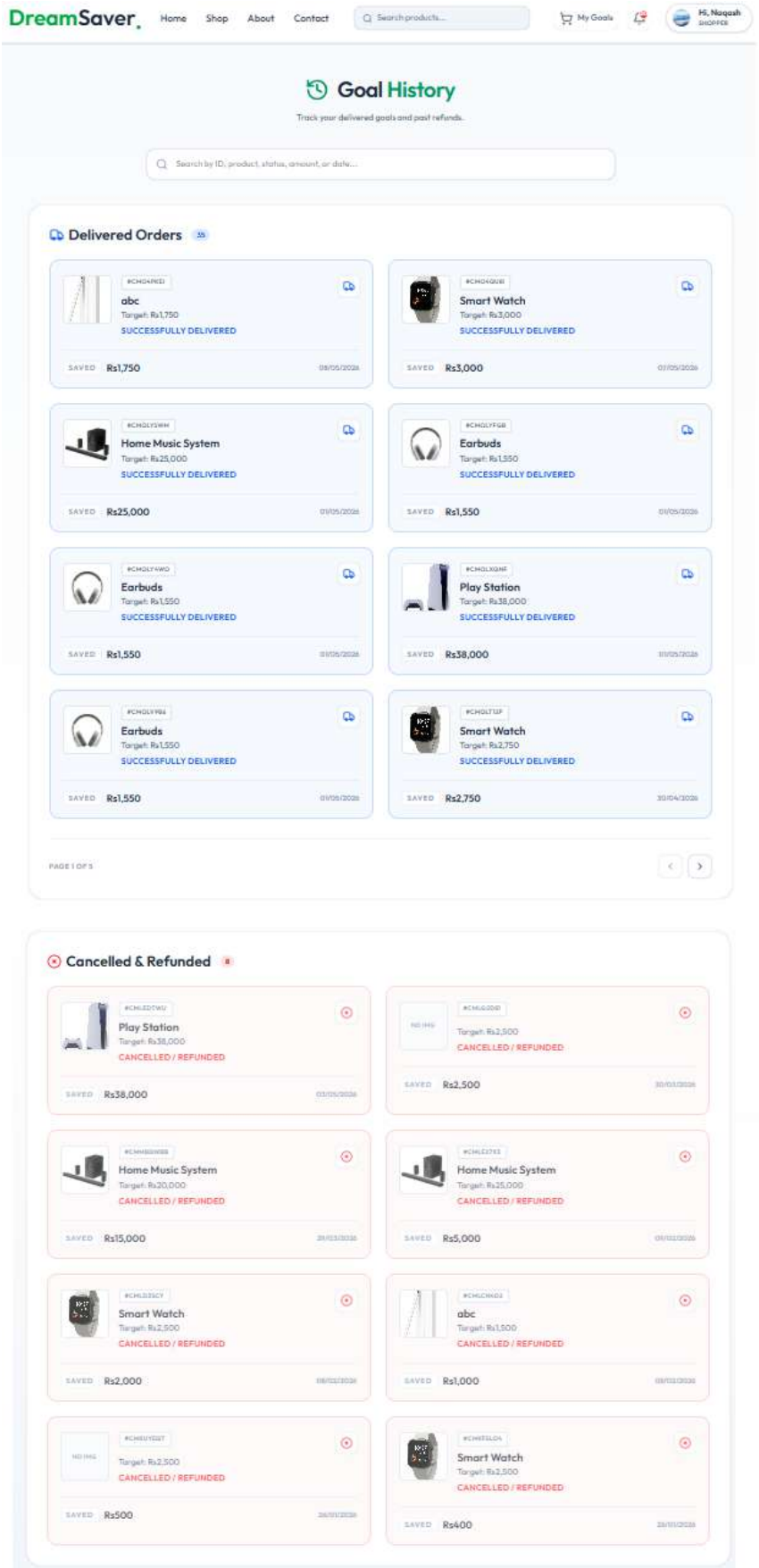


Figure 38: Goal History

4.4.27 My Coupons

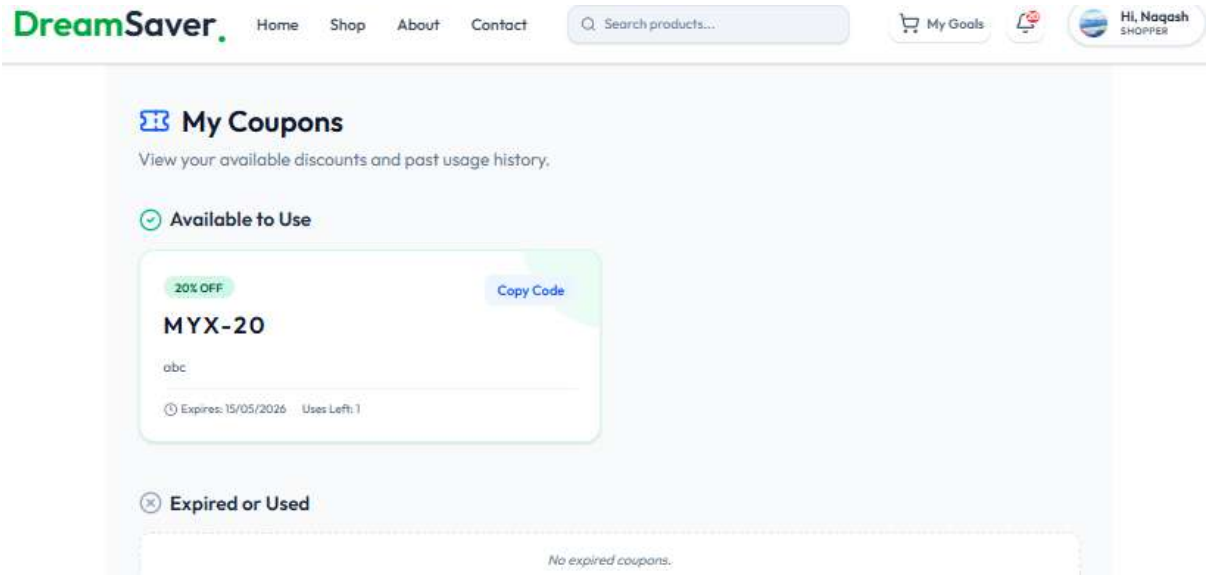


Figure 39: My Coupons

4.4.28 Map Tracking

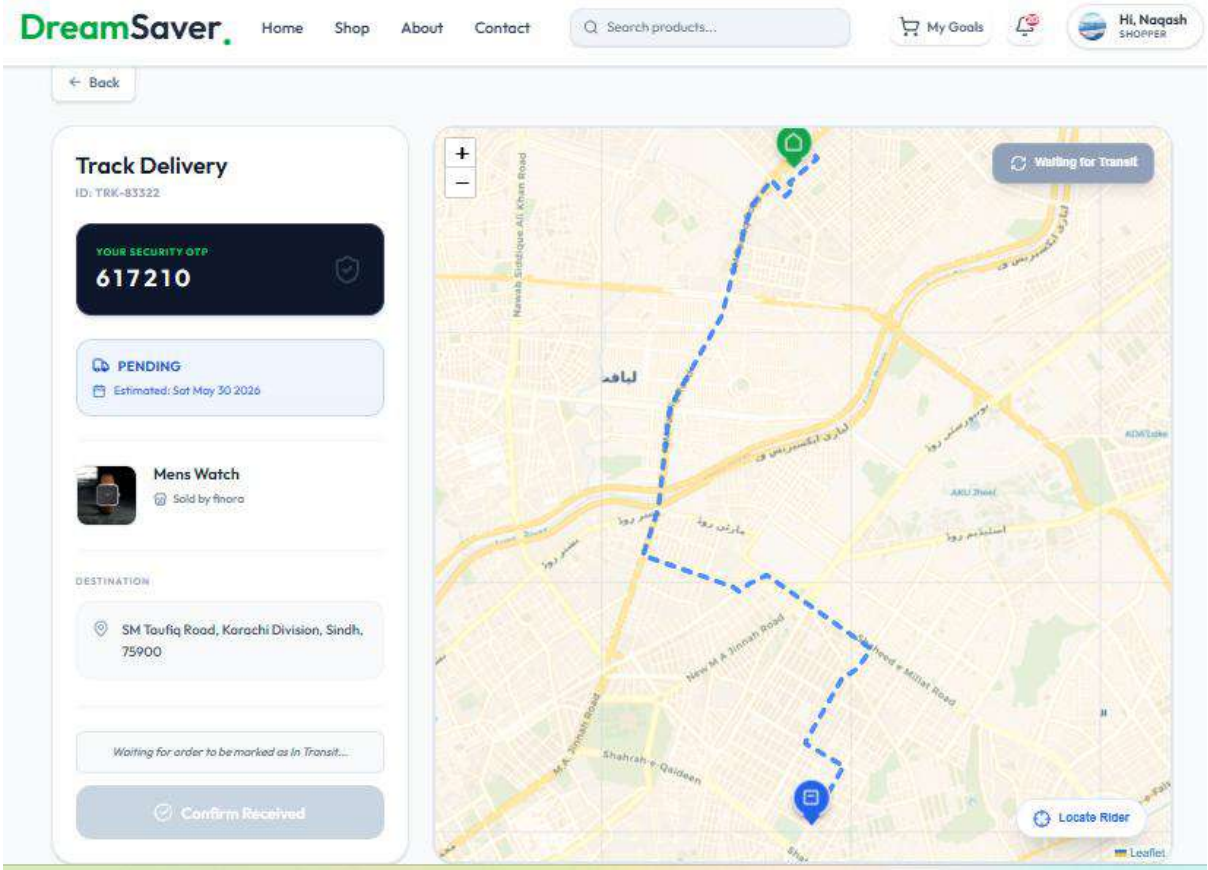


Figure 40: Map Tracking

4.4.29 User Management

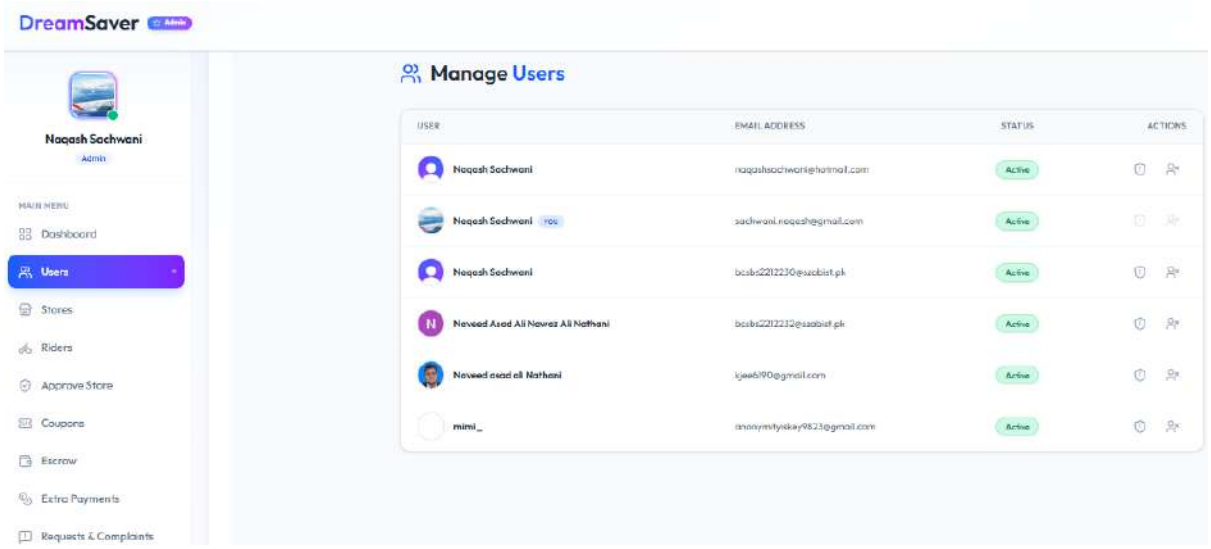


Figure 41: User Management

4.4.30 Escrow Management

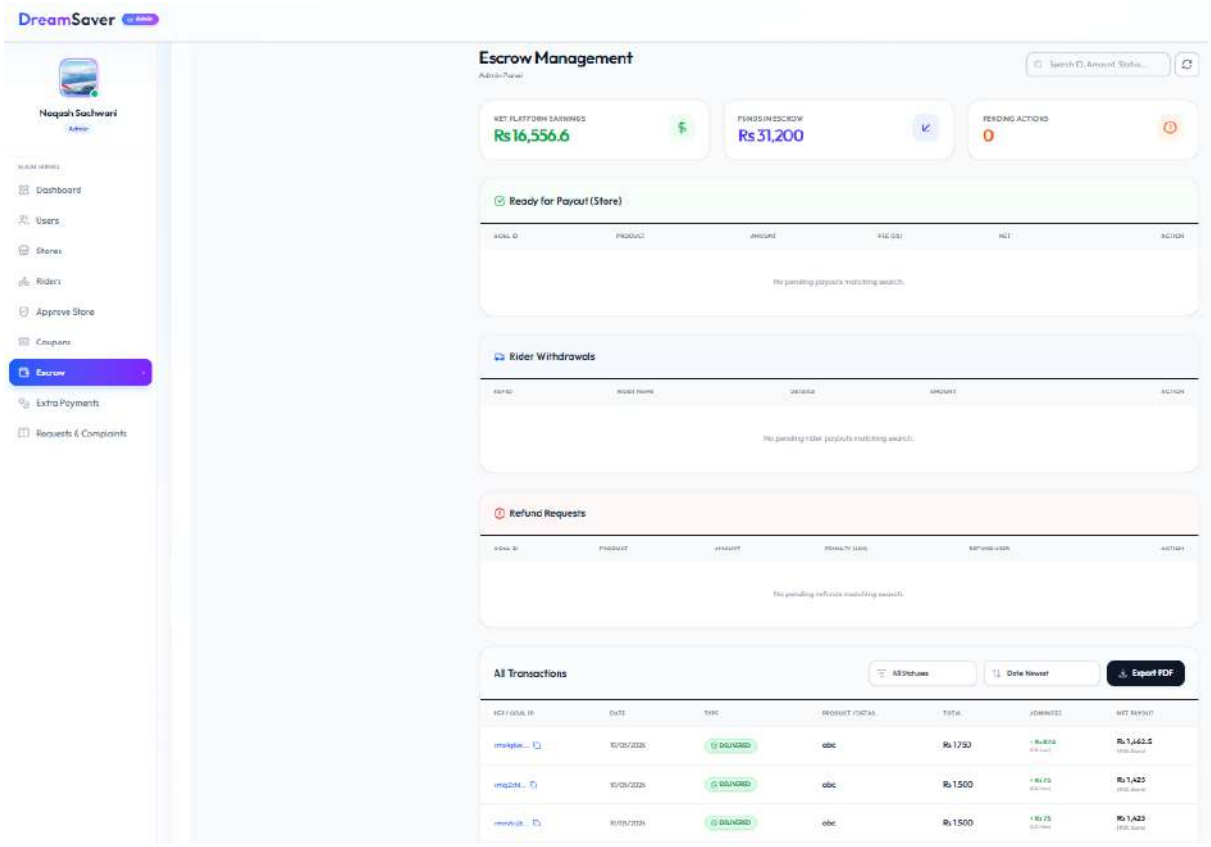


Figure 42: Escrow Management

4.4.31 Dispute Management

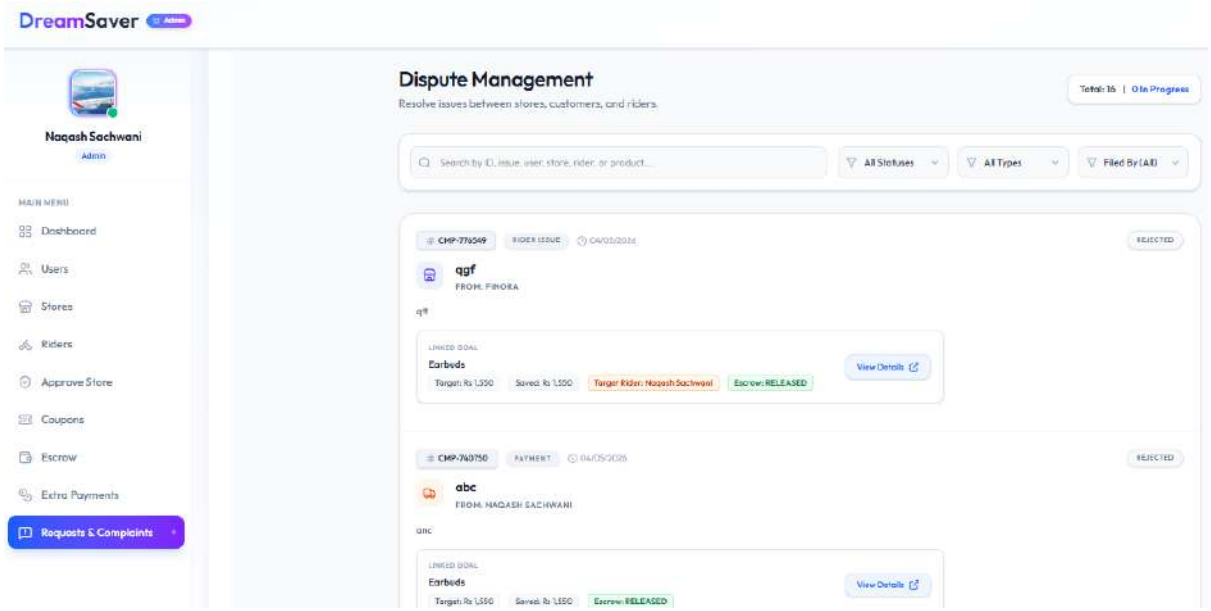


Figure 43: Dispute Management

4.4.32 Store Order Management

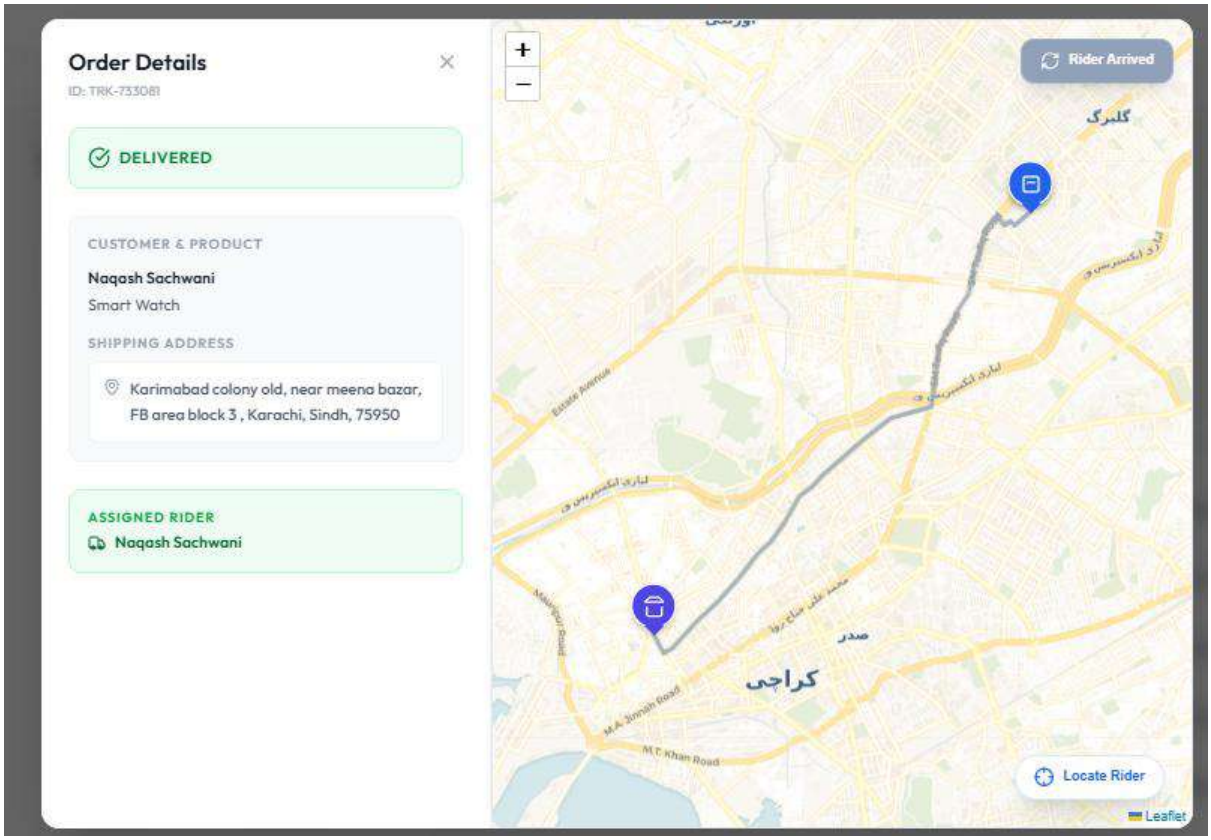


Figure 44: Store Order Management

4.4.33 Store Location Setting

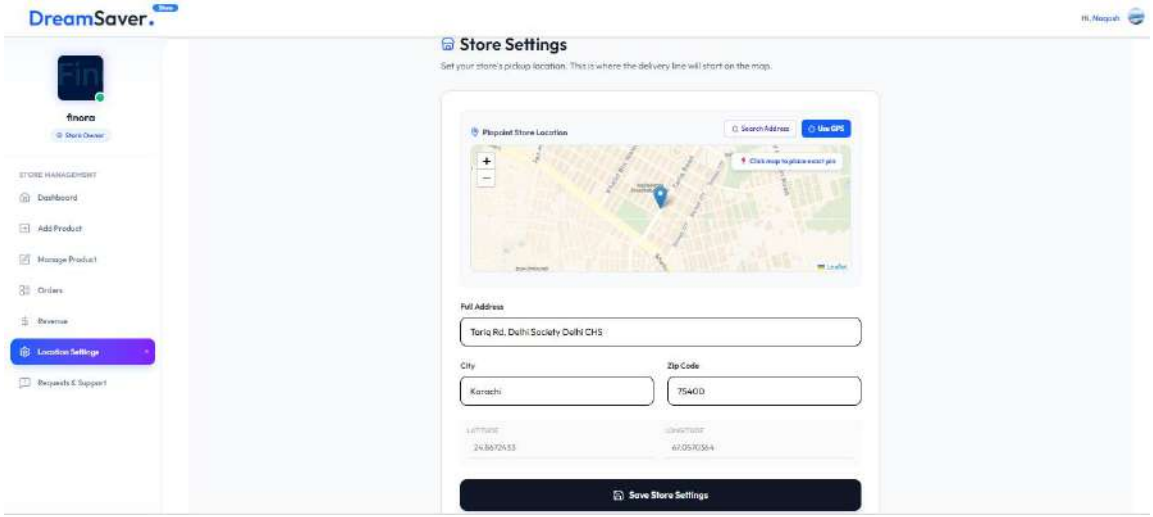


Figure 45: Store Location Setting

4.4.34 Store Requests & Support

Figure 46: Store Request & Support

4.4.35 Rider Registration

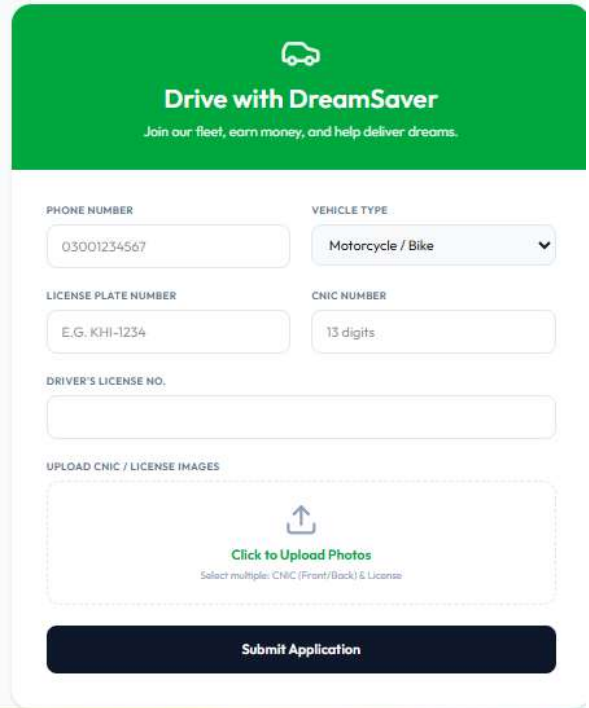


Figure 47: Rider Registration

4.4.36 Admin Rider Fleet Management

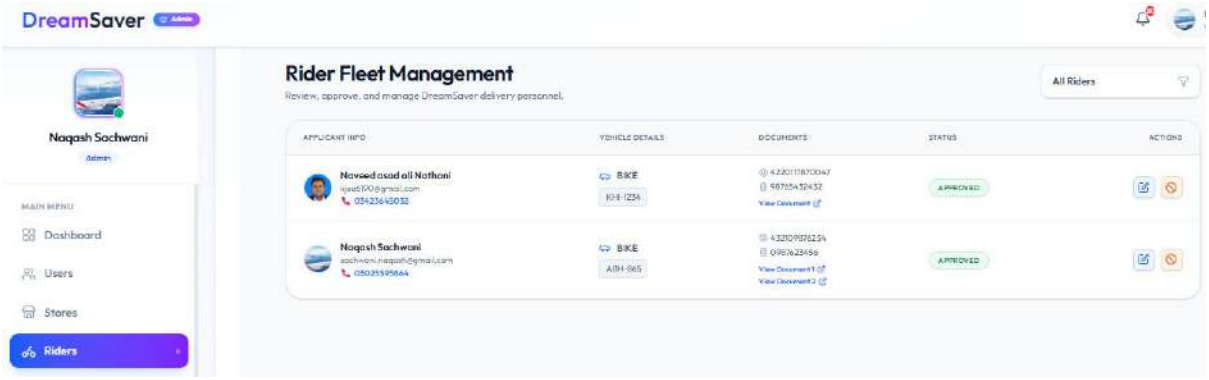


Figure 48: Admin Rider Fleet Management

4.4.37 Admin Extra Payments

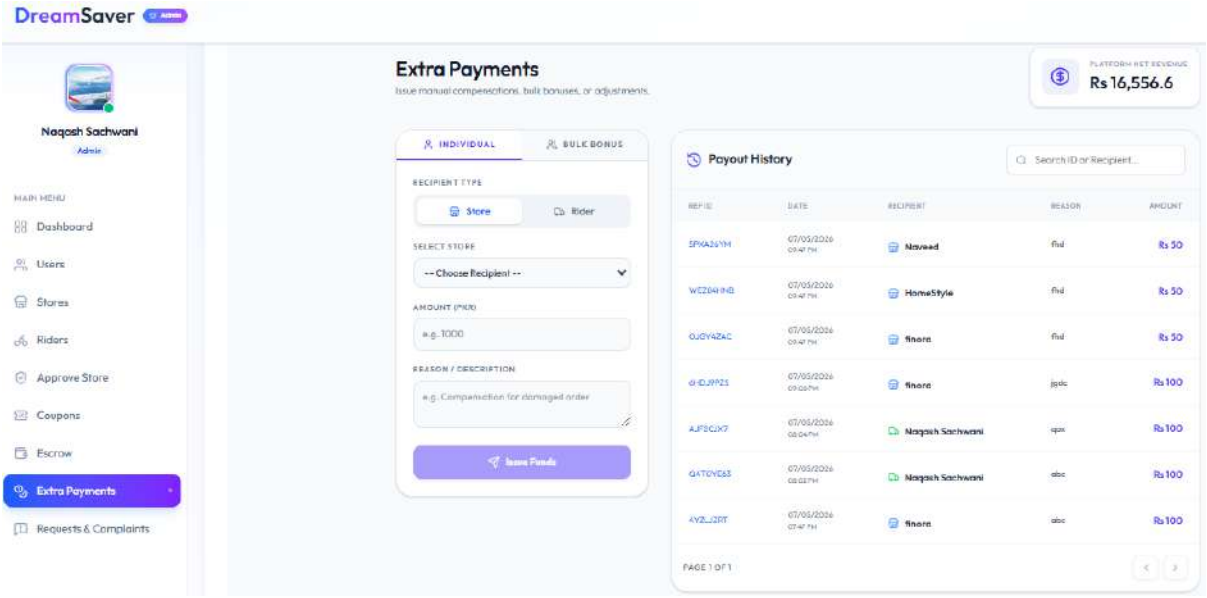


Figure 49: Admin Extra Payments

4.4.38 Rider Dashboard

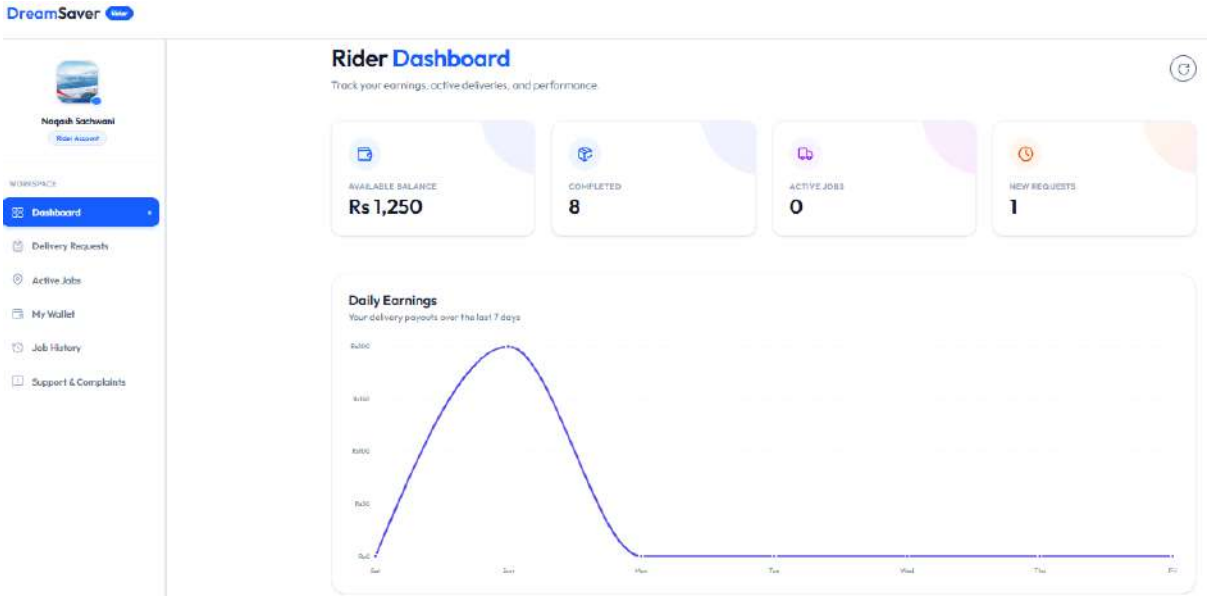


Figure 50: Rider Dashboard

4.4.39 Rider Delivery Requests

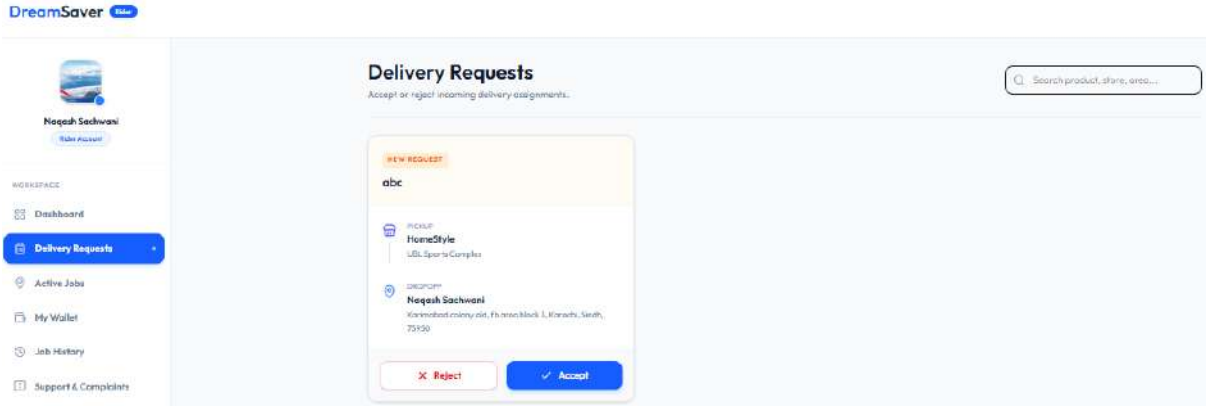


Figure 51: Rider Delivery Requests

4.4.40 Rider Active Jobs

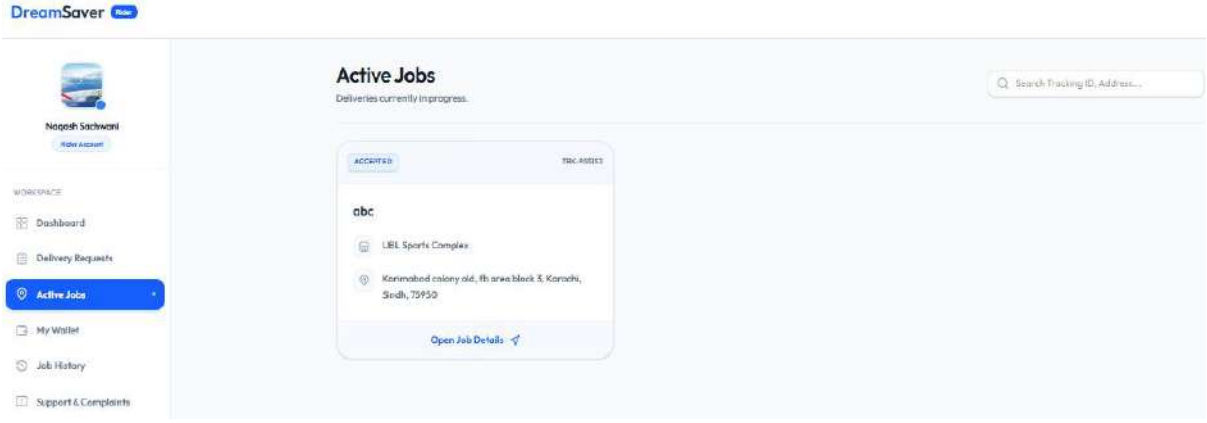


Figure 52: Rider Active Jobs

4.4.41 Rider Active Delivery

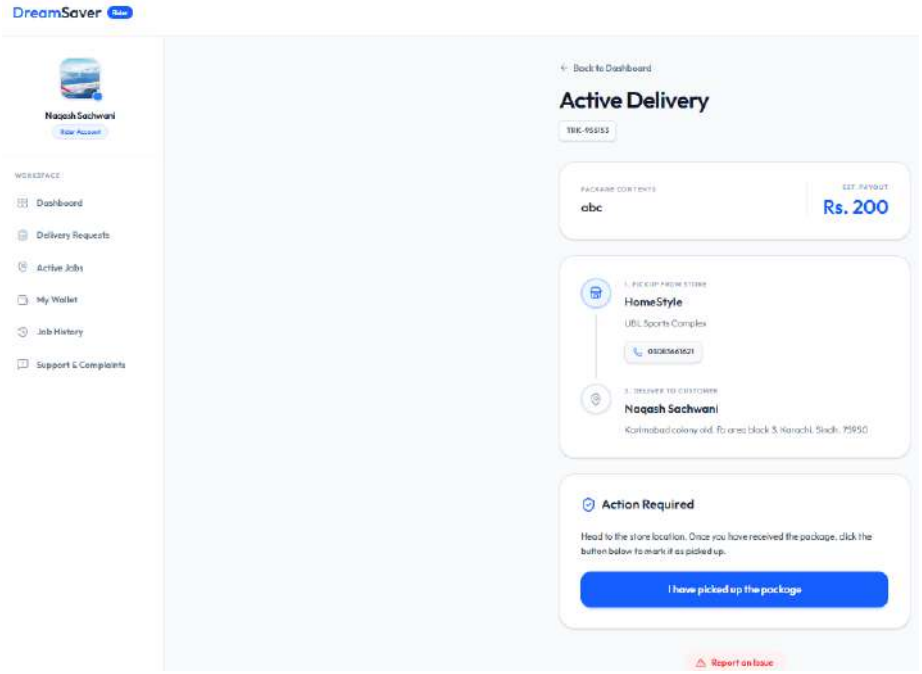


Figure 53: Rider Active Delivery

4.4.42 Rider Wallet

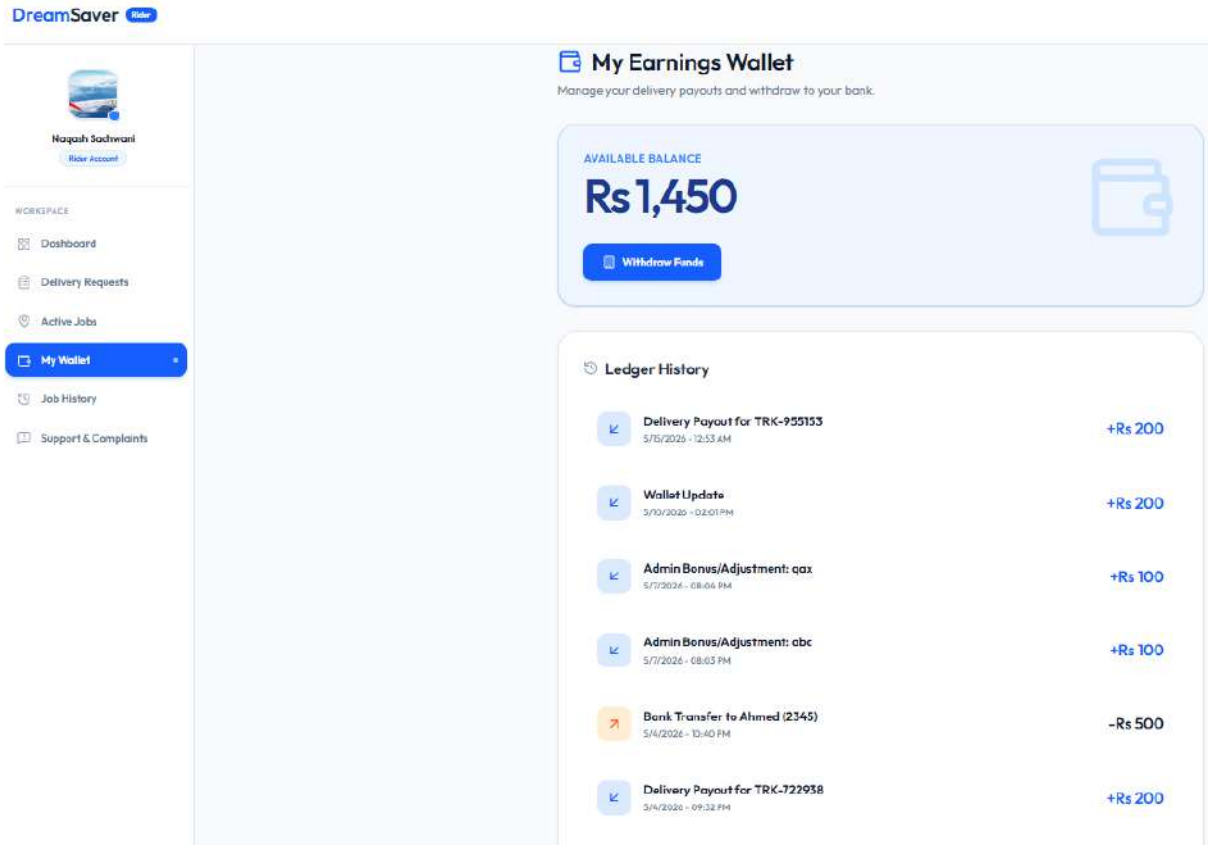


Figure 54: Rider Wallet

4.4.43 Rider Job History

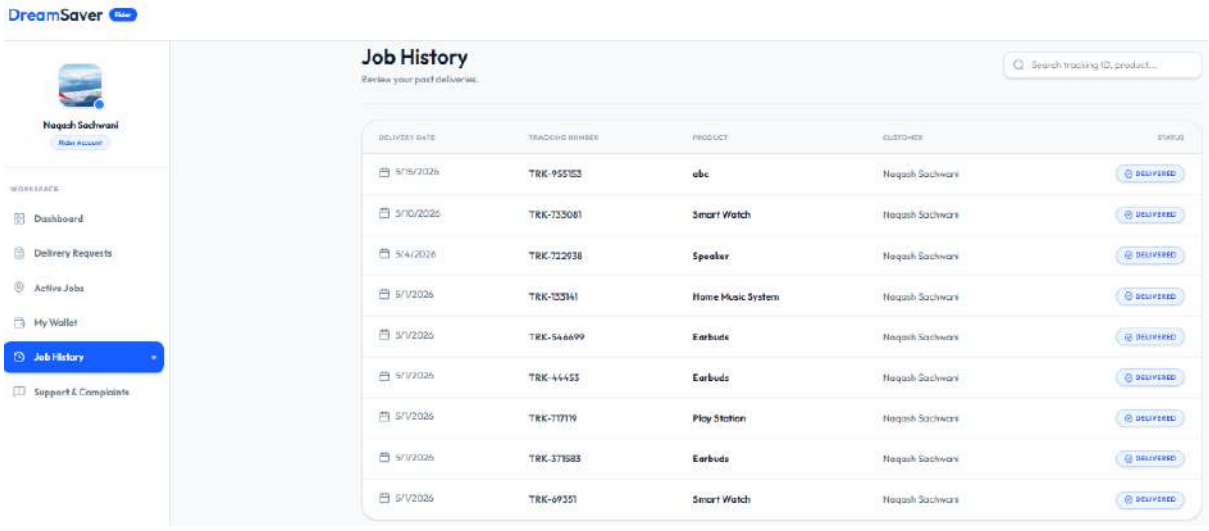


Figure 55: Rider Job History

4.4.44 Rider Support & Complaints

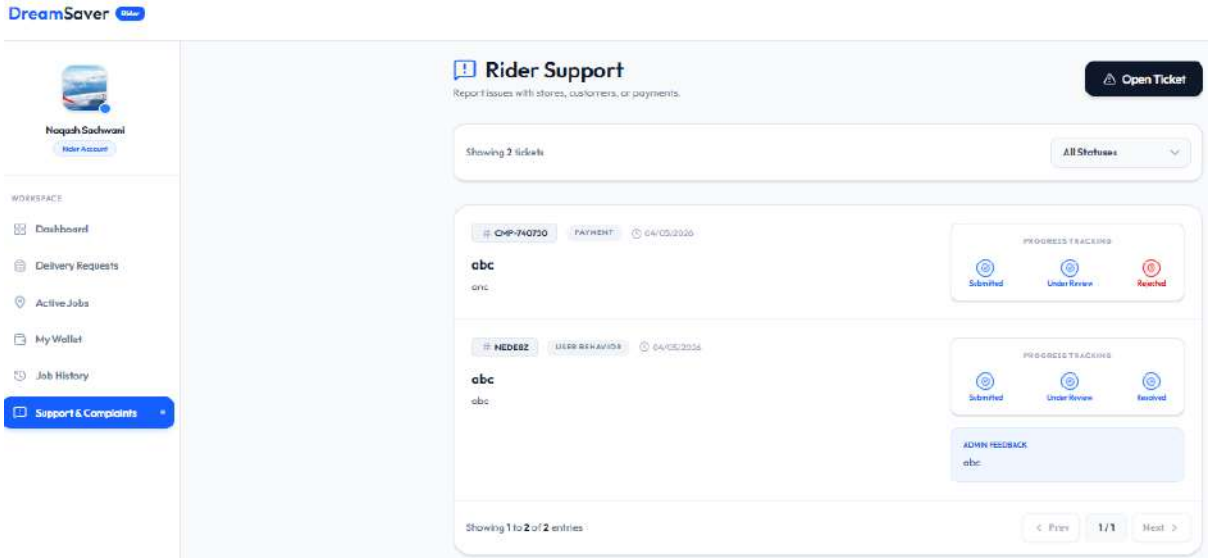


Figure 56: Rider Support & Complaints

4.4.45 Store Goal Progress

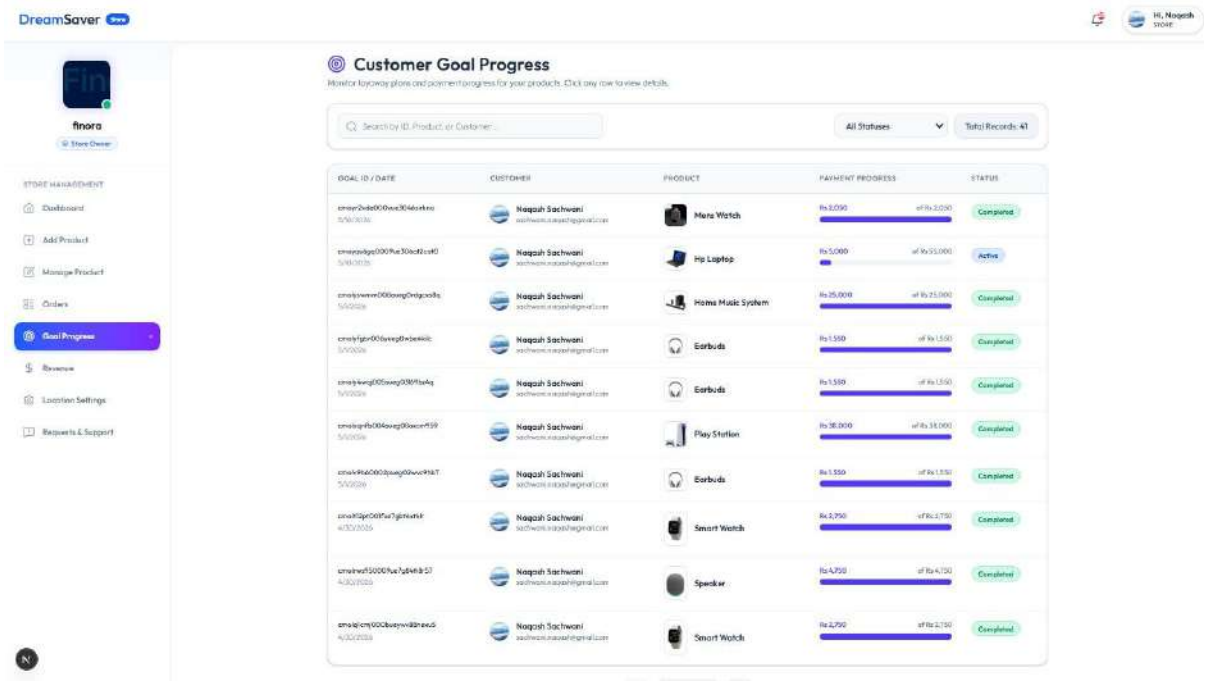


Figure 57: Store Goal Progress

4.4.46 Store Revenue

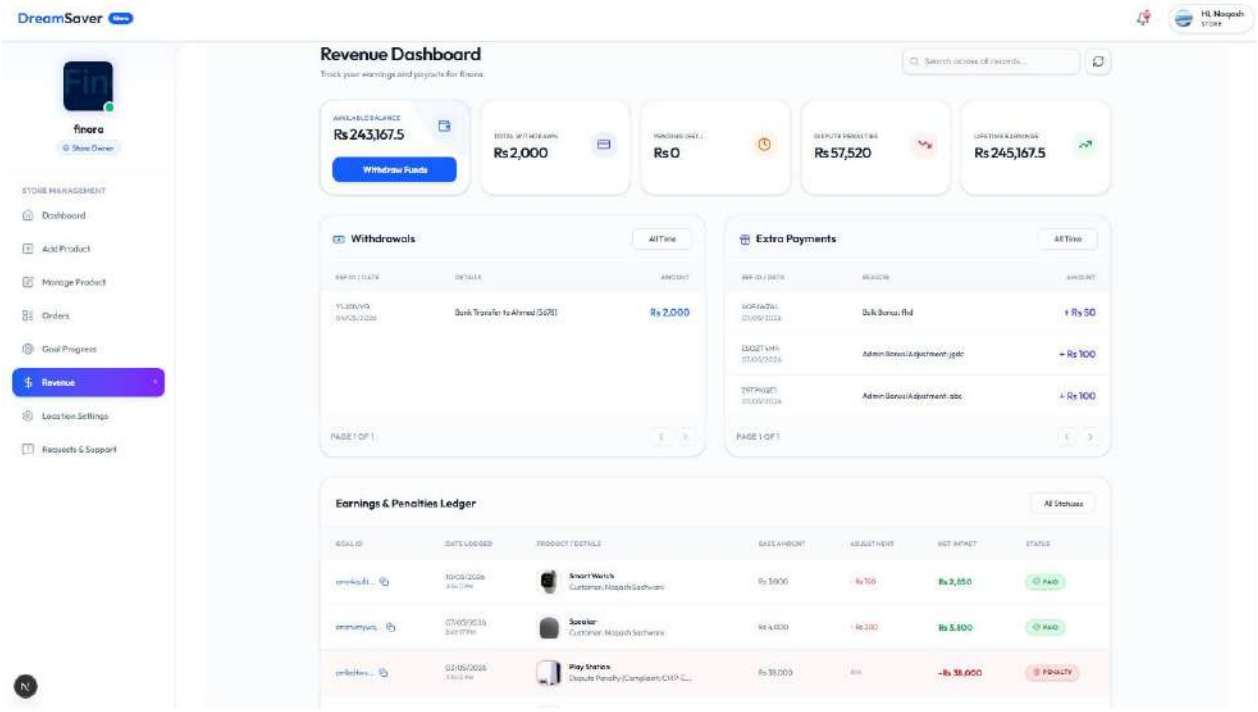


Figure 58: Store Revenue

4.4.47 Track Order

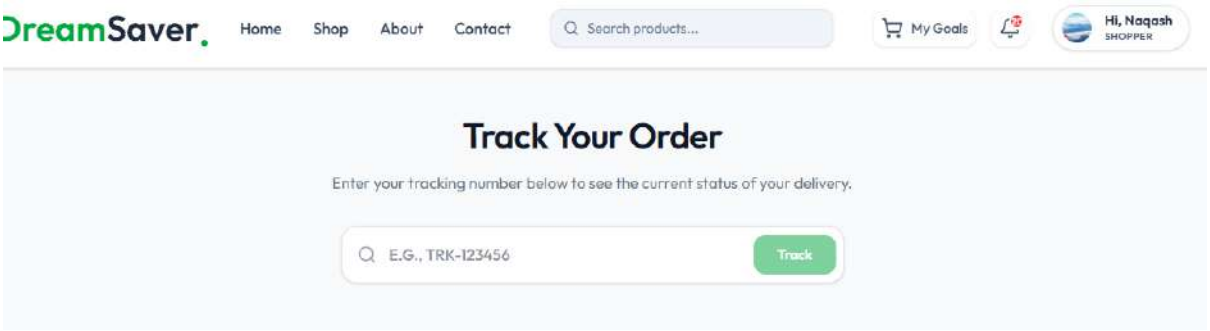


Figure 59: Track Order

4.4.48 Store Support Request

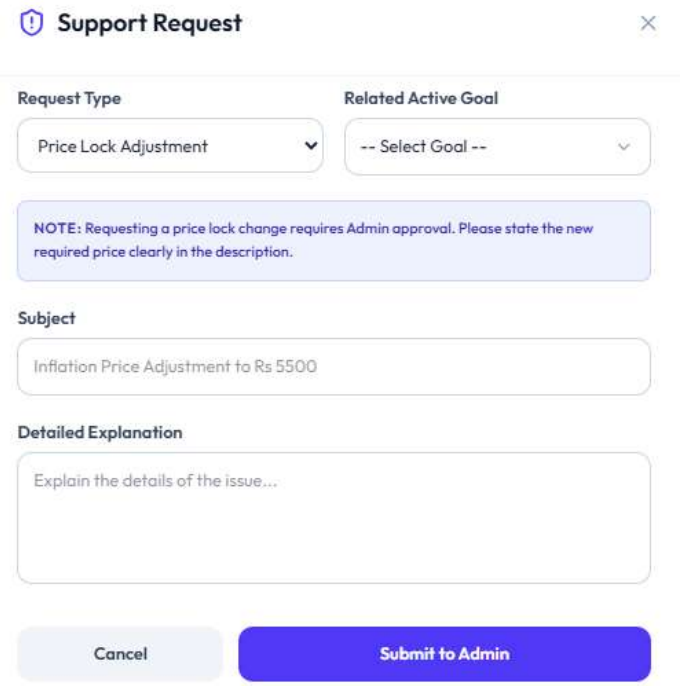


Figure 60: Store Support Request

5. Reuse and Relationship to Other Products

Reuse Sources

- Clerk Authentication (external)
- Payment Gateway
- Email Library
- Map (React leaflet)

Internal Reuse

- Shared React components (cards, forms, navbar)
- Reusable API utilities

Why Not Reusing Certain Tools

- BNPL systems: incompatible with non-credit model
- Complex ERP store systems: too heavy for FYP

6. Design Decisions & Tradeoffs

This section highlights the key decisions made during the design process and the tradeoffs considered:

Decision 1: Price Lock Policy

Balance between the risk in the stores and the trust of the customers. Selected to place at a goal begin.

Decision 2: Next.js Selected Over Node/Express

Reasons:

- Built-in routing
- Server-side rendering
- Better performance

Decision 3: Single Database vs Multi-DB
Single DB chosen for simplicity and atomic transactions.

Decision 4: Cloud Hosting
Chosen for reliability and easy deployment.
These choices are based on usability, performance, and resource usage.

7. Pseudocode For Components
7.1 User Registration (Signup)
```
FUNCTION RegisterUser(name, email, password)
  TRY
    // 1. Validate Input
    IF NOT IsValidEmail(email) OR PasswordLength < 8 THEN
      RETURN Error("Invalid input parameters")

    // 2. Check for existing user
    existingUser = Database.Users.FindByEmail(email)
    IF existingUser EXISTS THEN
      RETURN Error("Email is already registered")

    // 3. Secure Password
    hashedPassword = Hash(password, salt)

    // 4. Create User Record
    newUser = Database.Users.Create({
      name: name,
      email: email,
      password: hashedPassword,
      isVerified: FALSE
    })

    // 5. Send Verification
    verificationToken = GenerateToken(newUser.id)
    EmailService.SendVerificationEmail(newUser.email, verificationToken)

    RETURN Success("User created. Please check your email to verify.")
  CATCH Error
    RETURN Error("Registration failed")
END FUNCTION
```

7.2 User Login (Auth)

```
FUNCTION LoginUser(email, password)
  TRY
    // 1. Find User
    user = Database.Users.FindByEmail(email)
    IF NOT user EXISTS THEN
      RETURN Error("Invalid credentials")

    // 2. Verify Password
    isMatch = CompareHash(password, user.hashedPassword)
    IF NOT isMatch THEN
      RETURN Error("Invalid credentials")

    // 3. Check Verification Status
    IF NOT user.isVerified THEN
      RETURN Error("Please verify your email first")

    // 4. Check Ban/Suspension
    IF NOT user.isActive THEN
      RETURN Error("Account is suspended")

    // 5. Generate Auth Token
    sessionToken = GenerateJWT(user.id, user.role)

    RETURN Success(sessionToken, user.profile)
  CATCH Error
    RETURN Error("Login failed")
END FUNCTION
```

7.3 Create Savings Goal

```
FUNCTION CreateSavingsGoal(userId, productId, targetAmount, targetDate, couponCode)
  TRY
    // 1. Validate Product
    product = Database.Products.FindById(productId)
    IF NOT product EXISTS OR product.inStock == FALSE THEN
      RETURN Error("Product unavailable")

    // 2. Apply Coupon Logic (if any)
    finalTarget = product.price
    IF couponCode IS VALID THEN
       finalTarget = ApplyDiscount(finalTarget, couponCode)

    // 3. Create Goal
```

```
        newGoal = Database.Goals.Create({
            userId: userId,
            productId: productId,
            targetAmount: finalTarget,
            saved: 0,
            endDate: targetDate,
            status: "ACTIVE"
        })

        // 4. Lock Product Price
        Call LockProductForLayaway(userId, productId, newGoal.id)

        // 5. Notify User
        Call SendNotification(userId, "Goal Created", "You started saving for " + product.name)

        RETURN Success(newGoal)
    CATCH Error
        Database.RollbackTransaction()
        RETURN Error("Failed to create goal")
END FUNCTION
```

7.4 Deposit to Savings Goal (Payments)

```
FUNCTION ProcessDeposit(userId, goalId, amount, paymentMethod)
    TRY
        // 1. Fetch Goal
        goal = Database.Goals.FindById(goalId)
        IF goal.status == "COMPLETED" OR goal.status == "CANCELLED" THEN
            RETURN Error("Cannot deposit to this goal")

        // 2. Validate Amount
        remainingAmount = goal.targetAmount - goal.saved
        IF amount > remainingAmount THEN
            RETURN Error("Deposit exceeds remaining target")

        // 3. Process Payment via External Gateway or Wallet
        paymentResult = PaymentGateway.Charge(userId, amount, paymentMethod)
        IF paymentResult.status != "SUCCESS" THEN
            RETURN Error("Payment failed")

        // 4. Record Deposit & Escrow
        Database.Deposits.Create(goalId, userId, amount)
        Database.Escrow.Upsert(goalId, amount, status="HELD")

        // 5. Update Goal Progress
```

```
    goal.saved = goal.saved + amount
    IF goal.saved >= goal.targetAmount THEN
      goal.status = "COMPLETED"
      Call SendNotification(userId, "Goal Completed!", "Ready to redeem!")
    ELSE
      Call SendNotification(userId, "Deposit Successful", "Saved so far: " + goal.saved)

    Database.Goals.Update(goal)

    RETURN Success("Deposit successful")
  CATCH Error
    RETURN Error("Failed to process deposit")
END FUNCTION
```

7.5 Request Goal Cancellation & Refund

```
FUNCTION CancelGoal(userId, goalId)
  TRY
    // 1. Fetch Goal
    goal = Database.Goals.FindById(goalId)
    IF goal.userId != userId THEN RETURN Error("Unauthorized")

    // 2. Handle Zero-balance cancellations
    IF goal.saved == 0 THEN
      Database.Goals.Delete(goalId)
      Database.PriceLocks.Delete(goalId)
      RETURN Success("Goal deleted")

    // 3. Handle Funded cancellations (Refund)
    Database.RefundRequests.Create({
      userId: userId,
      goalId: goalId,
      amount: goal.saved,
      status: "PENDING_ADMIN_REVIEW"
    })

    // 4. Update Statuses
    goal.status = "CANCELLED"
    Database.Goals.Update(goal)
    Database.Escrow.Update(goalId, status="HELD") // Wait for admin action
    Database.PriceLocks.Release(goalId)

    // 5. Notify Admin and User
    Call SendNotification(userId, "Cancellation Initiated", "Refund processing")
```

```
    RETURN Success("Refund requested successfully")
  CATCH Error
    RETURN Error("Cancellation failed")
END FUNCTION
```

7.6 Redeem Product After Goal Completion

```
FUNCTION RedeemProduct(userId, goalId, shippingAddressId)
  TRY
    // 1. Fetch & Verify Goal
    goal = Database.Goals.FindById(goalId)
    IF goal.status != "COMPLETED" THEN
      RETURN Error("Goal is not fully funded yet")

    // 2. Verify Address
    address = Database.Addresses.FindById(shippingAddressId)
    IF NOT address EXISTS THEN RETURN Error("Invalid address")

    // 3. Create Delivery Record
    newDelivery = Database.Deliveries.Create({
      goalId: goalId,
      status: "PENDING_DISPATCH",
      address: address.fullText,
      trackingNumber: GenerateTracking()
    })

    // 4. Notify Store Owner
    storeId = goal.product.storeId
    storeOwner = Database.Stores.GetOwner(storeId)
    Call SendNotification(storeOwner.id, "New Order to Fulfill", "A customer has redeemed
their layaway item.")

    RETURN Success(newDelivery.trackingNumber)
  CATCH Error
    RETURN Error("Failed to redeem product")
END FUNCTION
```

7.7 Store/Store Registration & Approval

```
FUNCTION RegisterStore(userId, storeData, documents)
  TRY
    // 1. Validate User
    IF Database.Stores.ExistsForUser(userId) THEN
      RETURN Error("User already has a store application")
```

```
        // 2. Upload Documents
        documentUrls = StorageService.Upload(documents)

        // 3. Create Store Application
        Database.Stores.Create({
            userId: userId,
            name: storeData.name,
            details: storeData.details,
            status: "PENDING_APPROVAL",
            documents: documentUrls
        })

        // 4. Notify Admin
        Call SendNotification(ADMIN_GROUP, "New Store Application", storeData.name + "
applied.")

        RETURN Success("Application submitted for review")
    CATCH Error
        RETURN Error("Store registration failed")
END FUNCTION

// Admin Action
FUNCTION ApproveStore(adminId, storeId, decision)
    store = Database.Stores.FindById(storeId)
    IF decision == "APPROVE" THEN
        store.status = "ACTIVE"
        Call SendNotification(store.userId, "Store Approved!", "You can now list products.")
    ELSE
        store.status = "REJECTED"
    Database.Stores.Update(store)
END FUNCTION
```

7.8 Product Listing by Store

```
FUNCTION CreateProduct(userId, productData, images)
    TRY
        // 1. Verify Store Status
        store = Database.Stores.FindByUserId(userId)
        IF store.status != "ACTIVE" THEN
            RETURN Error("Store is not active")

        // 2. Process Images
        imageUrls = StorageService.UploadMultiple(images)

        // 3. Insert Product
```

```
      newProduct = Database.Products.Create({
        storeId: store.id,
        name: productData.name,
        price: productData.price,
        stock: productData.stock,
        images: imageUrls,
        inStock: productData.stock > 0
      })

      RETURN Success(newProduct)
    CATCH Error
      RETURN Error("Failed to list product")
END FUNCTION
```

7.9 Lock Product for Layaway

```
FUNCTION LockProductForLayaway(userId, productId, goalId)
    TRY
      // 1. Fetch Product
      product = Database.Products.FindById(productId)

      // 2. Create Price Lock
      Database.PriceLocks.Create({
        goalId: goalId,
        productId: productId,
        lockedPrice: product.price,
        lockedAt: CurrentTime()
      })

      // 3. Reserve Stock (Optional depending on business rules)
      // If system reserves physical inventory upon goal creation:
      product.stock = product.stock - 1
      IF product.stock == 0 THEN product.inStock = FALSE
      Database.Products.Update(product)

      RETURN Success("Product and price locked")
    CATCH Error
      RETURN Error("Failed to lock product")
END FUNCTION
```

7.10 Confirm Delivery & Final Payment Release

```
FUNCTION ConfirmDeliveryAndReleaseFunds(deliveryId)
    TRY
      // 1. Update Delivery Status
```

```
    delivery = Database.Deliveries.FindById(deliveryId)
    delivery.status = "DELIVERED"
    delivery.deliveryDate = CurrentTime()
    Database.Deliveries.Update(delivery)

    // 2. Fetch Associated Escrow
    escrow = Database.Escrow.FindByGoalId(delivery.goalId)

    // 3. Calculate Fees
    platformFee = escrow.amount * 0.05 // 5% Admin cut
    storeNet = escrow.amount - platformFee

    // 4. Update Escrow & Release Funds
    escrow.status = "RELEASED"
    escrow.platformFee = platformFee
    escrow.netAmount = storeNet
    Database.Escrow.Update(escrow)

    // 5. Credit Store Wallet
    storeOwnerId = delivery.goal.product.store.userId
    Database.Wallets.Credit(storeOwnerId, storeNet)

    // 6. Notify Parties
    Call SendNotification(delivery.goal.userId, "Item Delivered", "Enjoy your product!")
    Call SendNotification(storeOwnerId, "Funds Released", "Rs " + storeNet + " added to
wallet.")

    RETURN Success("Delivery confirmed and funds released")
  CATCH Error
    RETURN Error("Failed to release funds")
END FUNCTION
```

7.11 Notification

```
FUNCTION SendNotification(userId, title, message, type)
  TRY
    // 1. Save to Database (for in-app bell icon)
    Database.Notifications.Create({
      userId: userId,
      title: title,
      message: message,
      type: type,
      isRead: FALSE,
      createdAt: CurrentTime()
    })
```

```
    // 2. Fetch User Email/Settings
    user = Database.Users.FindById(userId)

    // 3. Send Email (Async)
    IF user.emailNotificationsEnabled THEN
      EmailService.Send(user.email, title, GenerateHTMLTemplate(message))

    // 4. Send Push Notification (Async - if mobile)
    IF user.pushToken EXISTS THEN
      PushService.Send(user.pushToken, title, message)

    RETURN Success()
  CATCH Error
    Log.Error("Notification failure: " + Error.message)
END FUNCTION
```

7.12 Delivery Tracking

```
FUNCTION UpdateTracking(deliveryId, latitude, longitude, locationString)
  TRY
    // 1. Validate Delivery
    delivery = Database.Deliveries.FindById(deliveryId)
    IF NOT delivery EXISTS THEN RETURN Error("Delivery not found")

    // 2. Create Tracking Node
    Database.DeliveryTrackings.Create({
      deliveryId: deliveryId,
      lat: latitude,
      lng: longitude,
      location: locationString,
      recordedAt: CurrentTime()
    })

    // 3. Update Delivery Master coordinates
    delivery.currentLat = latitude
    delivery.currentLng = longitude
    Database.Deliveries.Update(delivery)

    RETURN Success("Tracking updated")
  CATCH Error
    RETURN Error("Failed to update tracking")
END FUNCTION
```

7.13 Generate Reports & Logs

```
FUNCTION GeneratePDFReport(adminId, reportType, dateRange)
  TRY
    // 1. Authenticate Admin
    IF NOT IsAdmin(adminId) THEN RETURN Error("Unauthorized")

    // 2. Fetch Data based on Type
    data = NULL
    SWITCH reportType
      CASE "SALES": data = Database.Escrow.FindReleased(dateRange)
      CASE "USERS": data = Database.Users.FindNew(dateRange)
      CASE "DISPUTES": data = Database.Complaints.FindAll(dateRange)

    // 3. Generate HTML Template
    htmlContent = HTMLService.Render(reportType + "_TEMPLATE", data)

    // 4. Convert HTML to PDF
    pdfFile = PDFGeneratorLibrary.ConvertHTMLToPDF(htmlContent)

    // 5. Return File Stream
    RETURN FileStream(pdfFile, "application/pdf")
  CATCH Error
    RETURN Error("Report generation failed")
END FUNCTION
```

7.14 Dispute Resolution (Admin)

```
FUNCTION ResolveDispute(adminId, complaintId, action, adminNotes)
  TRY
    // 1. Verify Admin & Fetch Complaint
    IF NOT IsAdmin(adminId) THEN RETURN Error("Unauthorized")
    complaint = Database.Complaints.FindById(complaintId)
    goalId = complaint.goalId

    // 2. Apply Resolution Logic
    IF action == "REFUND_USER" THEN
      // Calculate penalty (e.g., store loses sale, 20% penalty applied)
      Call ProcessRefundLogic(goalId, penalty=TRUE)
    ELSE IF action == "RELEASE_FUNDS_TO_STORE" THEN
      // Force release funds ignoring user complaint
      Call ConfirmDeliveryAndReleaseFunds(complaint.deliveryId)
    ELSE IF action == "ISSUE_COUPON" THEN
      // Generate apology coupon for user
      Database.Coupons.CreateApologyCoupon(complaint.userId)

    // 3. Close Complaint
    complaint.status = "RESOLVED"
```

```
        complaint.adminNotes = adminNotes
        Database.Complaints.Update(complaint)

        // 4. Notify both Filer and Target
        Call SendNotification(complaint.filerId, "Dispute Resolved", adminNotes)
        IF complaint.targetId EXISTS THEN
            Call SendNotification(complaint.targetId, "Dispute Resolved", adminNotes)

        RETURN Success("Dispute resolved successfully")
    CATCH Error
        Database.RollbackTransaction()
        RETURN Error("Failed to resolve dispute")
END FUNCTION
```

7.15 Rider Earnings & Payouts (Withdrawal)

```
FUNCTION processRiderWithdrawal(riderId, requestedAmount):

    // 1. Fetch Rider Profile
    rider = DATABASE.RiderProfile.findById(riderId)

    // 2. Validate Available Balance
    IF rider.walletBalance < requestedAmount THEN
        RETURN Error("Insufficient funds in wallet.")
    END IF

    // 3. Temporarily Deduct Funds (Prevent double-spending)
    rider.walletBalance = rider.walletBalance - requestedAmount
    DATABASE.RiderProfile.save(rider)

    // 4. Create Payout Record
    payout = NEW RiderPayout(
        riderId = riderId,
        amount = requestedAmount,
        status = "PENDING",
        type = "WITHDRAWAL",
        createdAt = CURRENT_TIMESTAMP
    )
    DATABASE.RiderPayout.save(payout)

    // 5. Trigger External Payment Gateway (e.g., Stripe)
    transferResult = StripeAPI.transferFunds(riderId, requestedAmount)

    // 6. Handle Gateway Response
    IF transferResult == SUCCESS THEN
        payout.status = "TRANSFERRED"
```

```
    payout.transferredAt = CURRENT_TIMESTAMP
    DATABASE.RiderPayout.save(payout)

    RETURN Success("Withdrawal transferred to your bank account.")
  ELSE
    // Rollback: Refund the wallet if the bank transfer fails
    rider.walletBalance = rider.walletBalance + requestedAmount
    DATABASE.RiderProfile.save(rider)

    payout.status = "FAILED"
    DATABASE.RiderPayout.save(payout)

    RETURN Error("Bank transfer failed. Funds have been returned to your wallet.")
  END IF

END FUNCTION
```

8. Appendices

8.1 Class Diagram

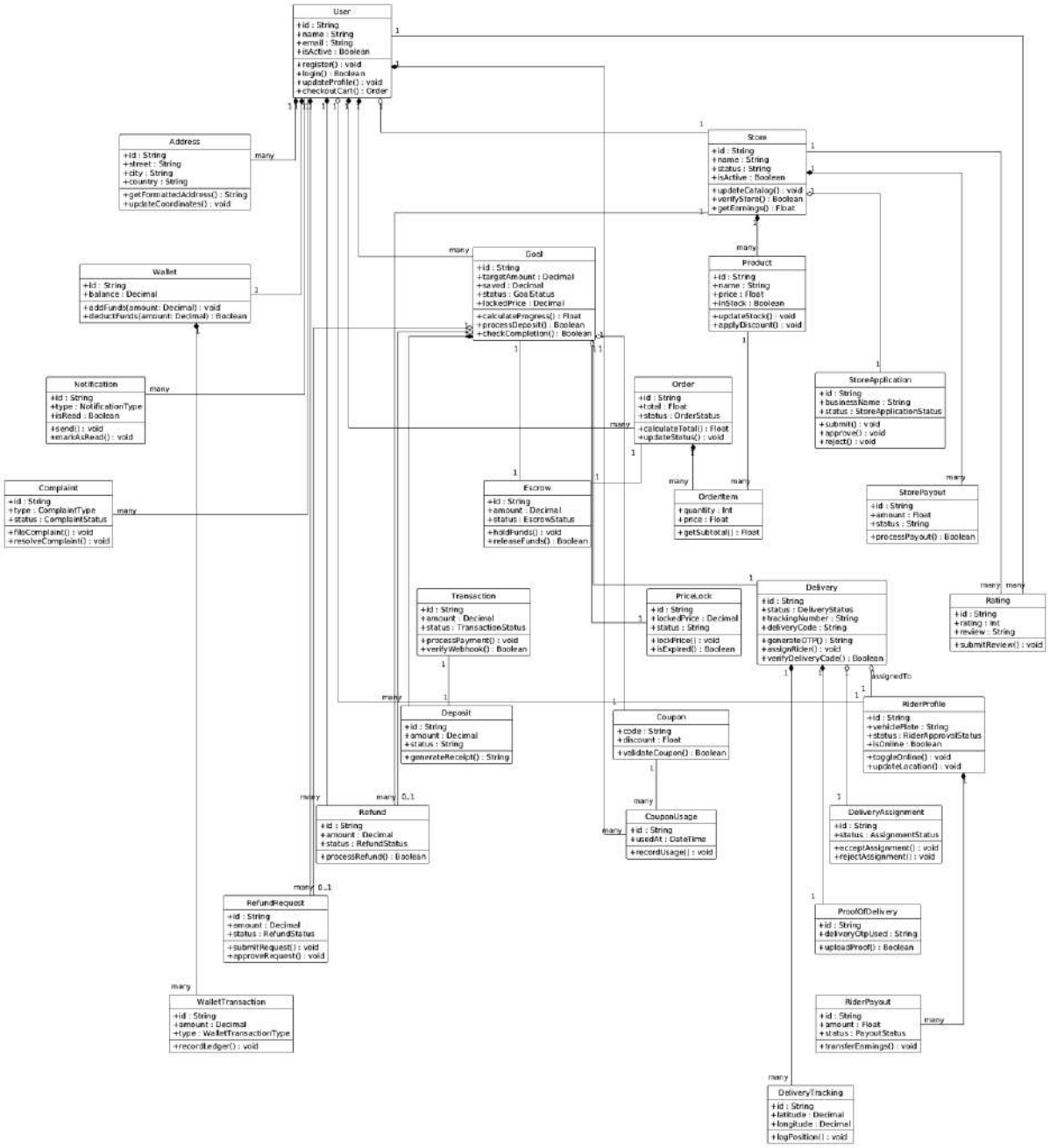


Figure 61: Class Diagram

8.2 Object Diagram

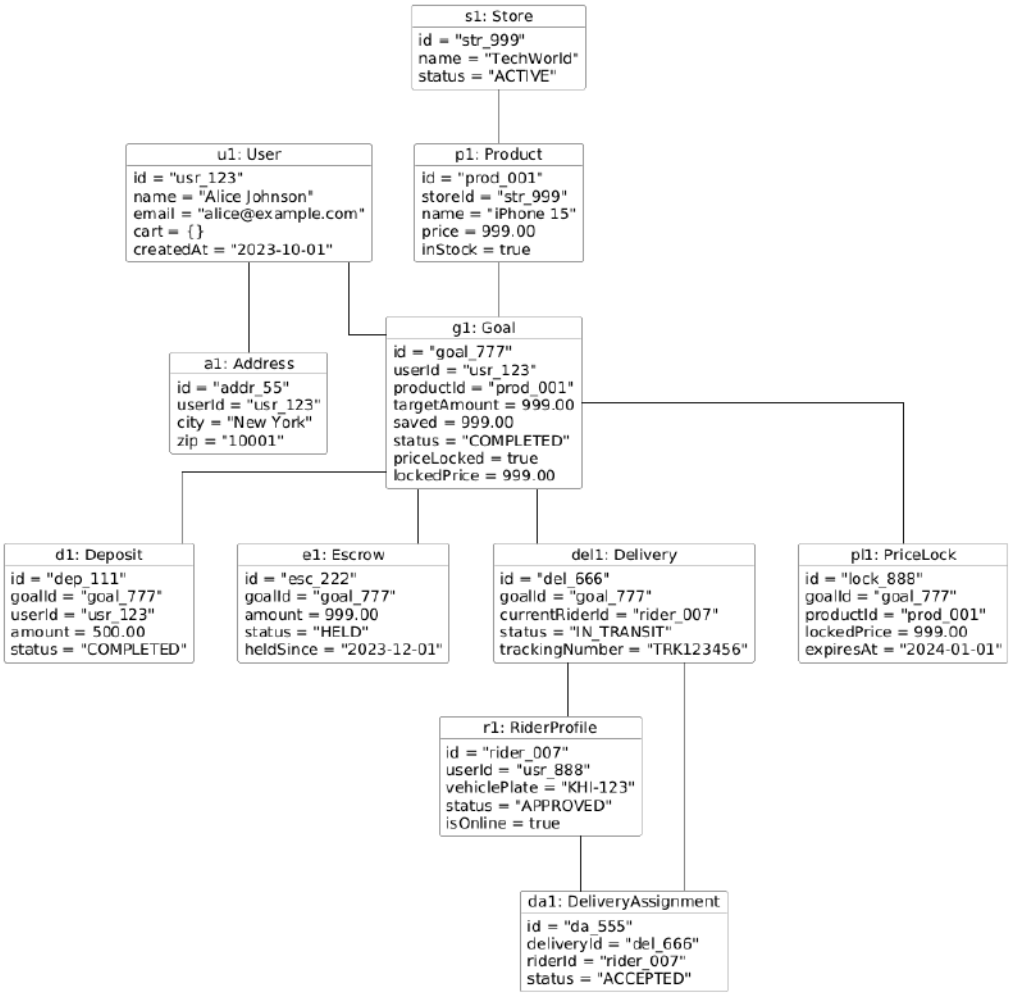


Figure 62: Object Diagram

8.3 State Chart Diagram

8.3.1 Admin State Chart Diagram

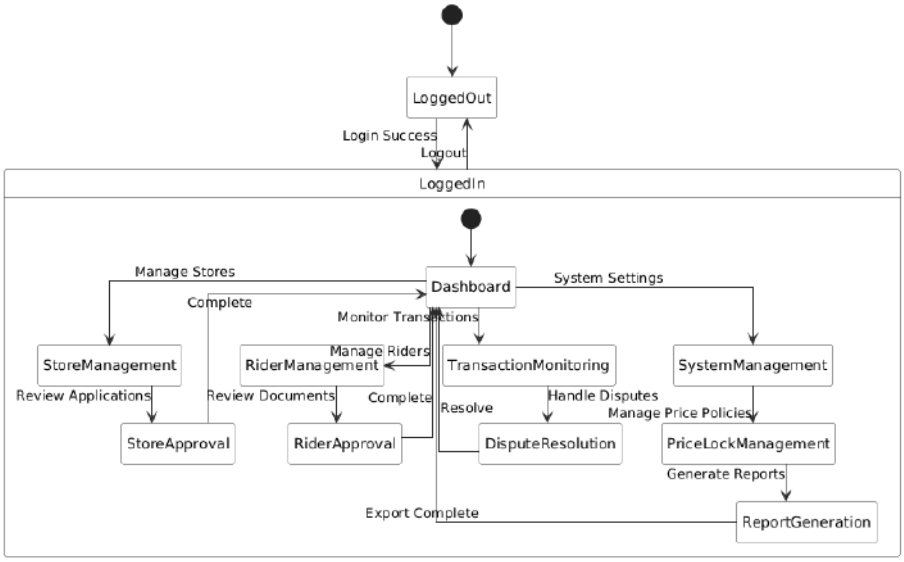


Figure 63: Admin State Chart Diagram

8.3.2 User State Chart Diagram

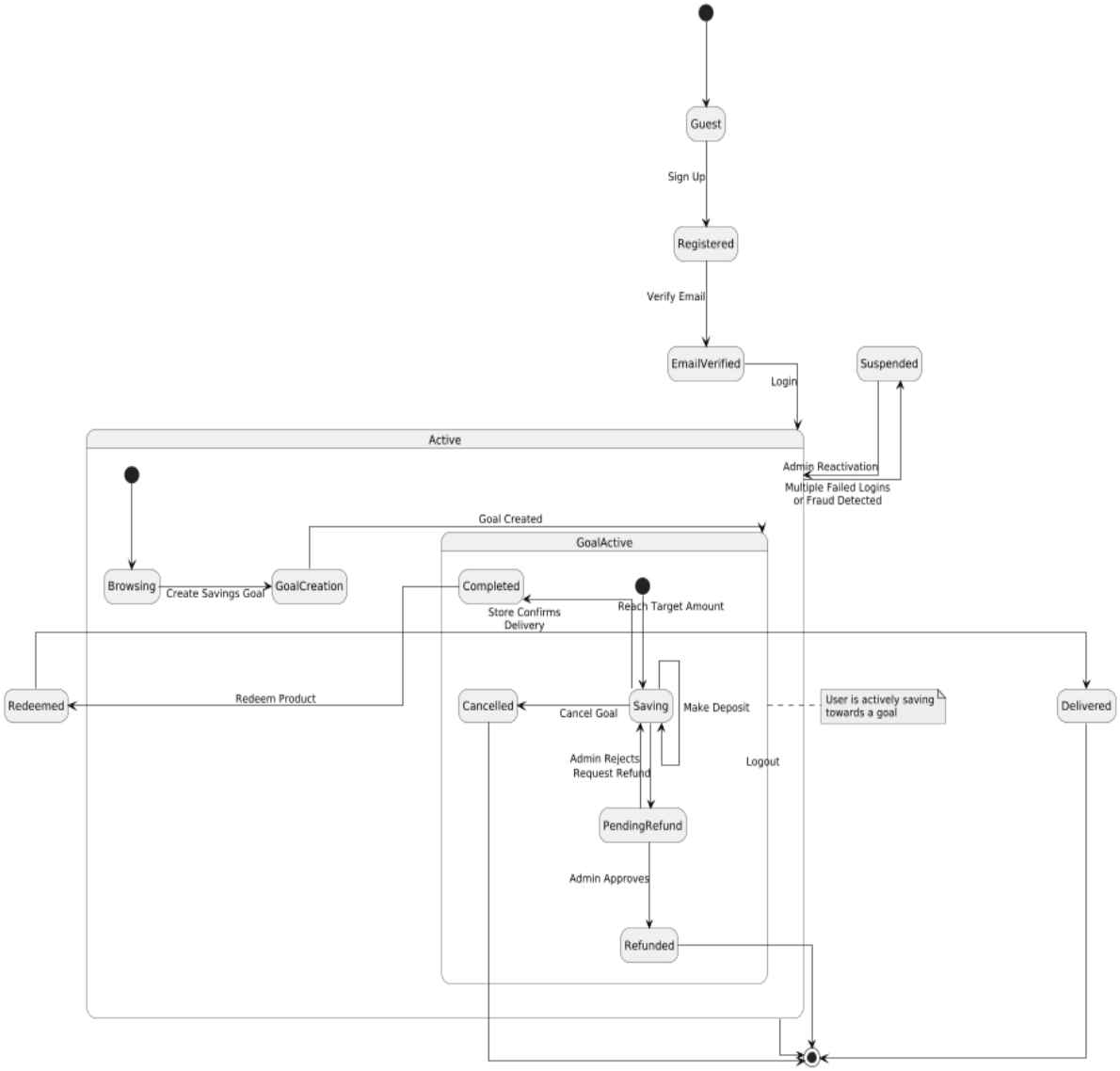


Figure 64: User State Chart Diagram

8.3.3 Store State Chart Diagram

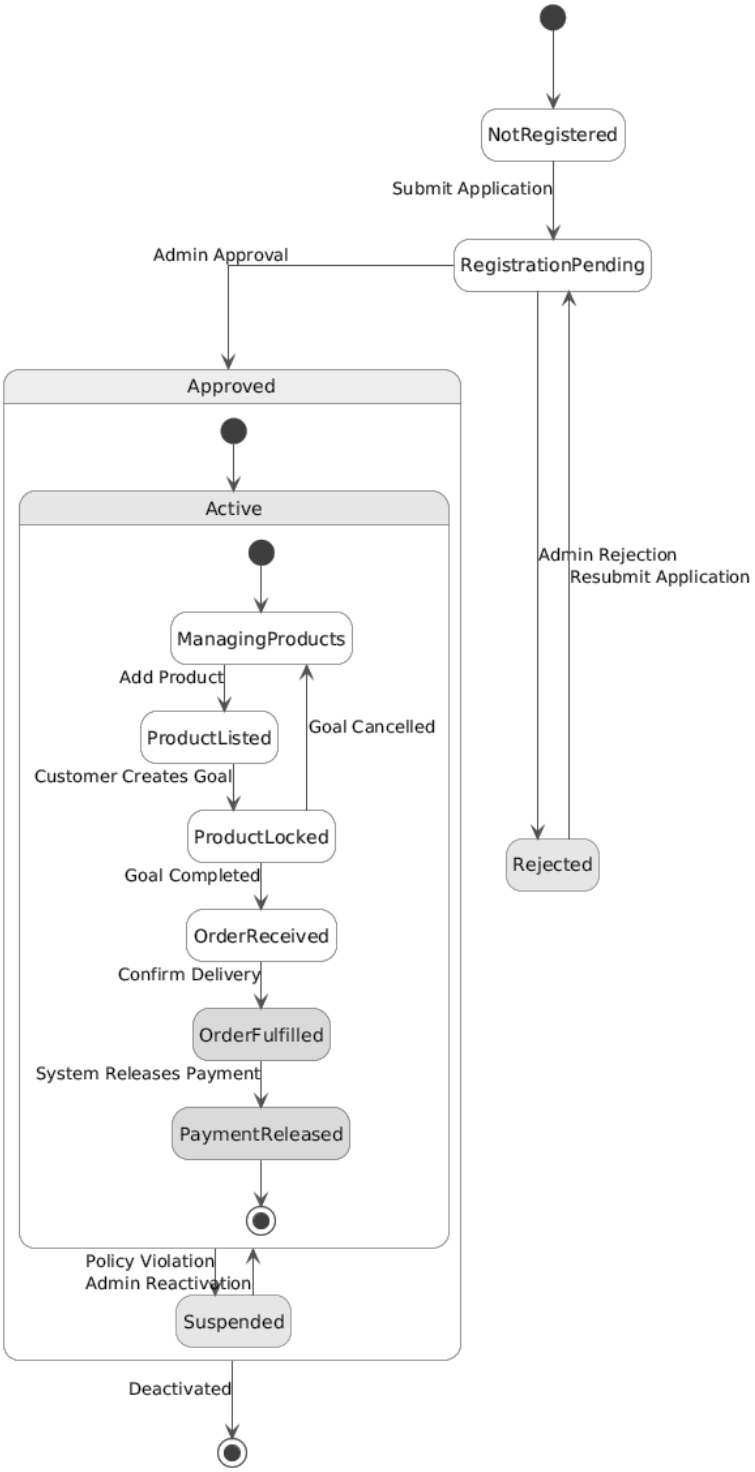


Figure 65: Store State Chart Diagram

8.3.4 Rider State Chart Diagram

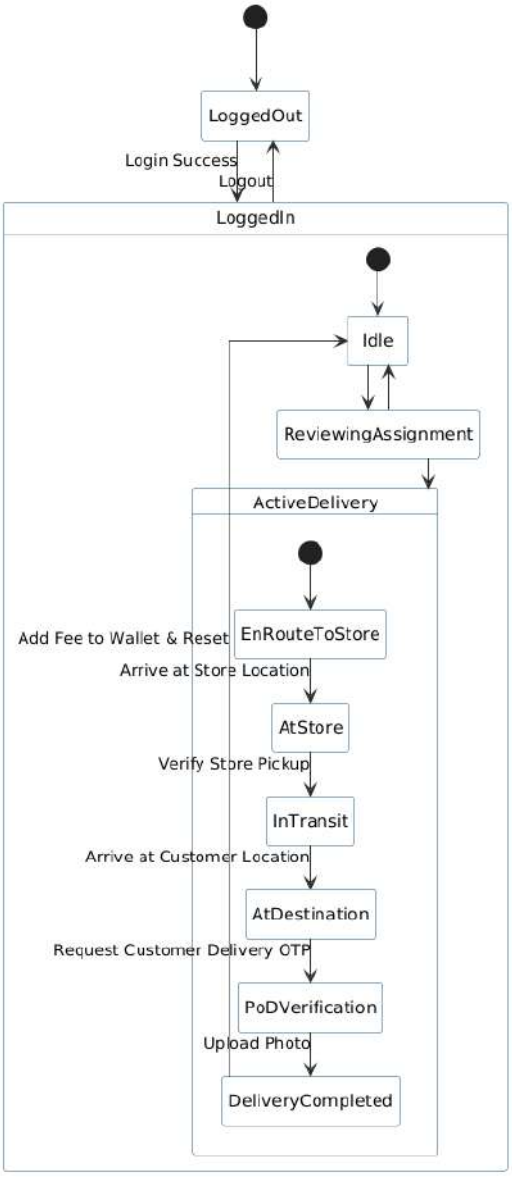


Figure 66: Rider State Chart Diagram

8.4 Activity Diagram

8.4.1 User Registration

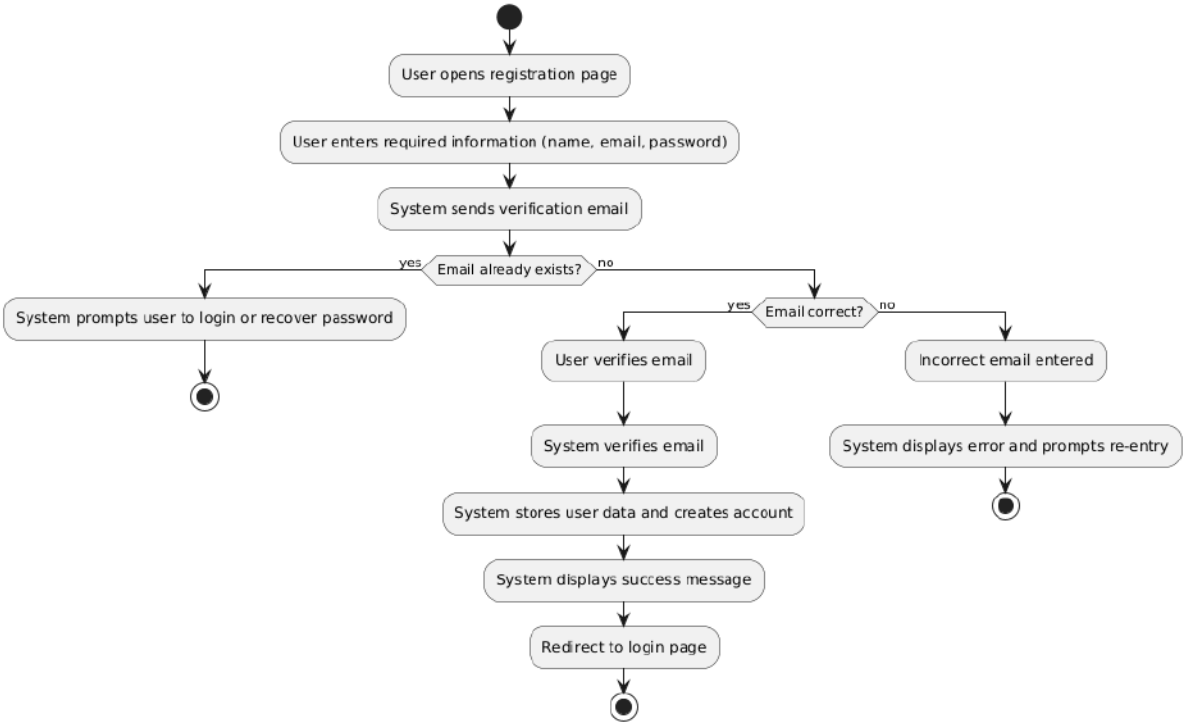


Figure 67: User Registration - Activity Diagram

8.4.2 User Login

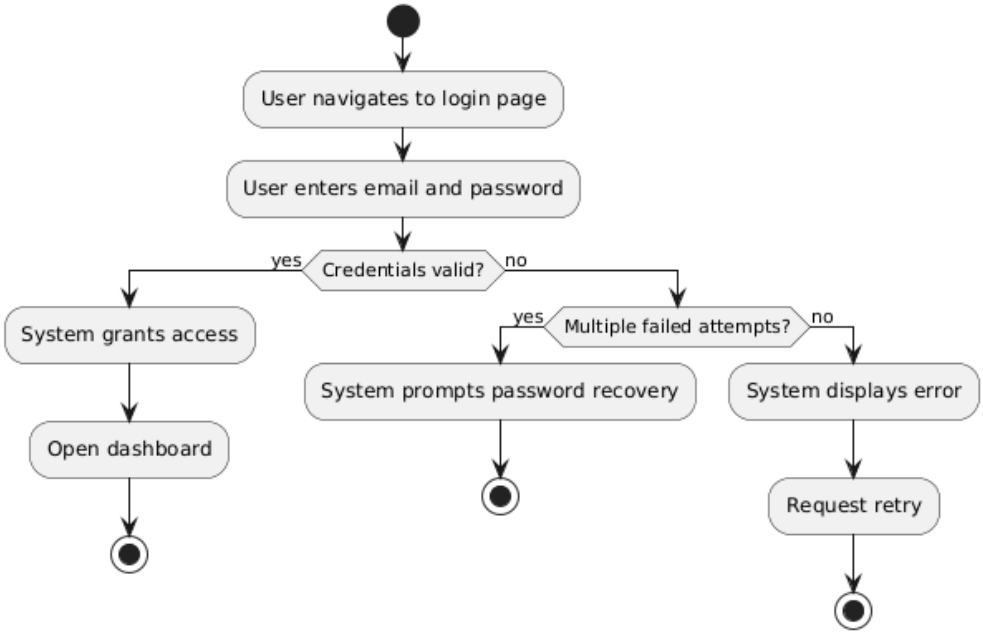


Figure 68: User Login - Activity Diagram

8.4.3 Create Saving Goal

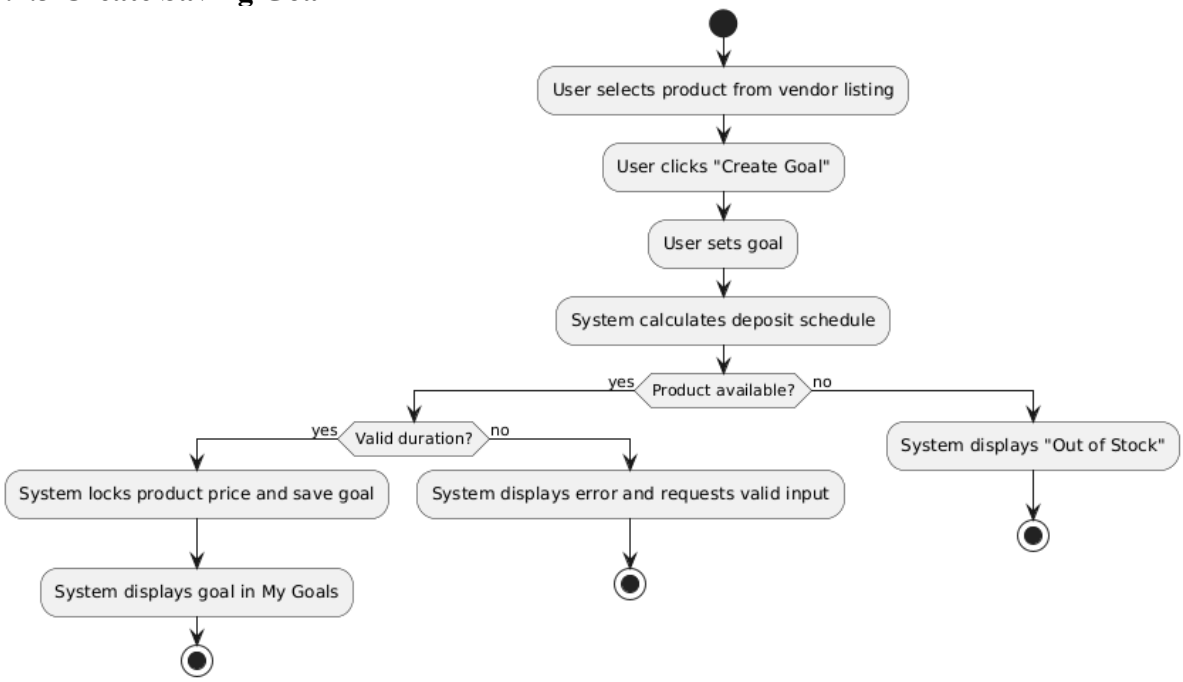


Figure 69: Create Saving Goal - Activity Diagram

8.4.4 Deposit to Saving Goal

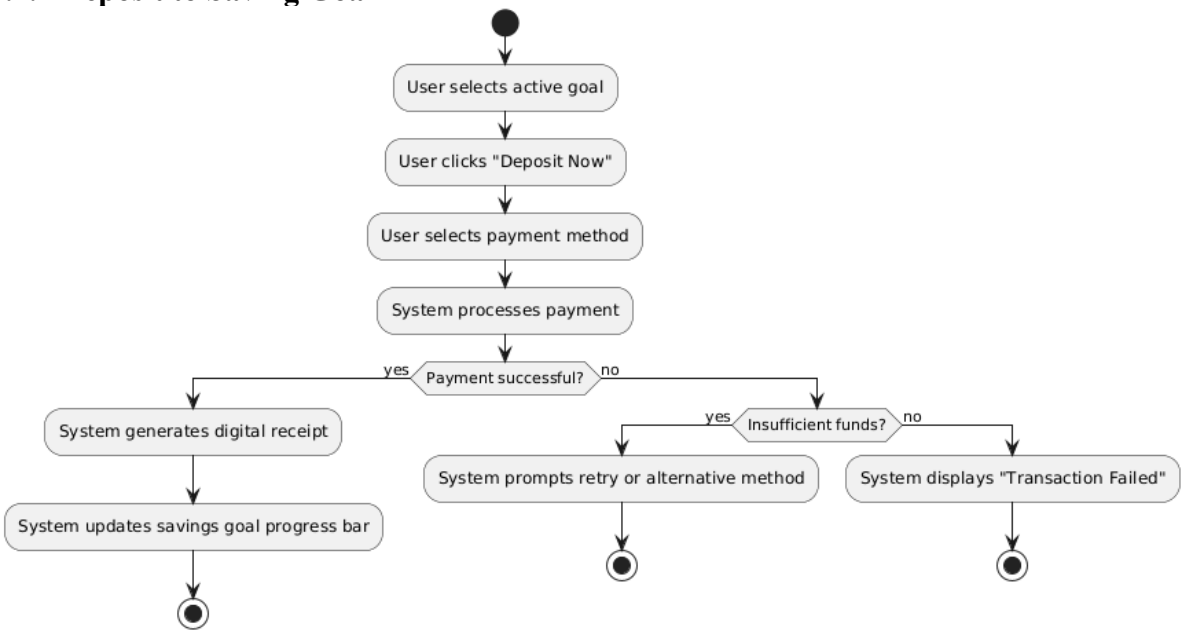


Figure 70: Deposit to Saving Goal - Activity Diagram

8.4.5 Track Saving Progress

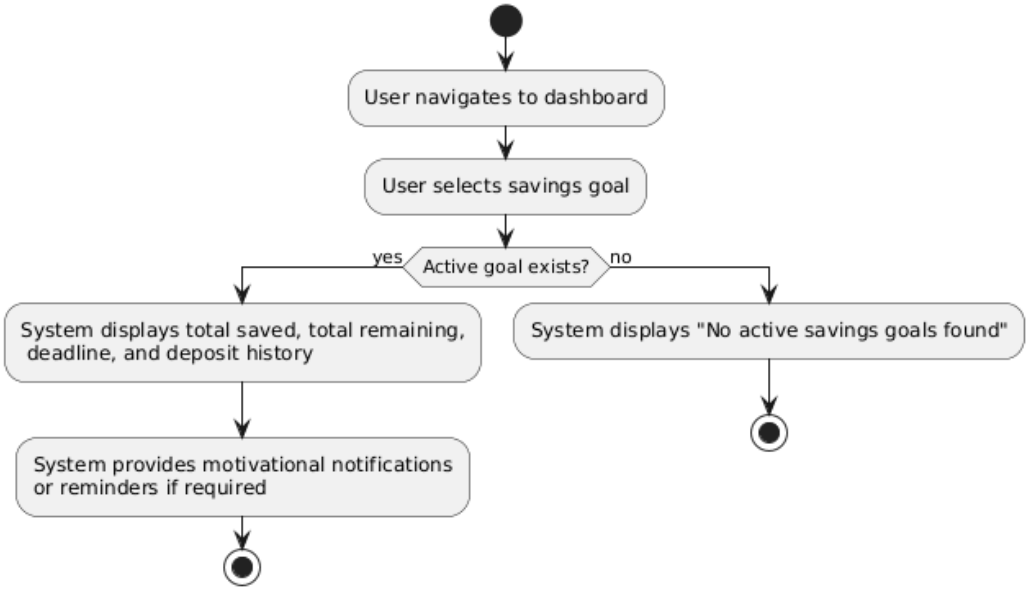


Figure 71: Create Saving Goal - Activity Diagram

8.4.6 Request Goal Cancellation & Refund

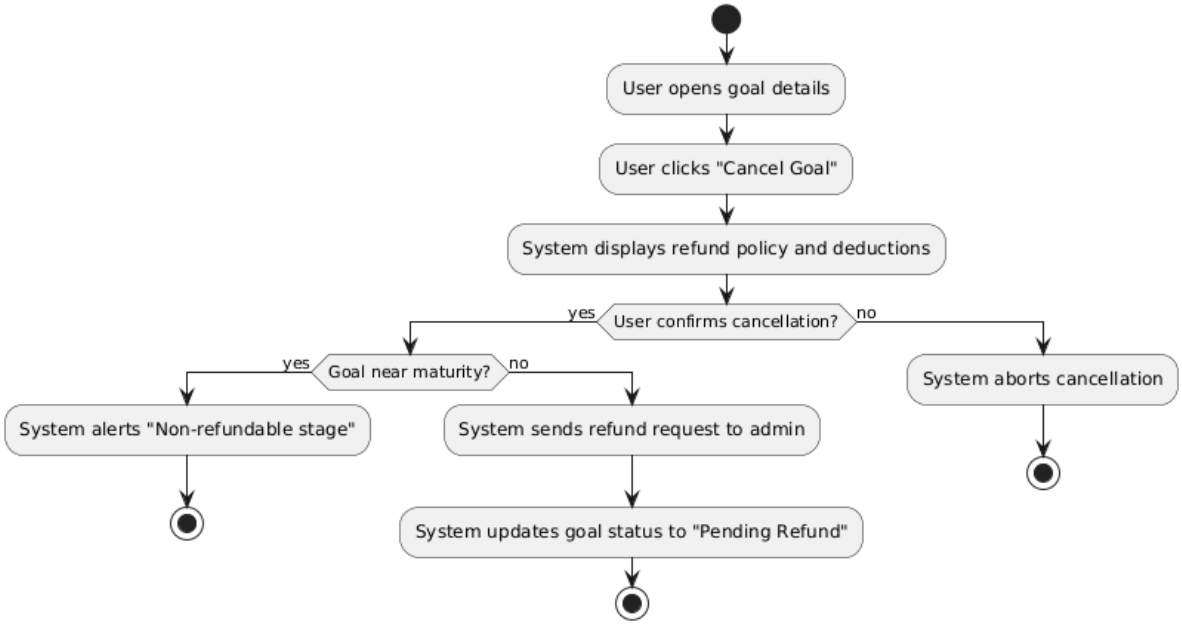


Figure 72: Request Goal Cancellation & Refund - Activity Diagram

8.4.7 Redeem Product After Goal Completion

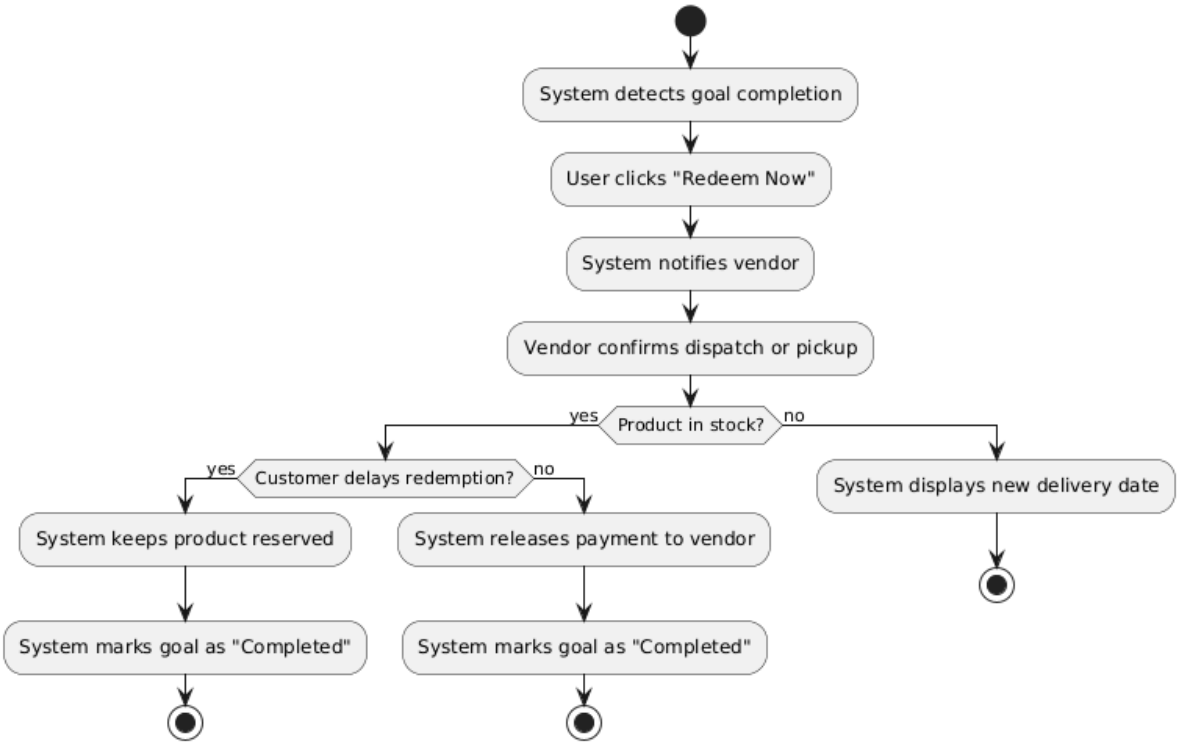


Figure 73: Redeem Product After Goal Completion - Activity Diagram

8.4.8 Receive Notification

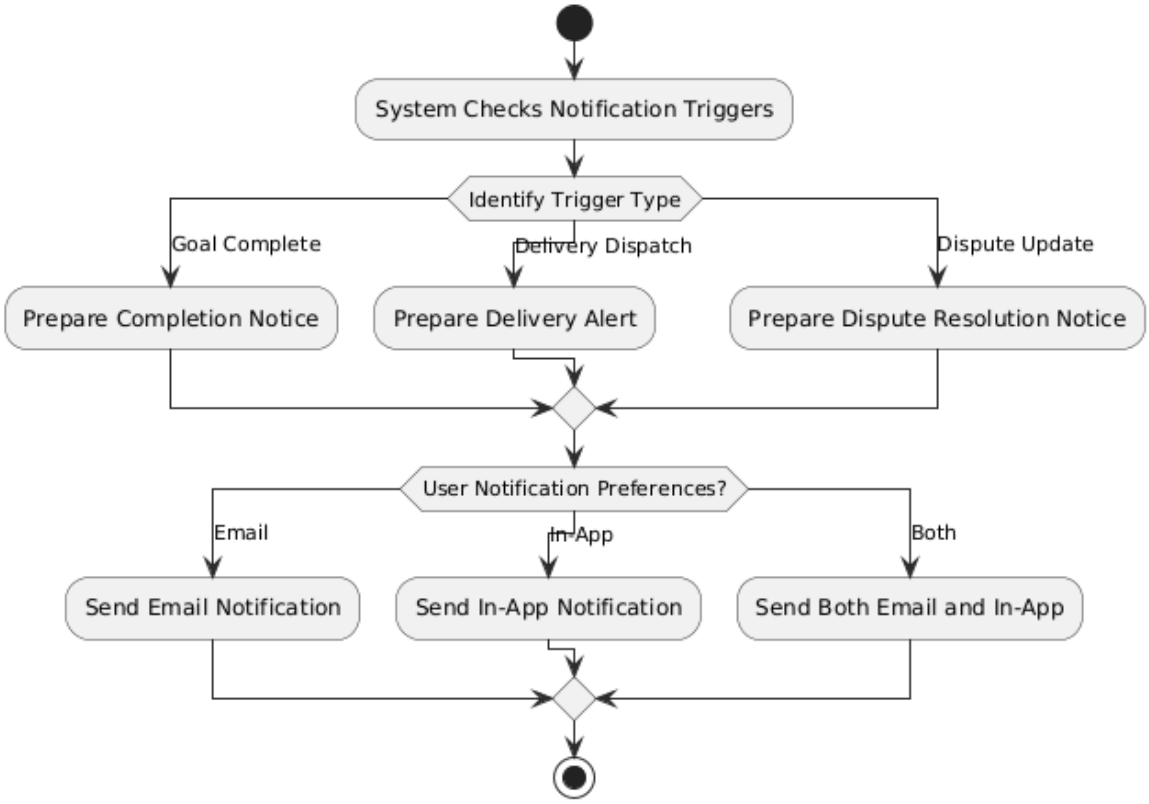


Figure 74: Receive Notification and Reminders - Activity Diagram

8.4.9 Store Registration

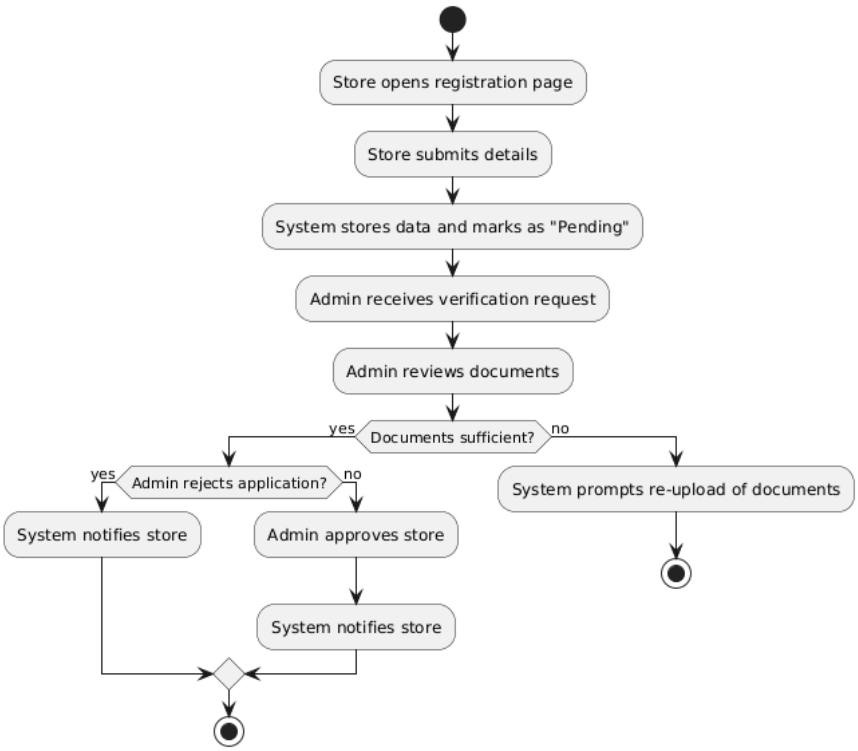


Figure 75: Store Registration - Activity Diagram

8.4.10 Product Listing by Store

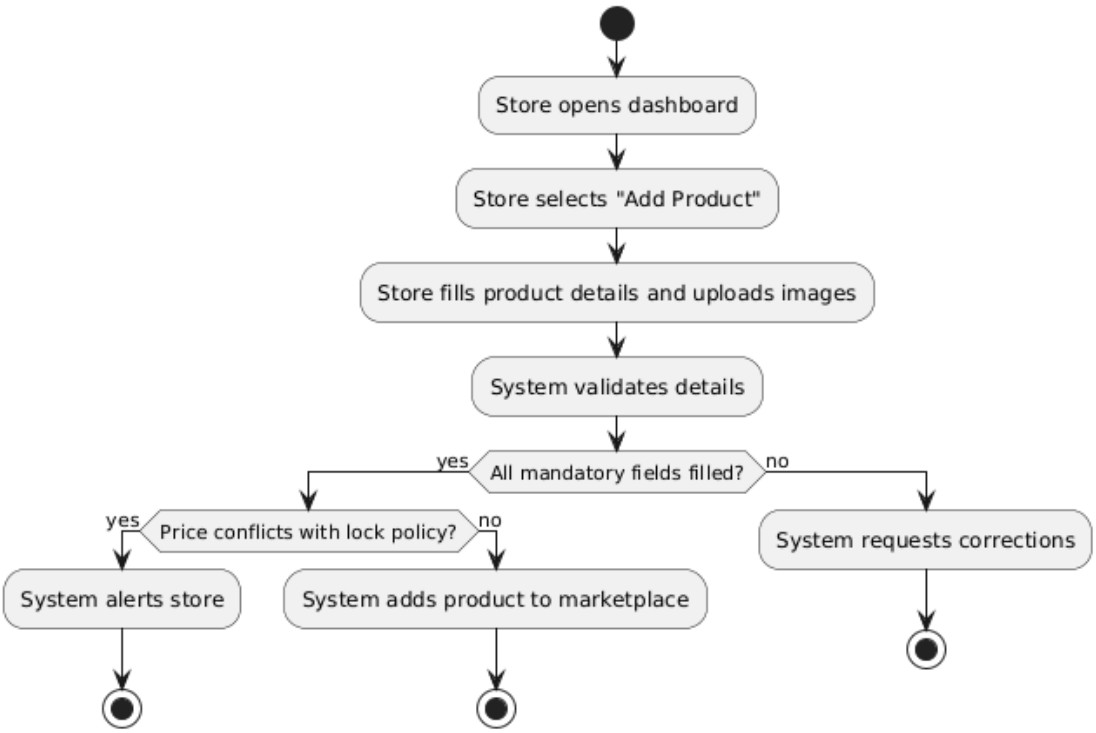


Figure 76: Product Listing by Store - Activity Diagram

8.4.11 Lock Product for Layaway

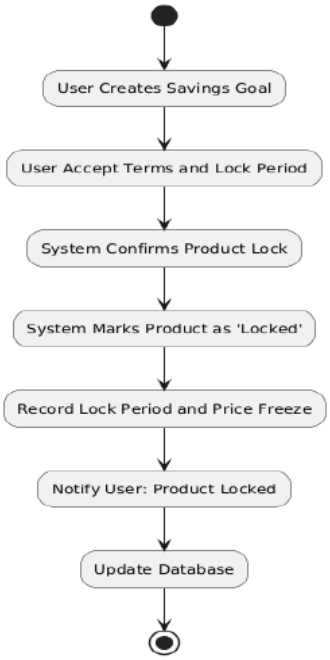


Figure 77: Lock Product for Layaway - Activity Diagram

8.4.12 Confirm Delivery & Final Payment Release

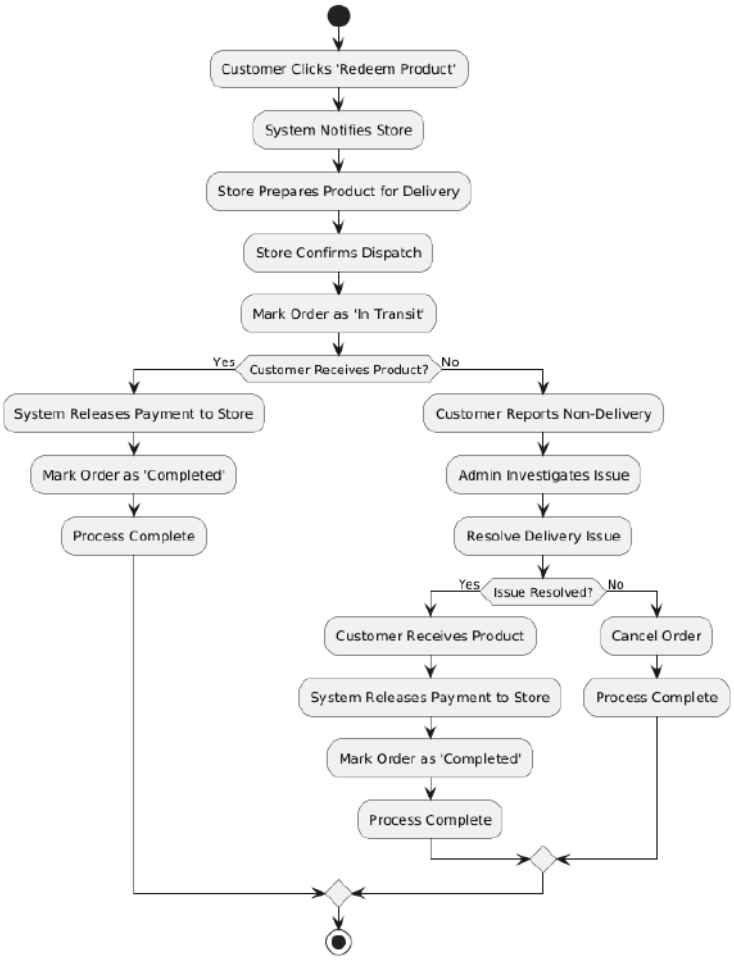


Figure 78: Confirm Delivery & Final Payment Release - Activity Diagram

8.4.13 View Store Dashboard & Earnings

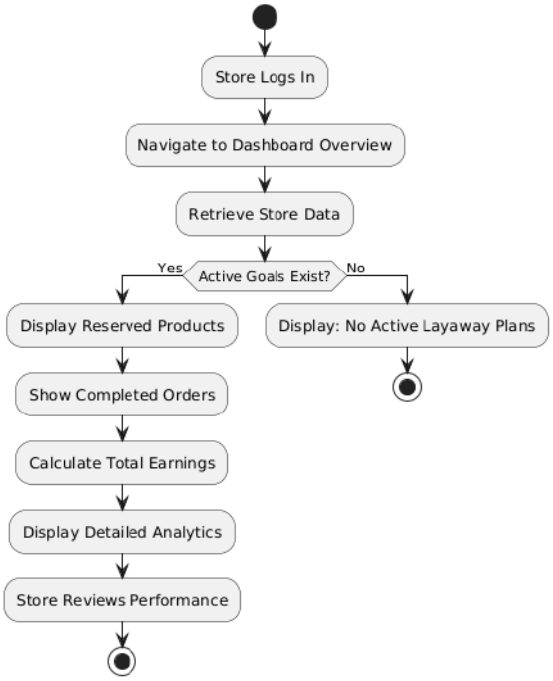


Figure 79: View Store Dashboard & Earnings - Activity Diagram

8.4.14 Approve or Reject Store Applications

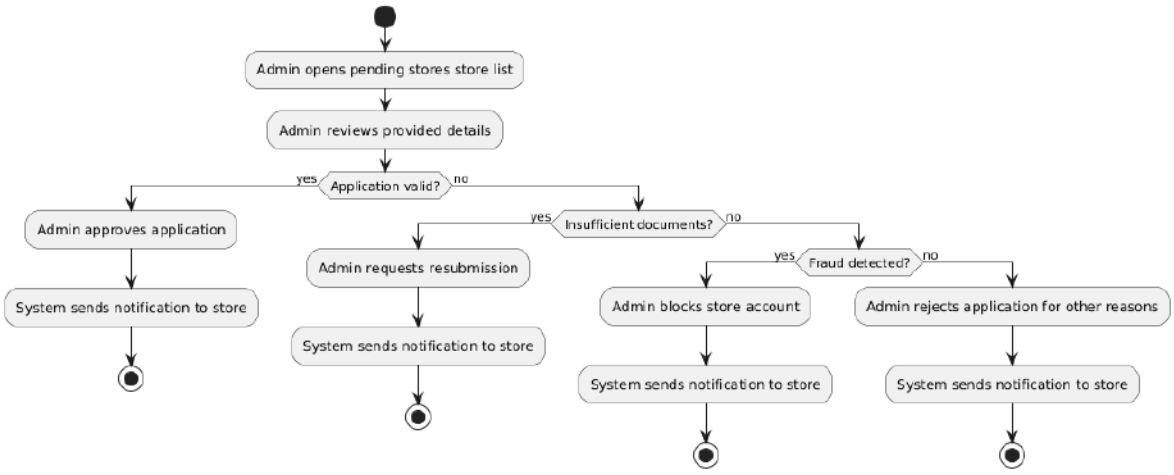


Figure 80: Approve or Reject Store Applications - Activity Diagram

8.4.15 Monitor Transactions & Resolve Disputes

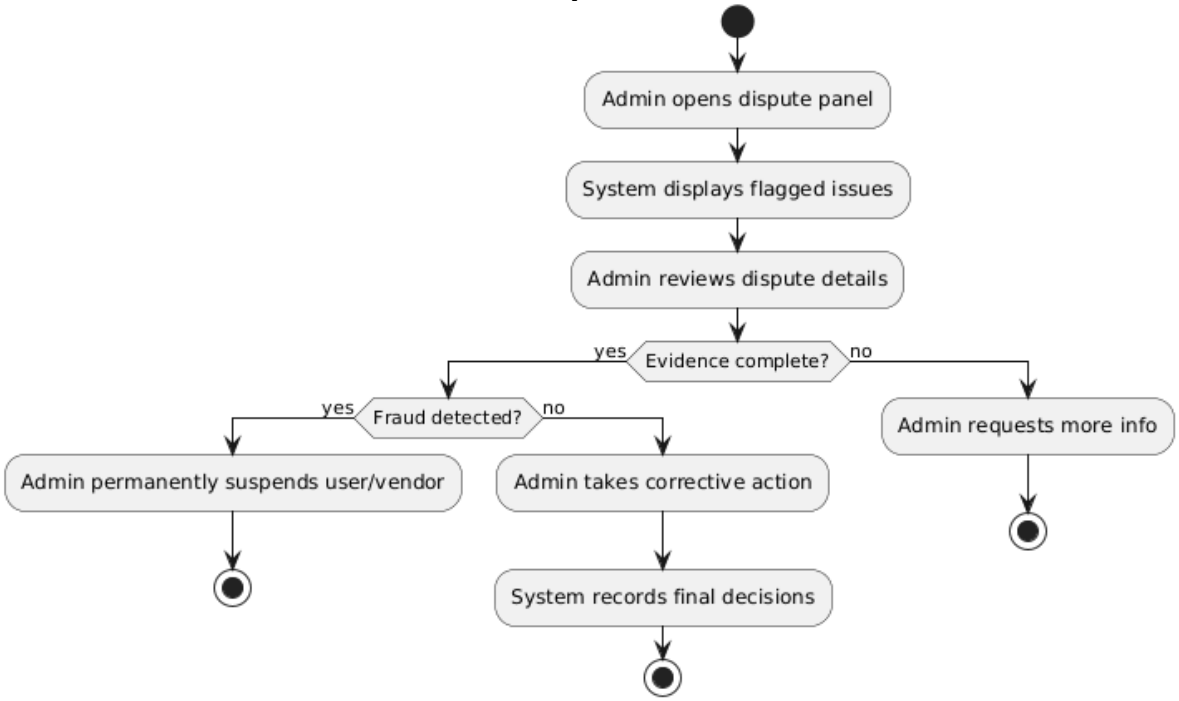


Figure 81: Monitors Transactions & Resolve Disputes - Activity Diagram

8.4.16 Manage Price Lock & Inflation Policy

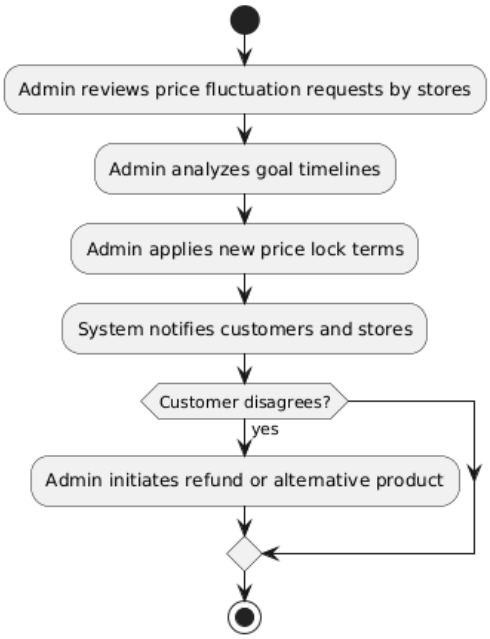


Figure 82: Manage Price Lock & Inflation Policy - Activity Diagram

8.4.17 Generate System Reports & Audit Logs

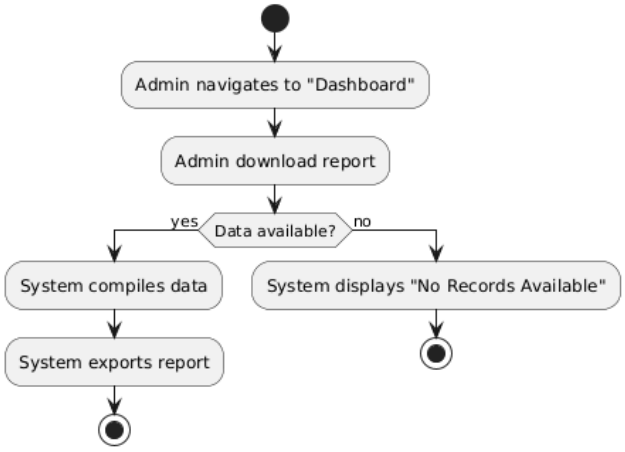


Figure 83: Generate System Reports & Audit Logs - Activity Diagram

8.4.18 Automated Notifications

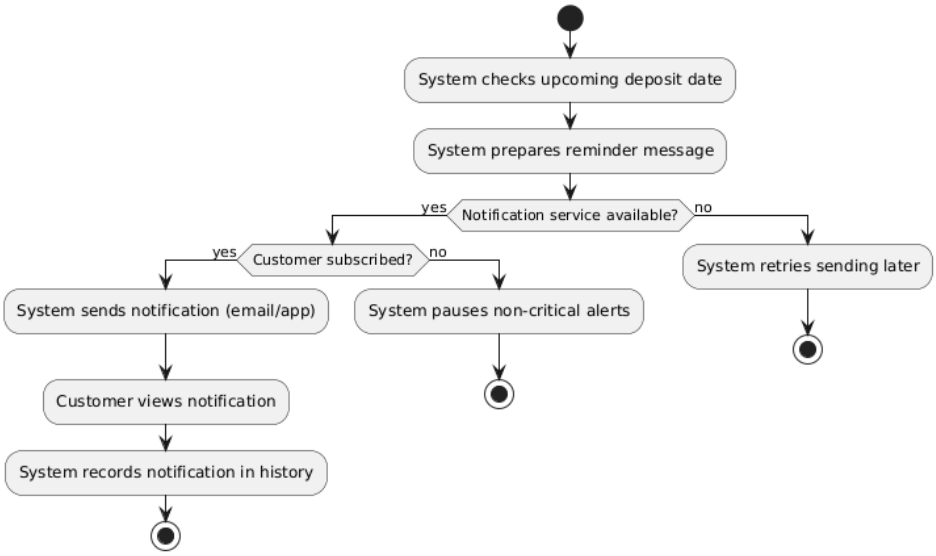


Figure 84: Automated Notifications - Activity Diagram

8.4.19 Track Delivery via Map

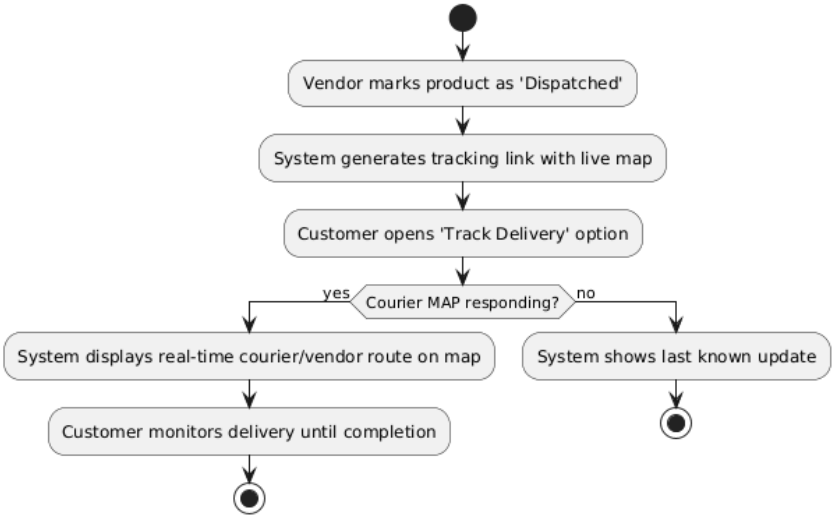


Figure 85: Track Delivery via Map - Activity Diagram

8.4.20 Apply Coupon

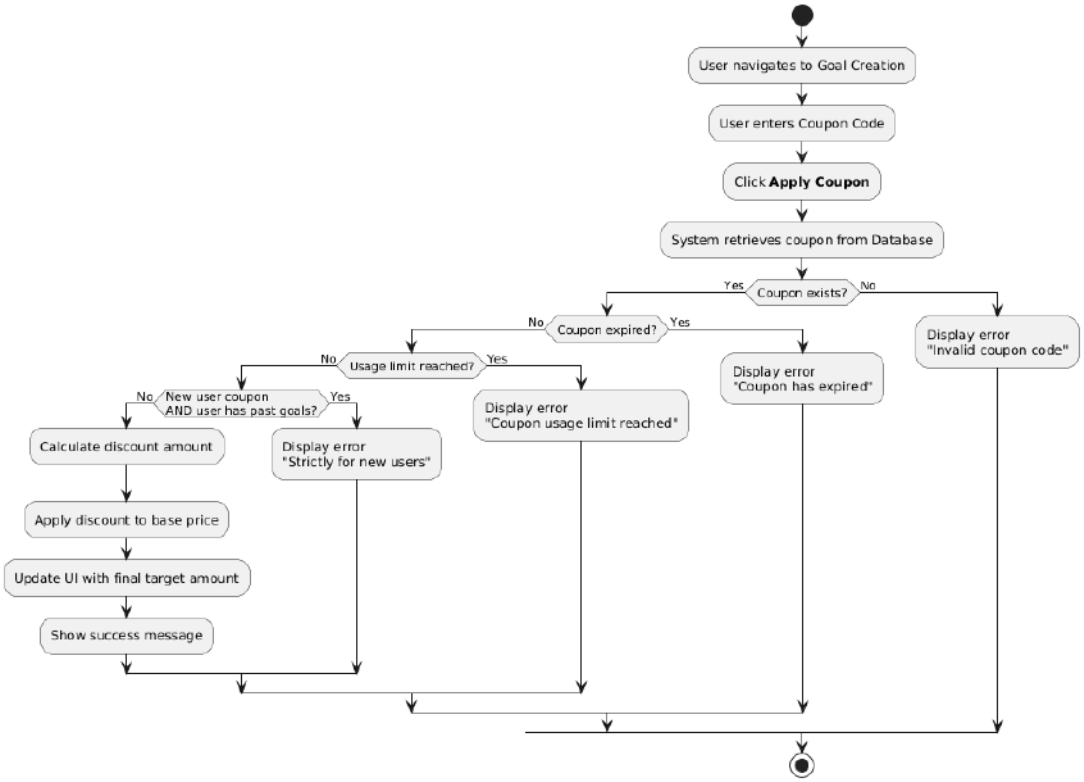


Figure 86: Apply Coupon - Activity Diagram

8.4.21 Digital Wallet

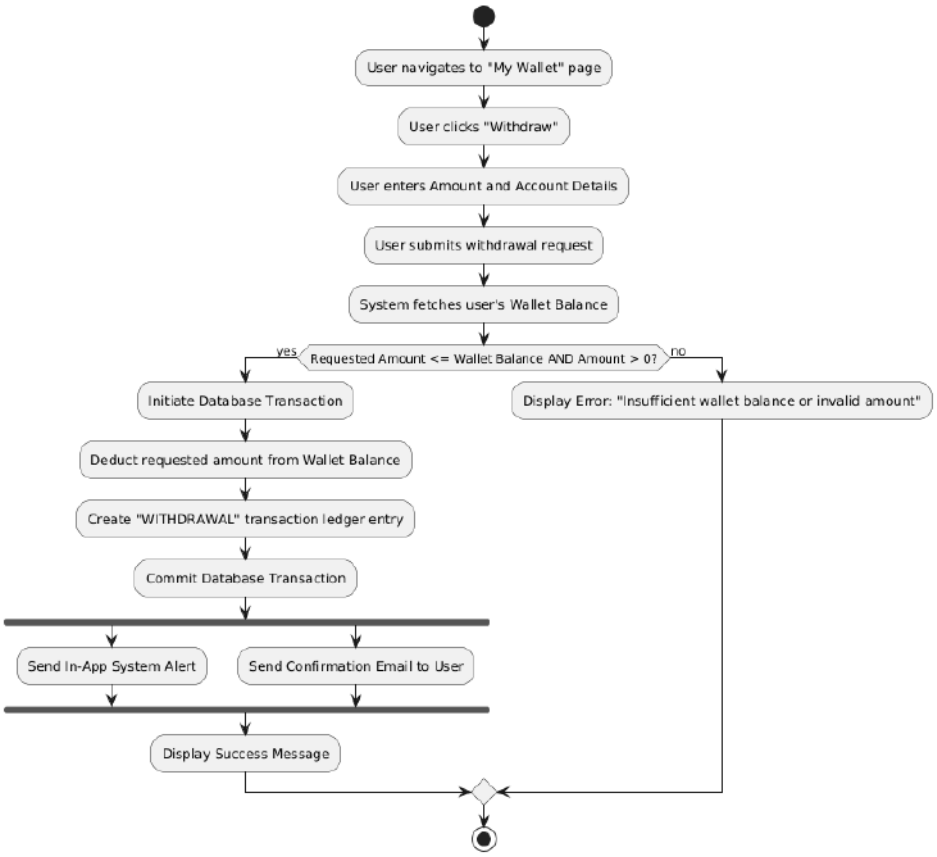


Figure 87: Digital Wallet - Activity Diagram

8.4.22 Support & Complaints

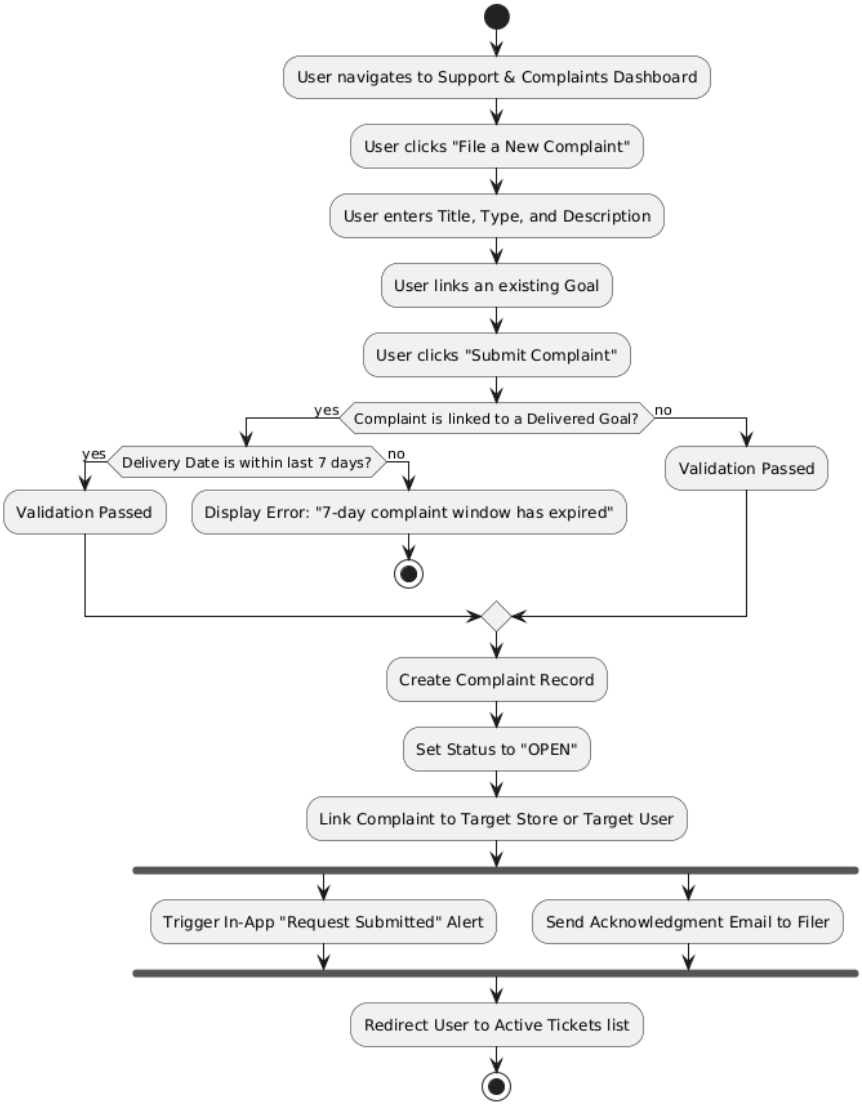


Figure 88: Support & Complaints - Activity Diagram

8.4.23 Rider Onboarding & Status Management

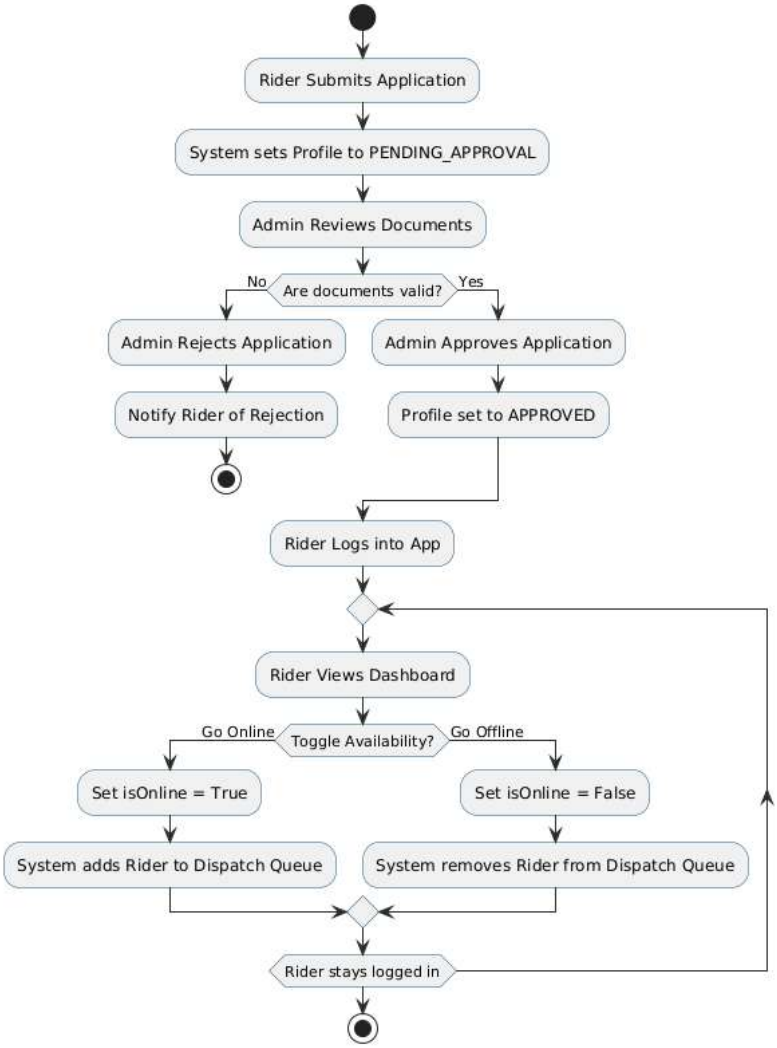


Figure 89: Rider Onboarding & Status Management - Activity Diagram

8.4.24 Delivery Assignment & Dispatch

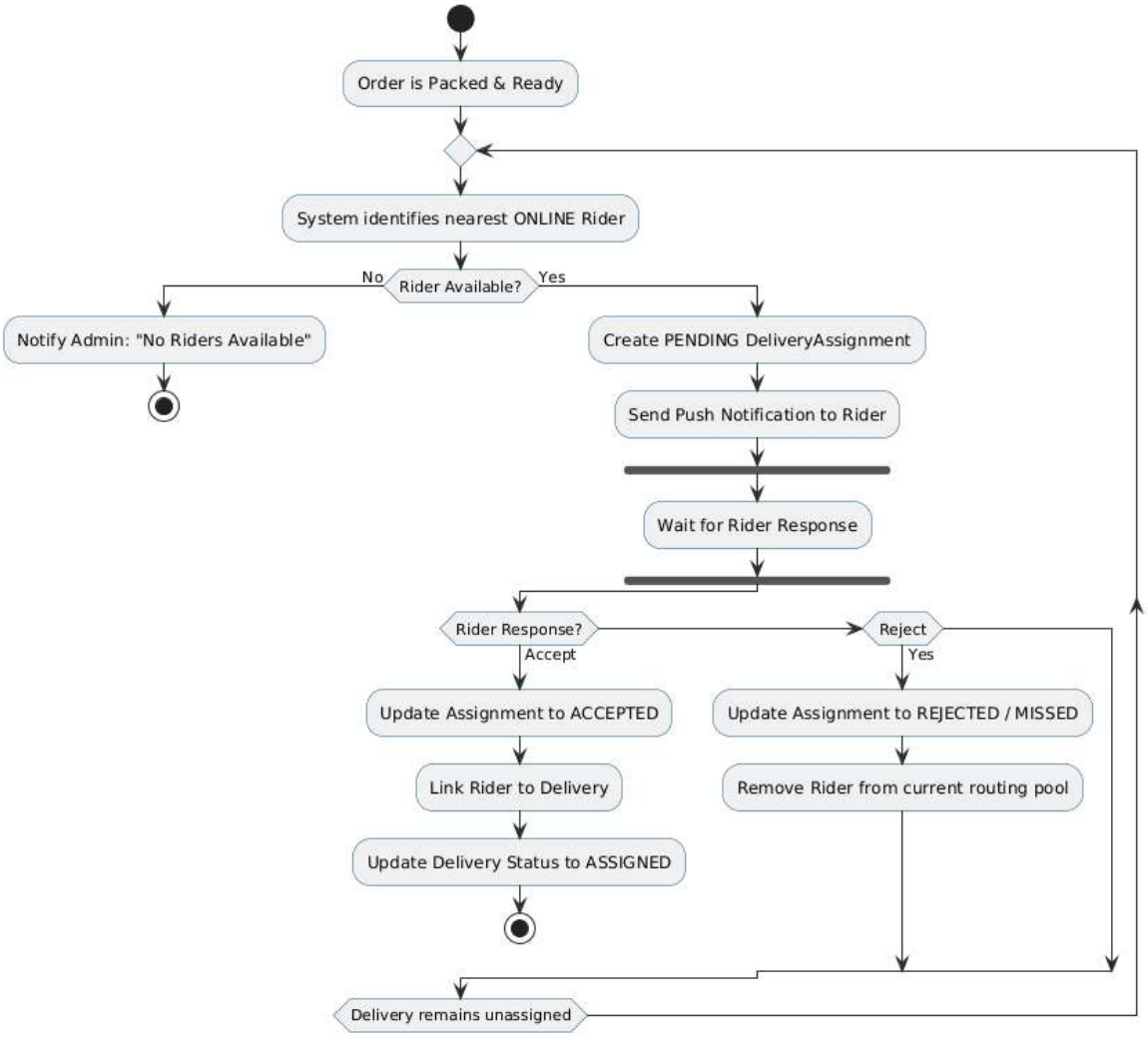


Figure 90: Delivery Assignment & Dispatch - Activity Diagram

8.5 Sequence Diagram

8.5.1 User Registration

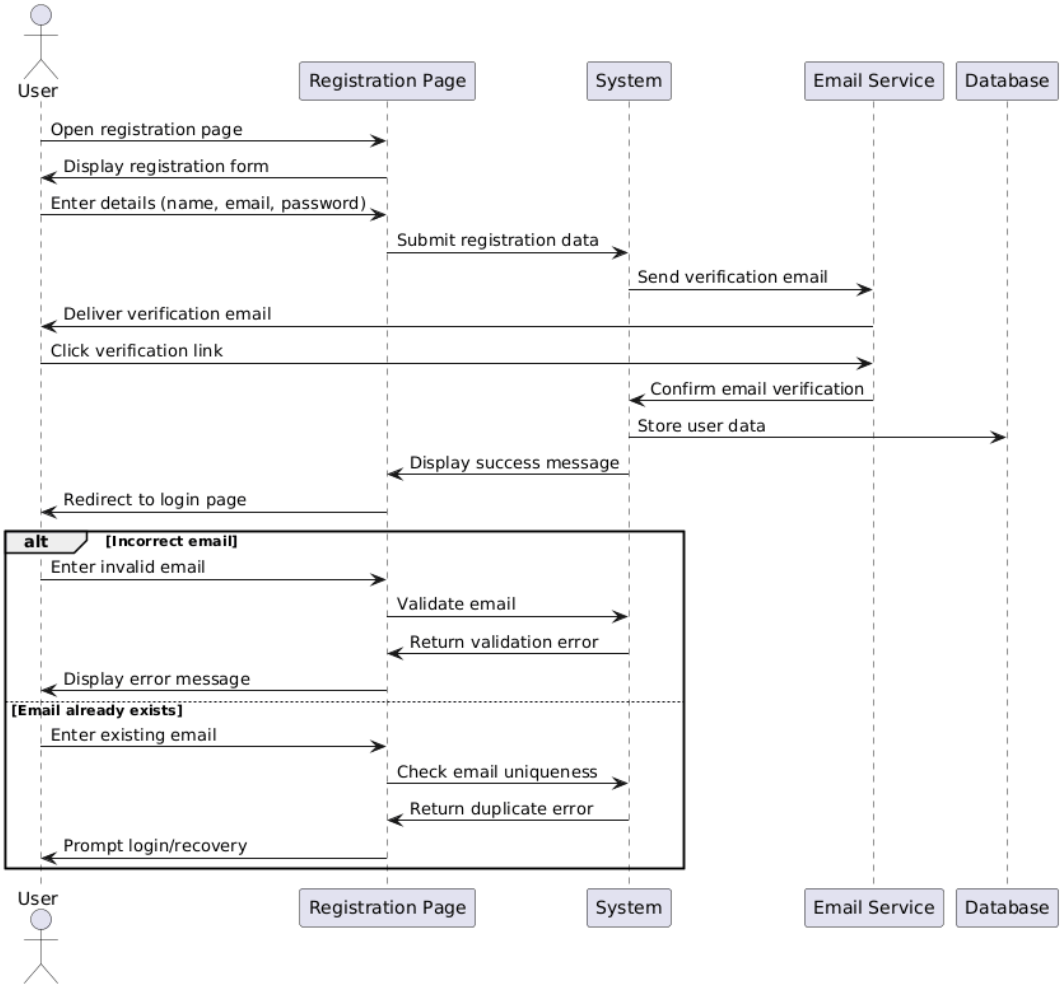


Figure 91: User Registration - Sequence Diagram

8.5.2 User Login

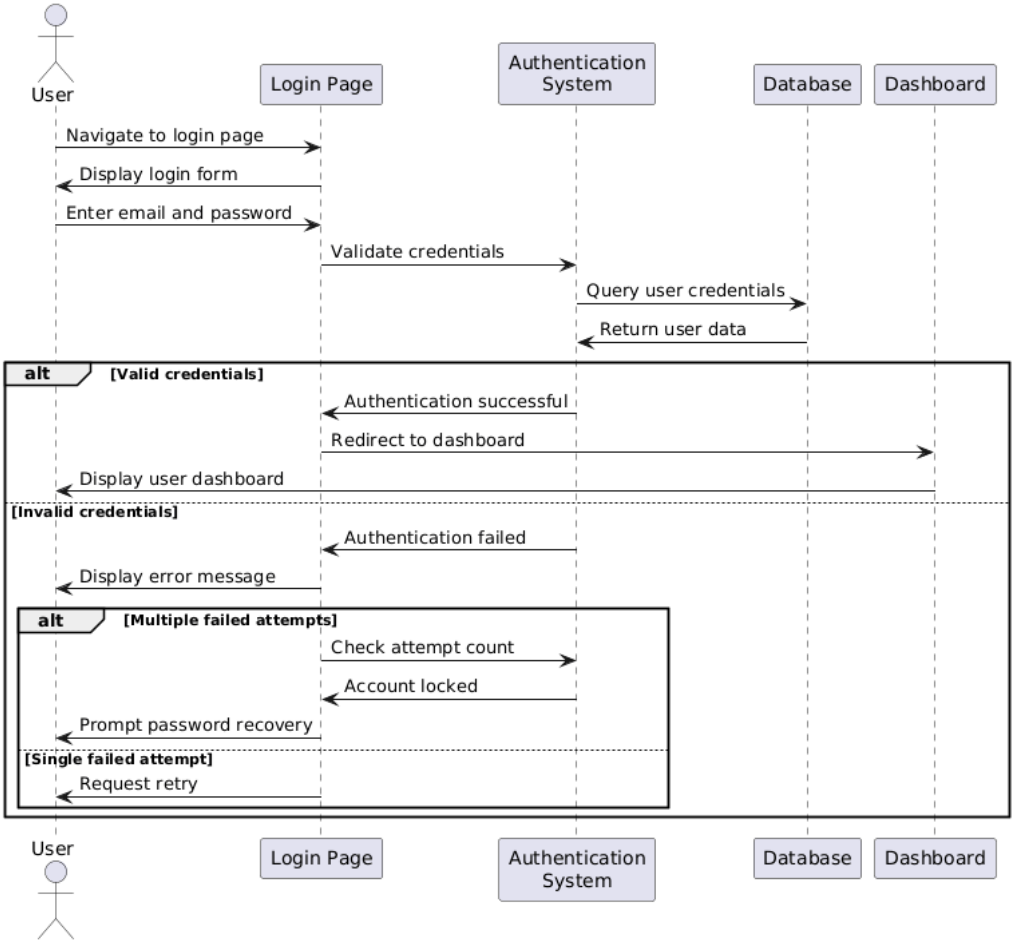


Figure 92: User Login - Sequence Diagram

8.5.3 Create Saving Goal

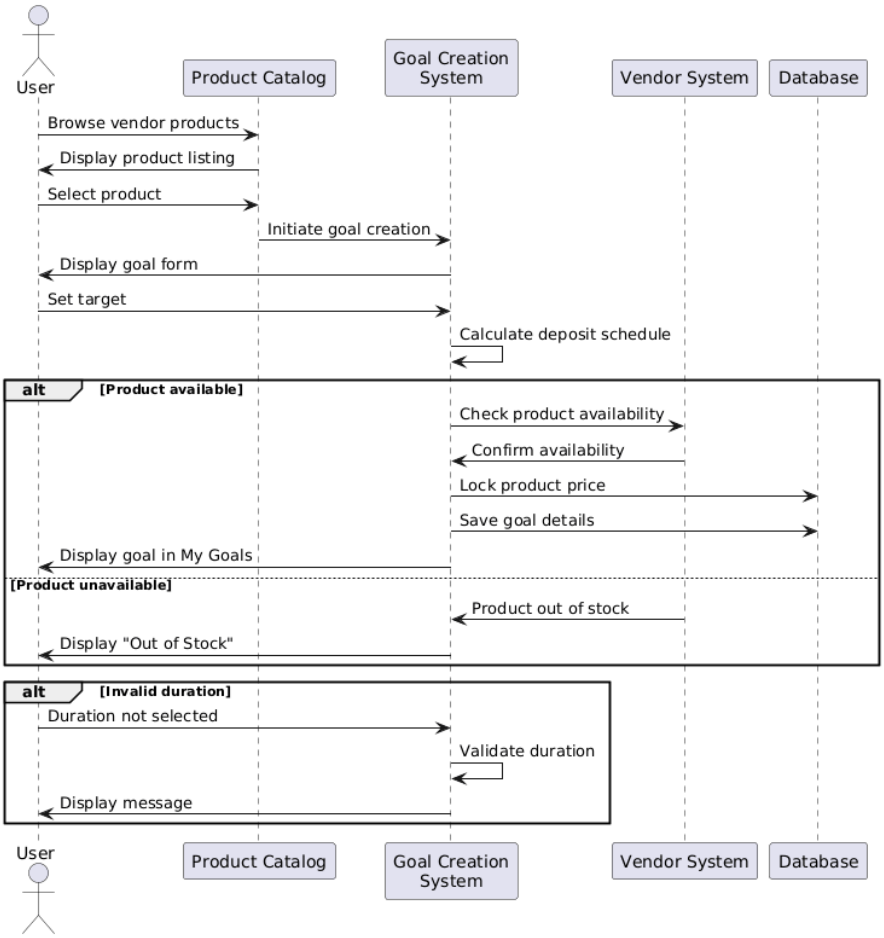


Figure 93: Create Saving Goal - Sequence Diagram

8.5.4 Deposit to Saving Goal

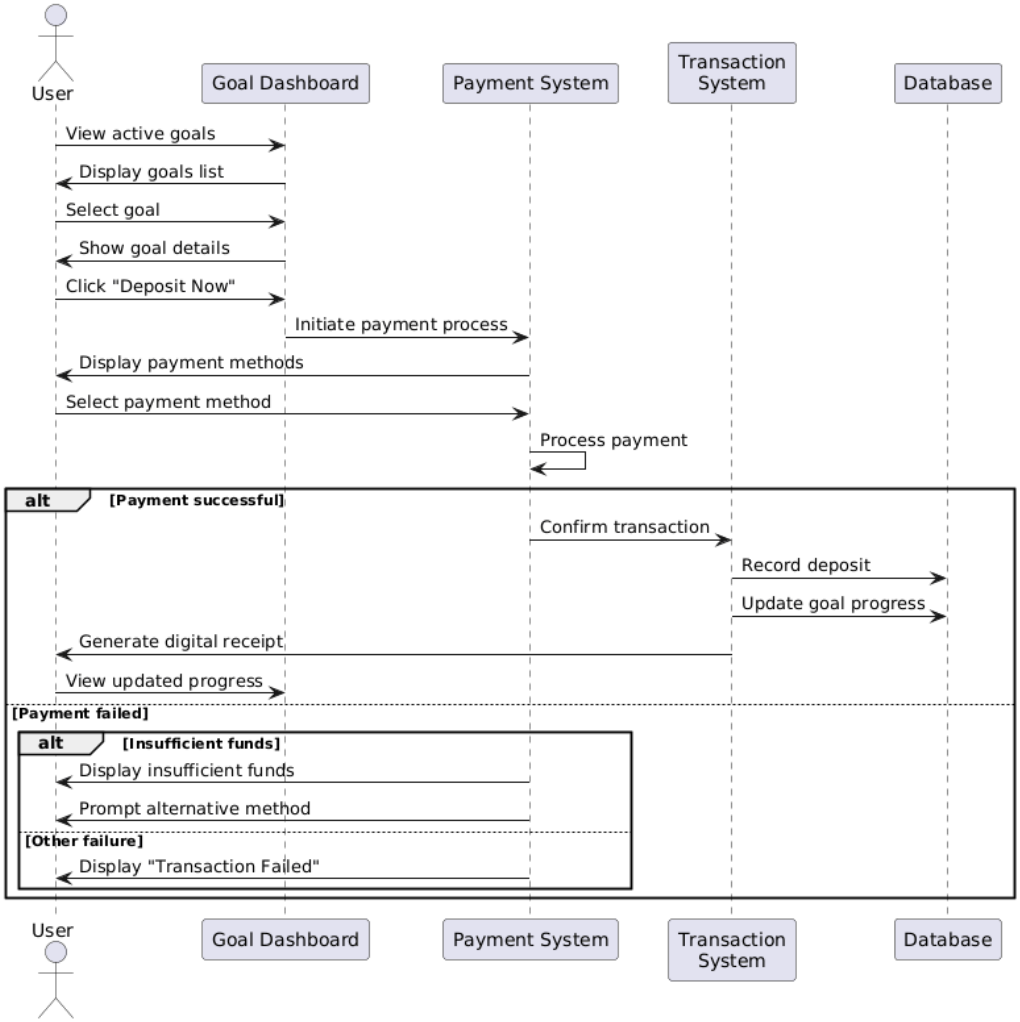


Figure 94: Deposit to Saving Goal - Sequence Diagram

8.5.5 Track Saving Progress

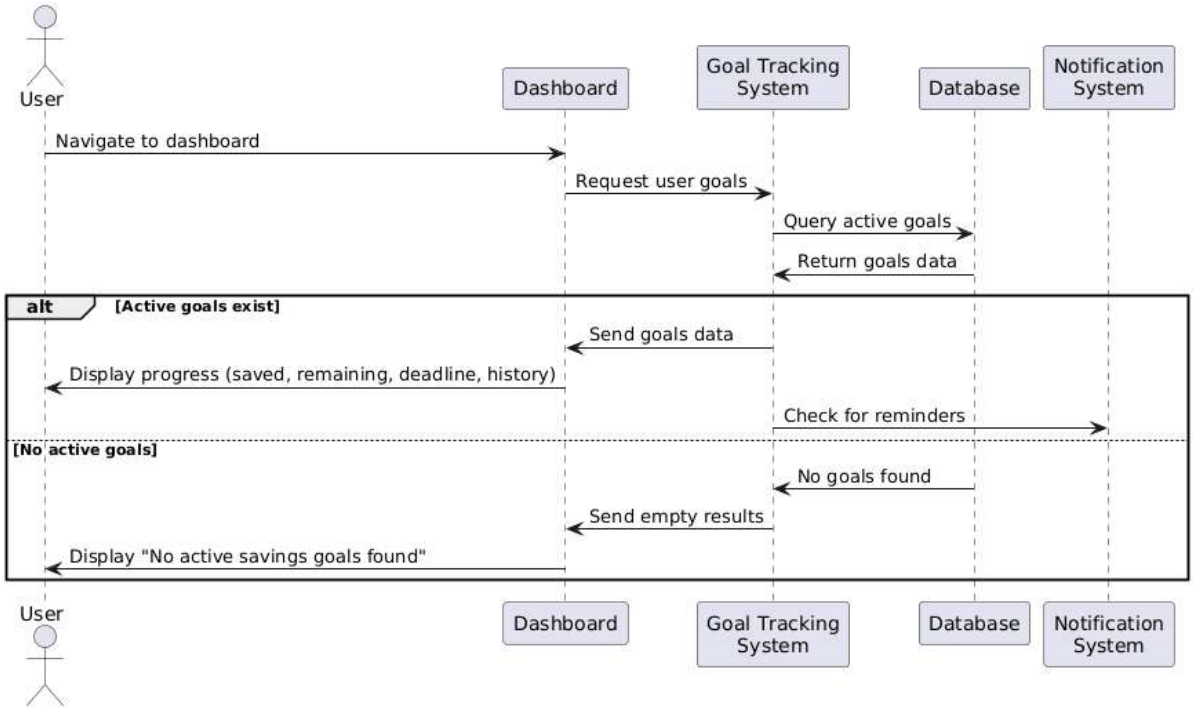


Figure 95: Track Saving Progress - Sequence Diagram

8.5.6 Request Goal Cancellation & Refund

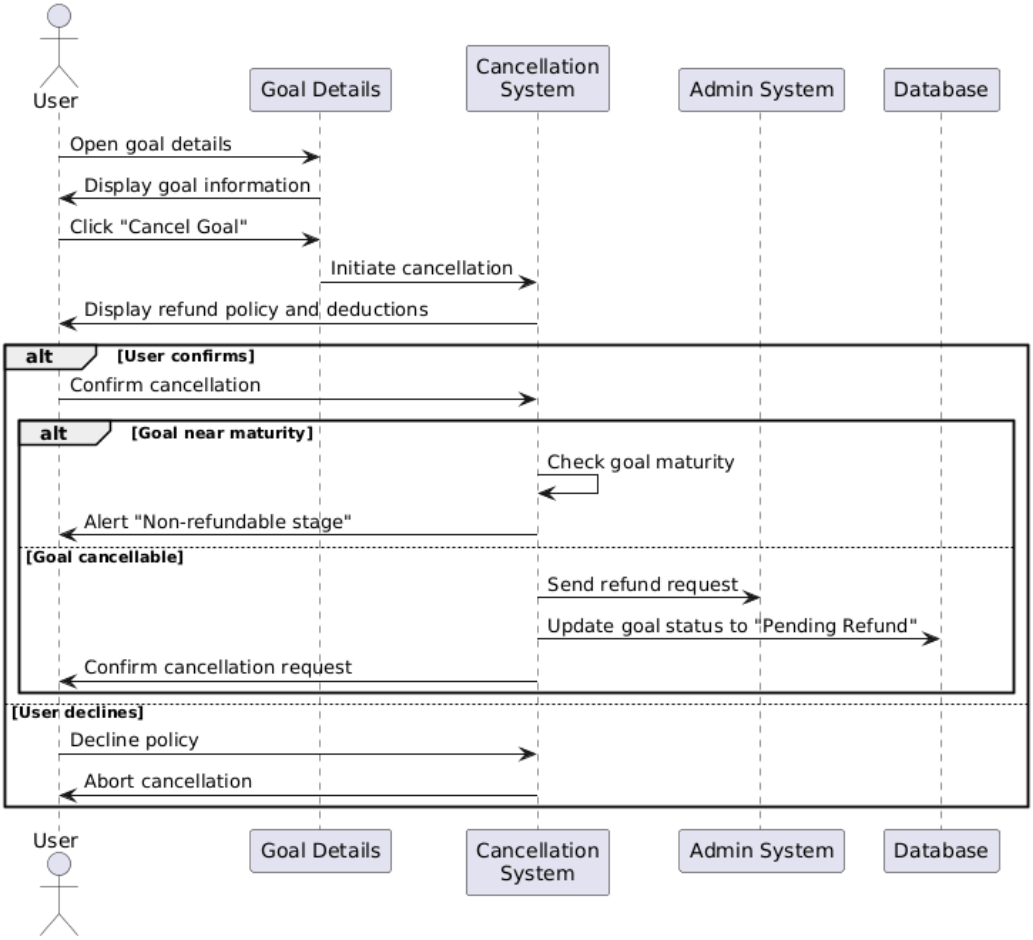


Figure 96: Request Goal Cancellation & Refund - Sequence Diagram

8.5.7 Redeem Product After Goal Completion

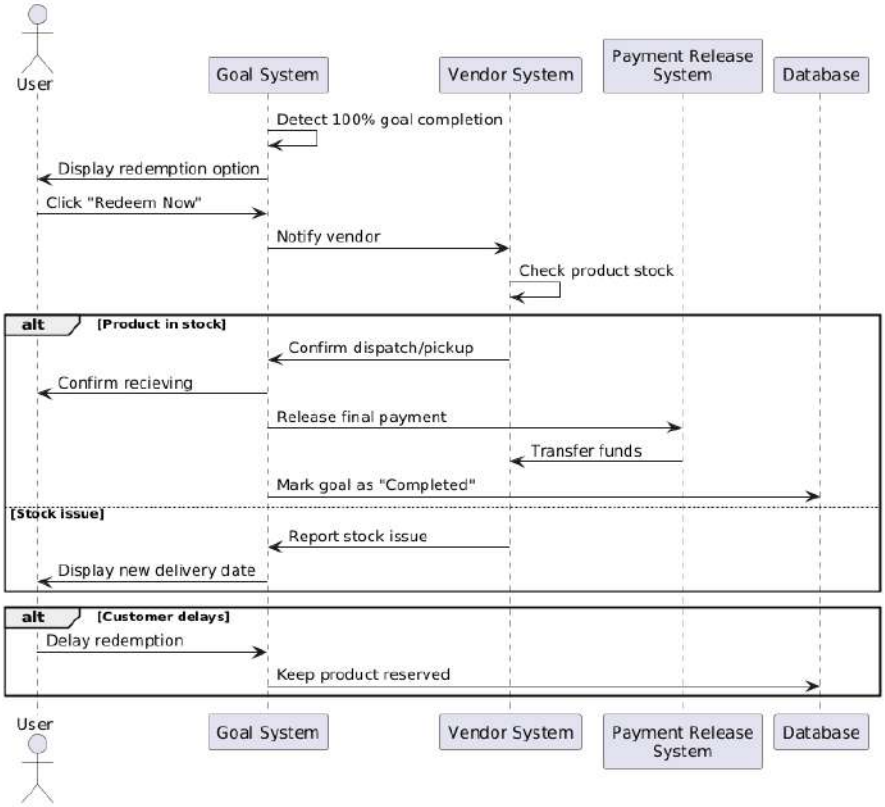


Figure 97: Redeem Product After Goal Completion - Sequence Diagram

8.5.8 Receive Notifications

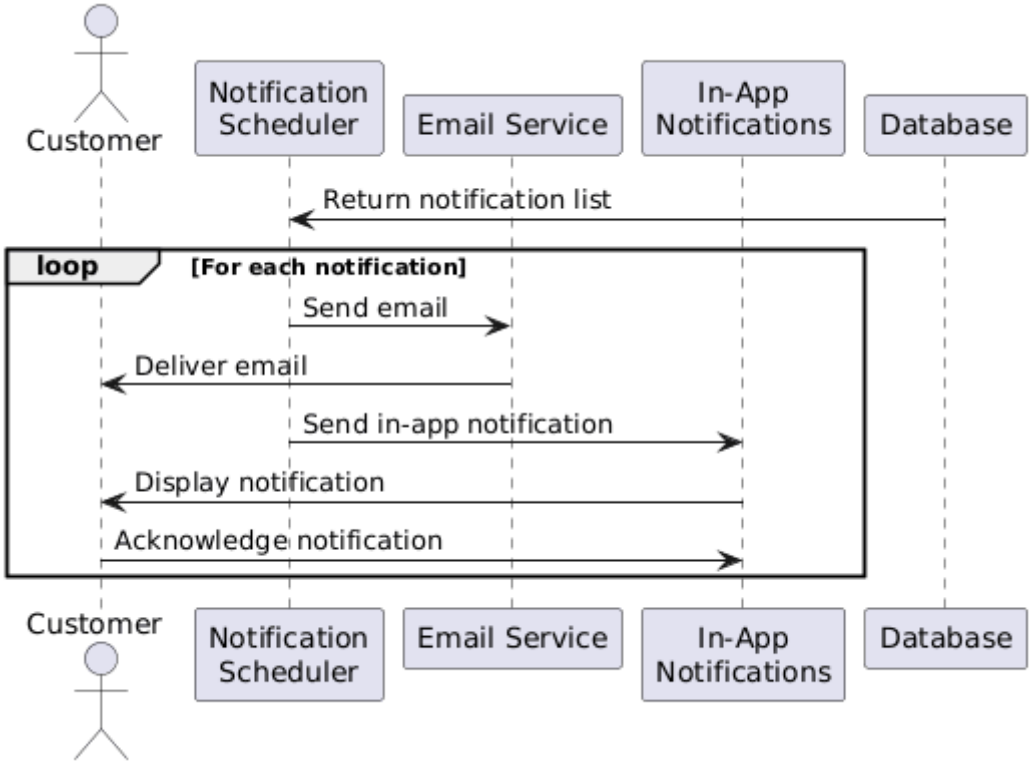


Figure 98: Receive Notifications and Reminders - Sequence Diagram

8.5.9 Store Registration

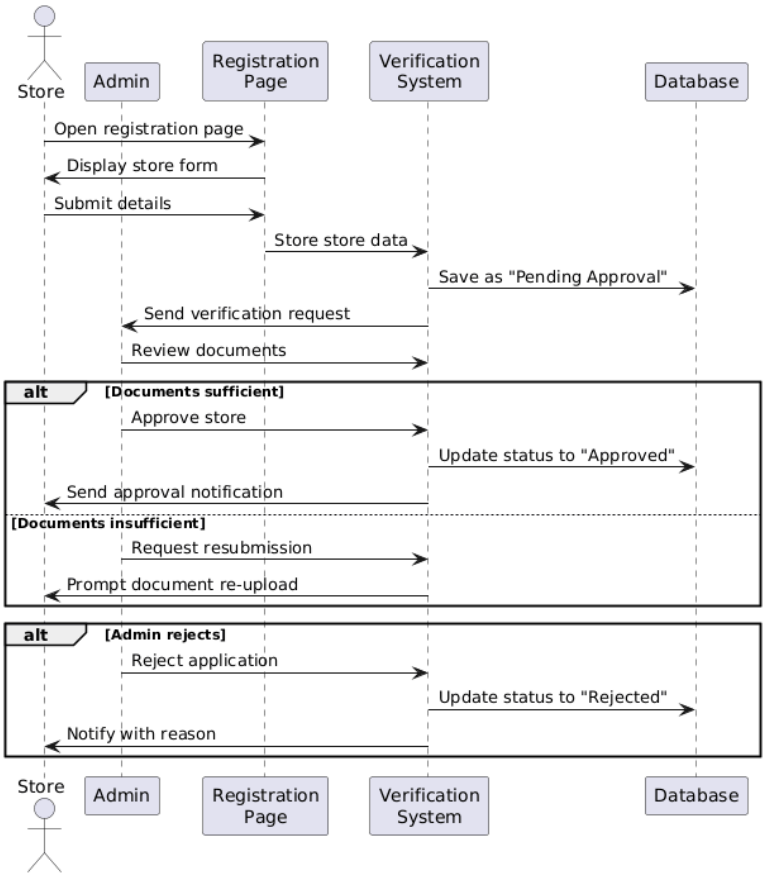


Figure 99: Store Registration - Sequence Diagram

8.5.10 Product Listing by Store

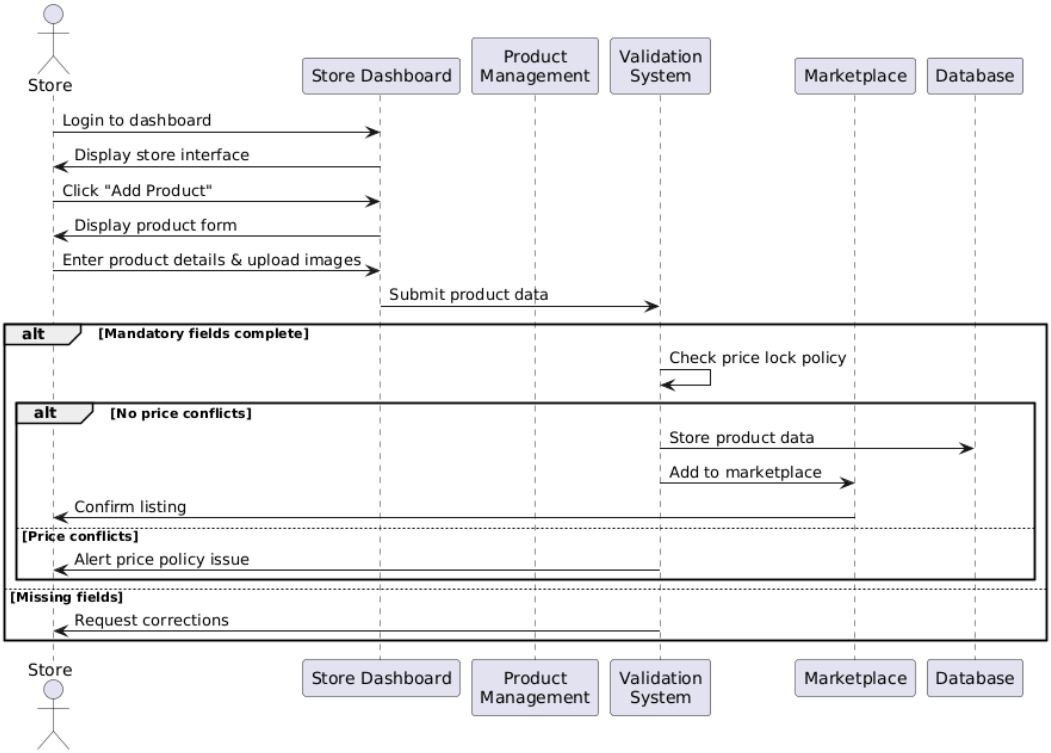


Figure 100: Product Listing by Store - Sequence Diagram

8.5.11 Lock Product for Layaway

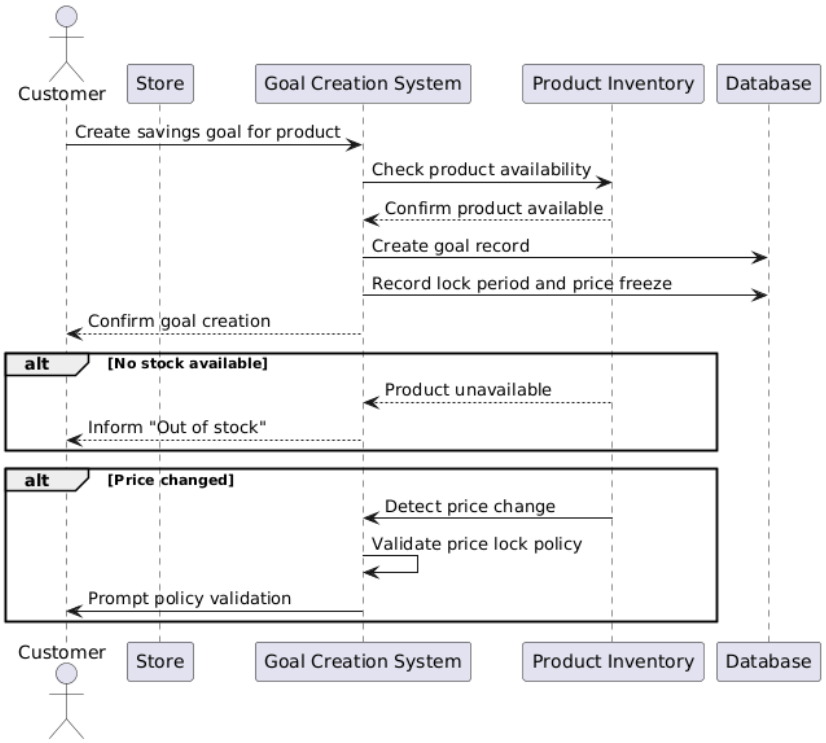


Figure 101: Lock Product for Layaway - Sequence Diagram

8.5.12 Confirm Delivery & Final Payment Release

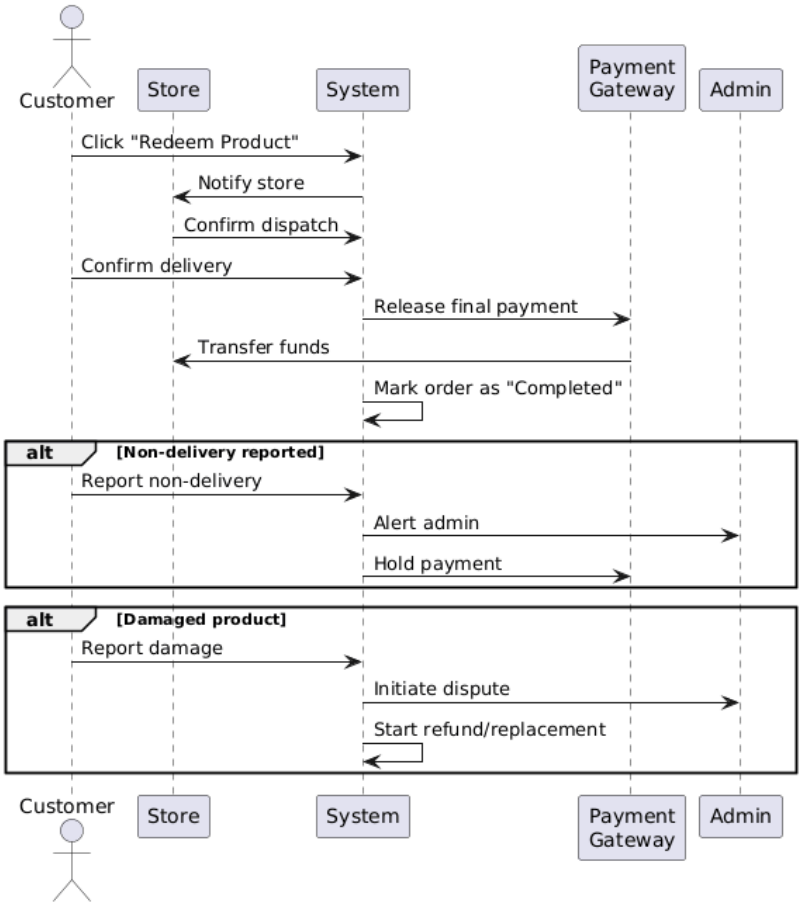


Figure 102: Confirm Delivery & Final Payment Release - Sequence Diagram

8.5.13 View Store Dashboard & Earnings

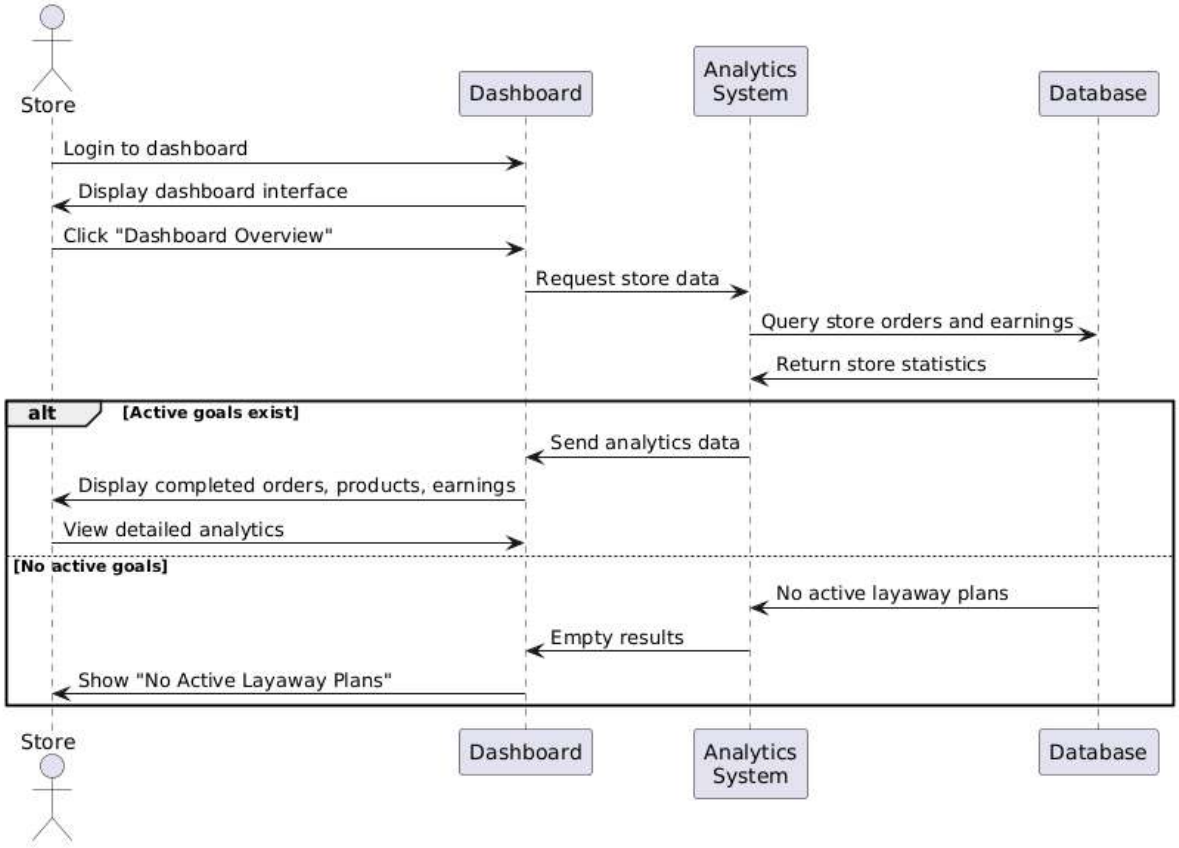


Figure 103: View Store Dashboard & Earnings - Sequence Diagram

8.5.14 Approve or Reject Store Applications

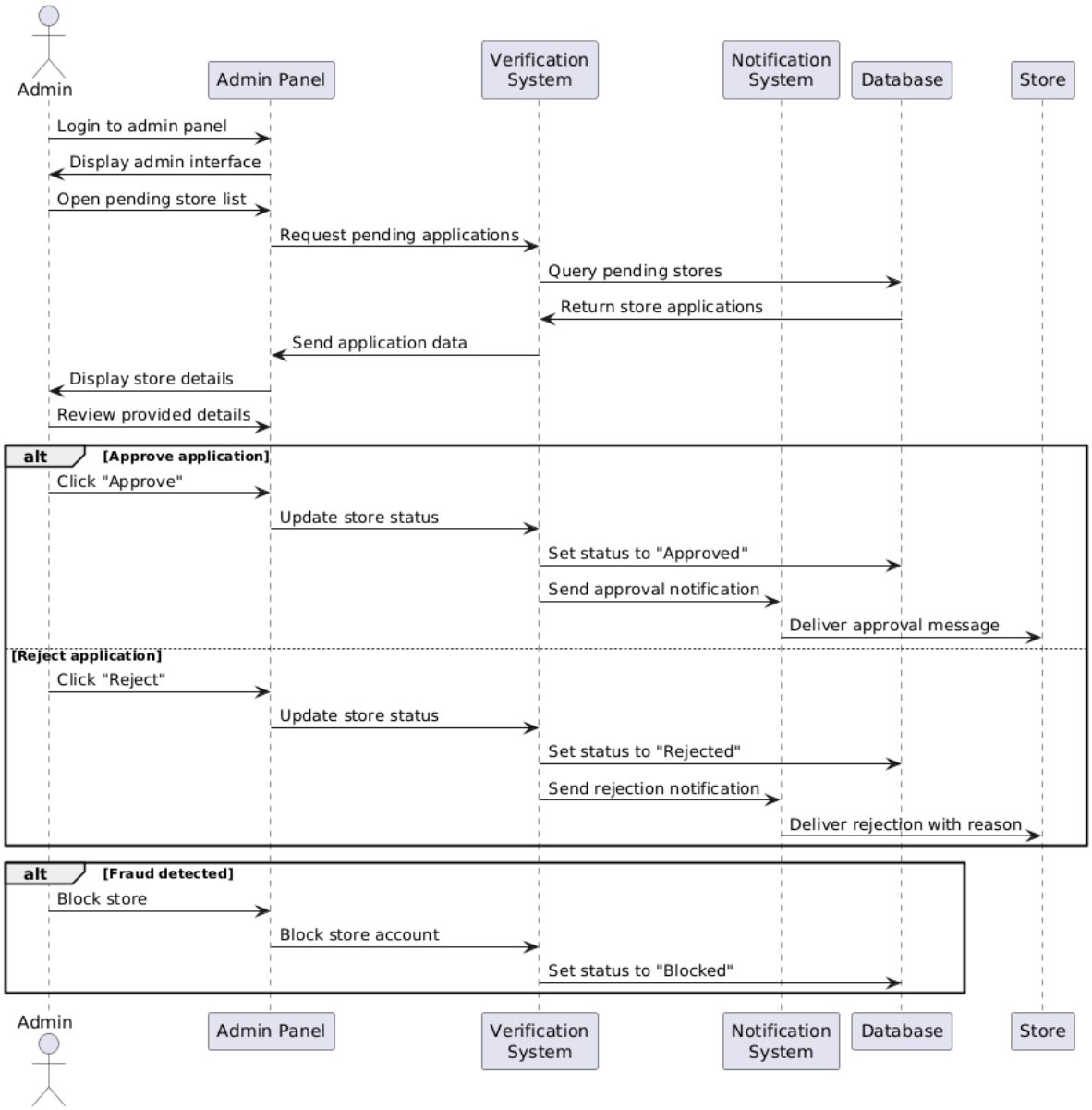


Figure 104: Approve or Reject Store Applications - Sequence Diagram

8.5.15 Monitor Transactions & Resolve Disputes

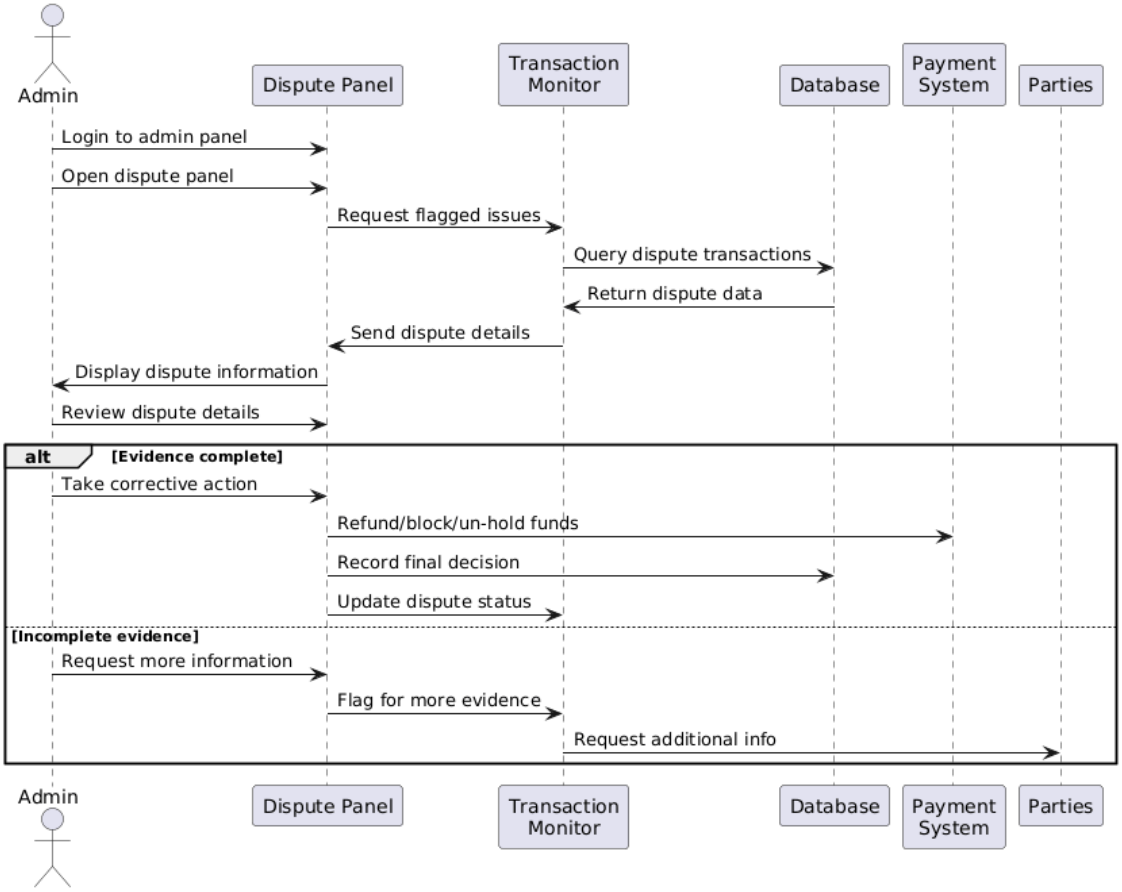


Figure 105: Monitor Transactions & Resolve Disputes - Sequence Diagram

8.5.16 Manage Price Lock & Inflation Policy

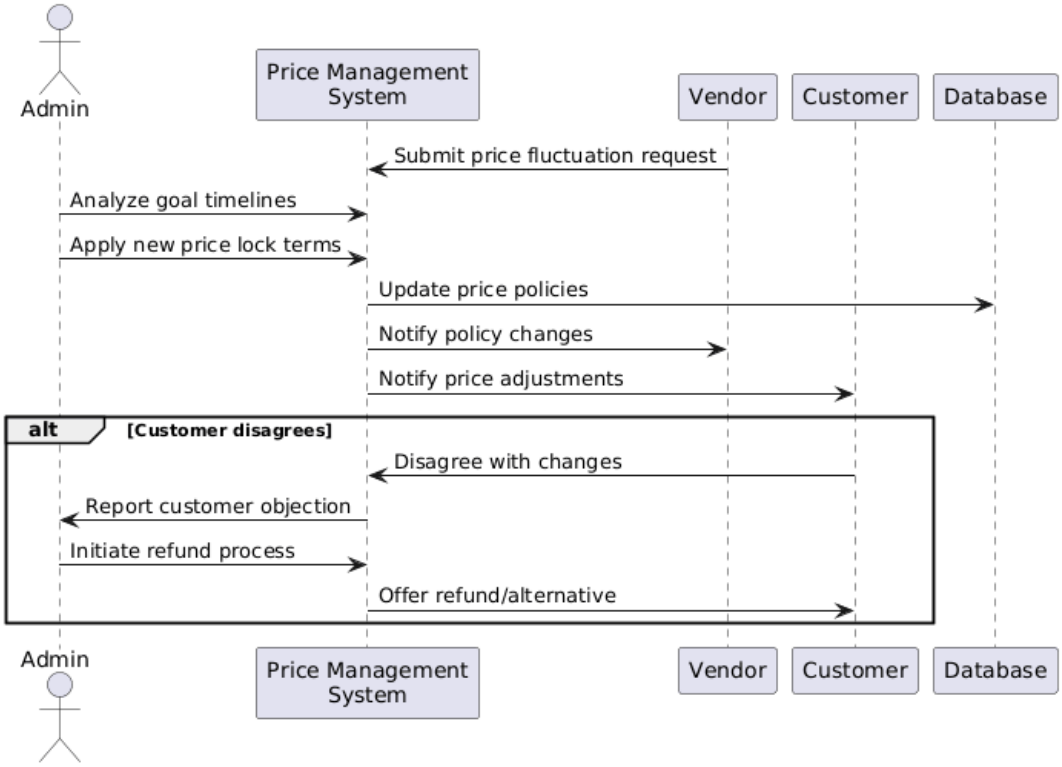


Figure 106: Manage Price Lock & Inflation Policy - Sequence Diagram

8.5.17 Generate System Reports & Audit Logs

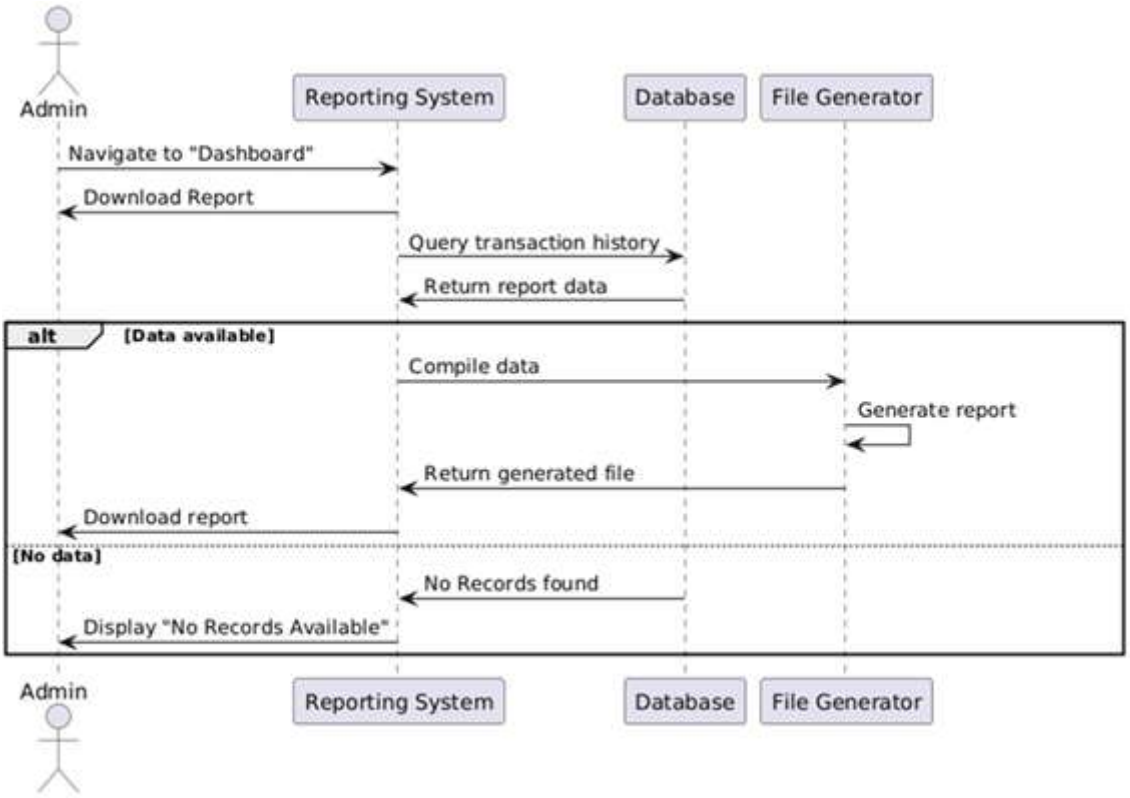


Figure 107: Generate System Reports & Audit Logs - Sequence Diagram

8.5.18 Automated Notifications

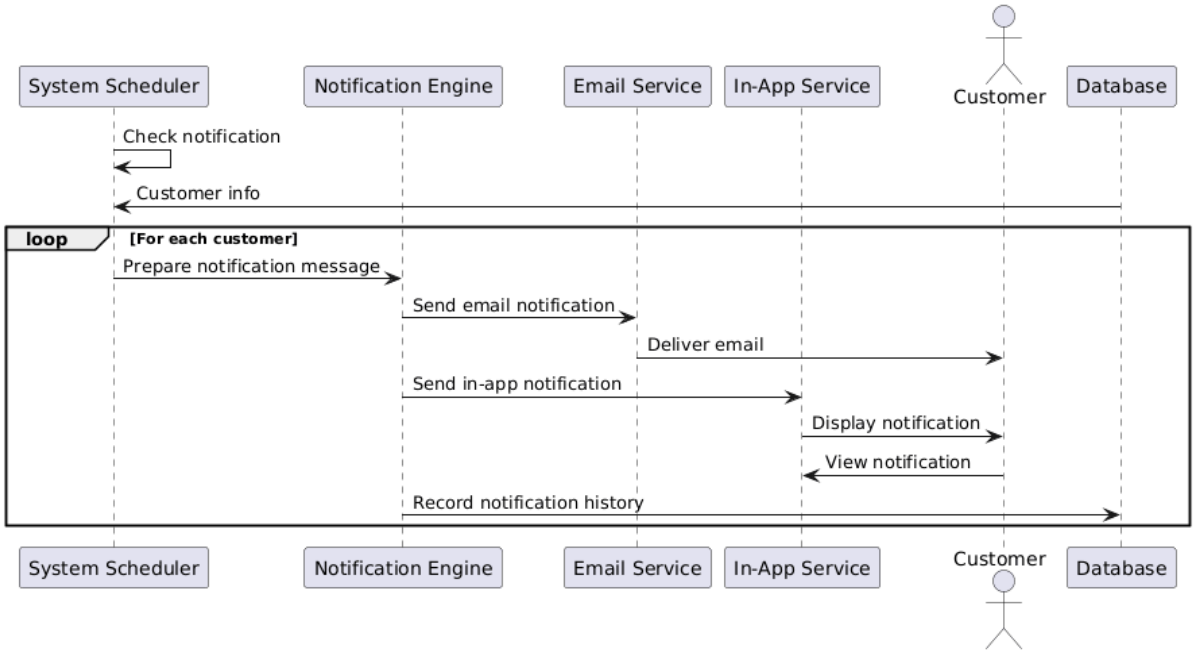


Figure 108: Automated Notifications - Sequence Diagram

8.5.19 Track Delivery via Map

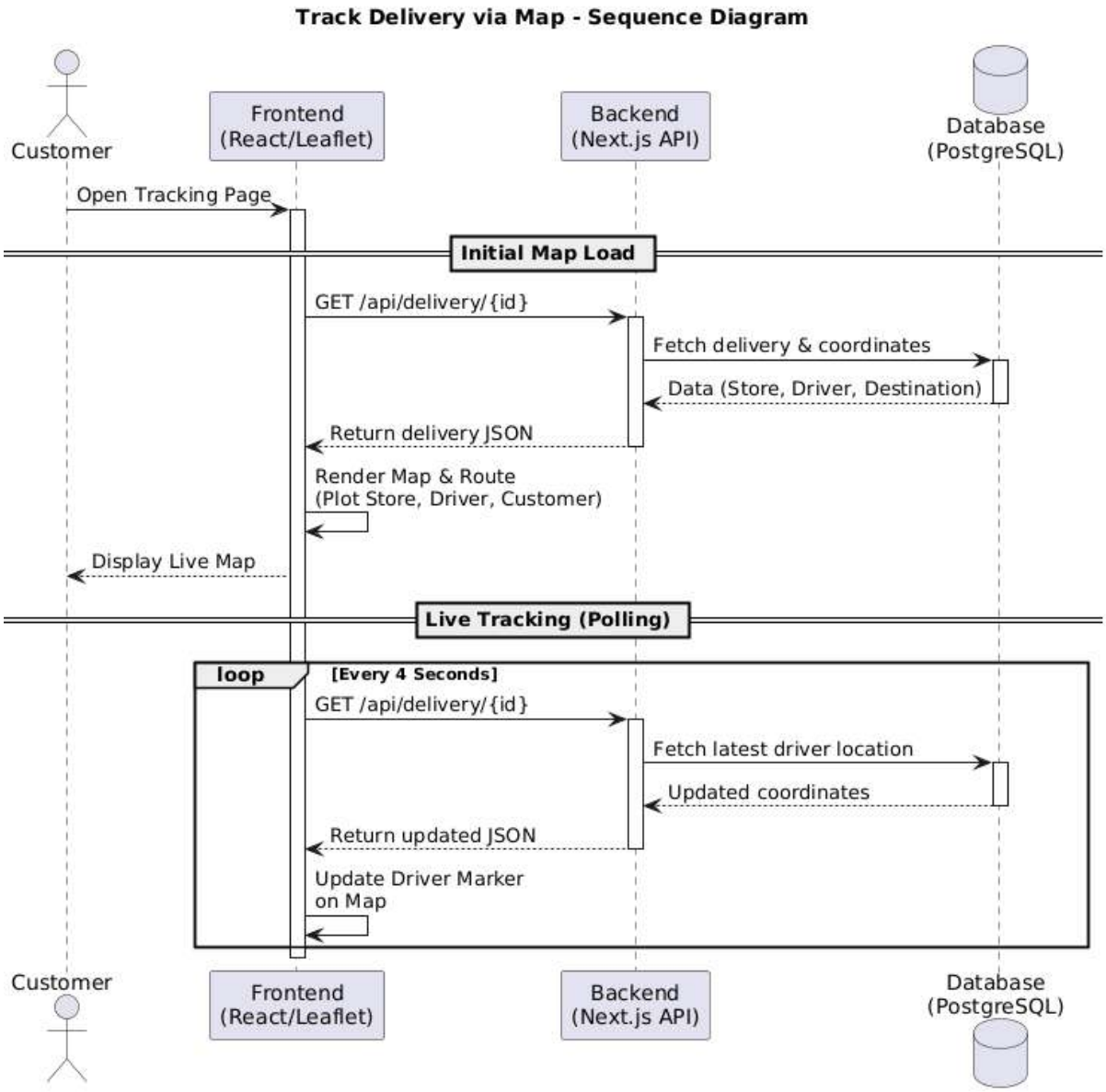


Figure 109: Track Delivery via Map - Sequence Diagram

8.5.20 Rider Delivery Execution

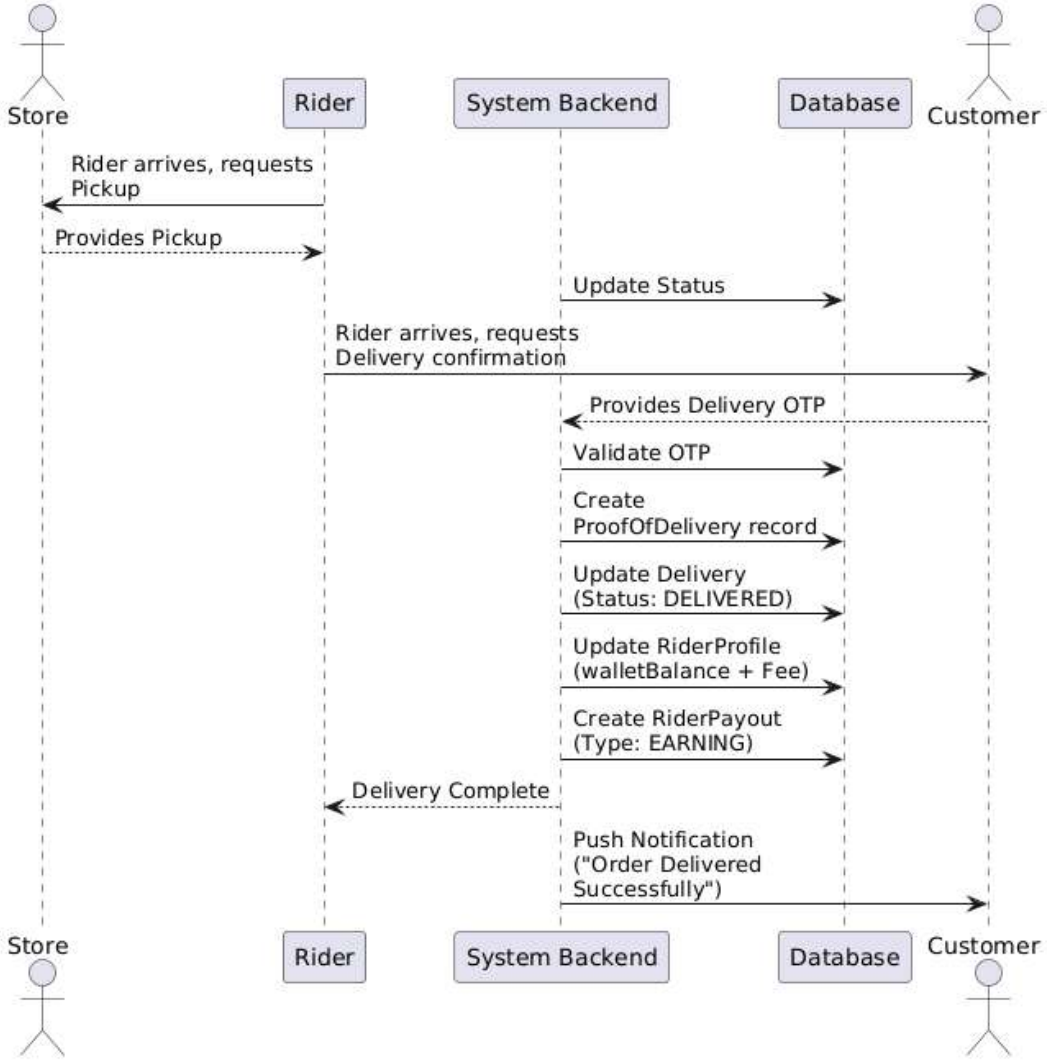


Figure 110: Rider Delivery Execution - Sequence Diagram

8.6 Collaboration Diagram

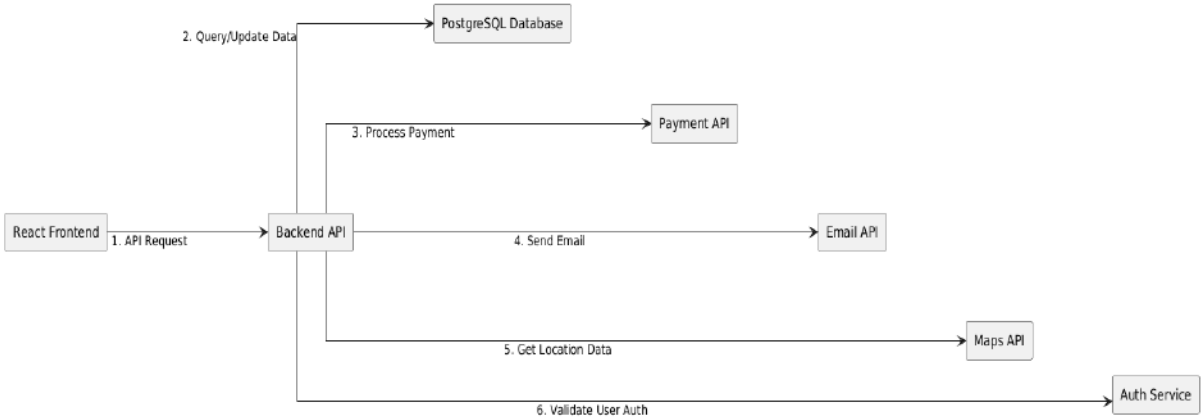


Figure 111: Collaboration Diagram

8.7 Use Case Diagram

8.7.1 Store Use Case Diagram

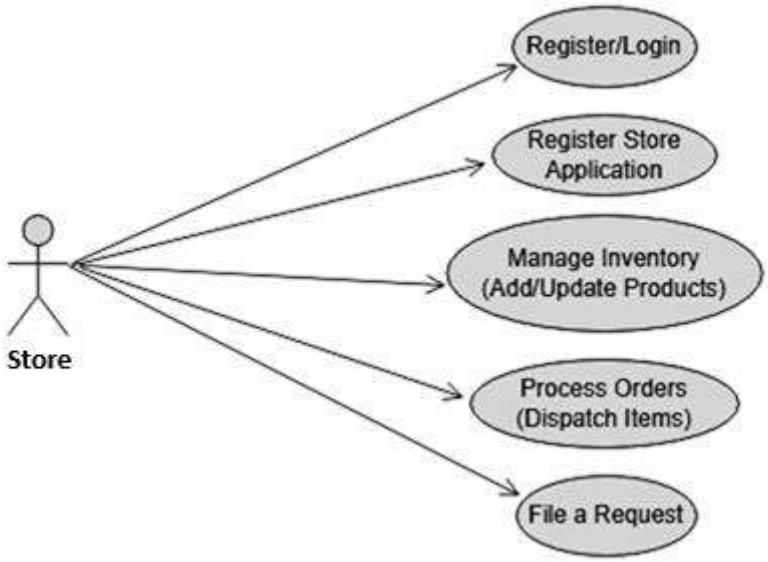


Figure 112: Store Use Case Diagram

8.7.2 Admin Use Case Diagram

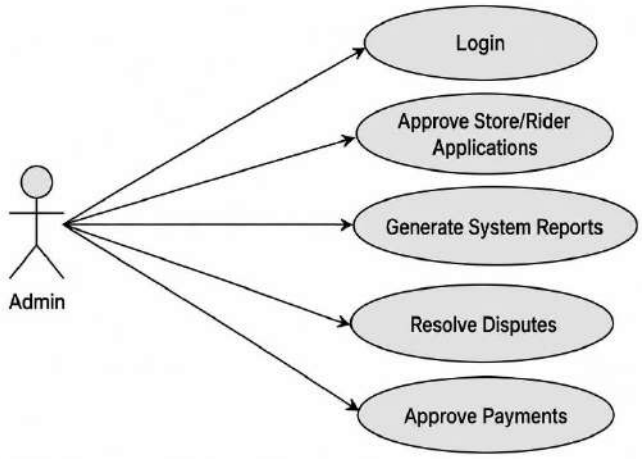


Figure 113: Admin Use Case Diagram

8.7.3 Rider Use Case Diagram

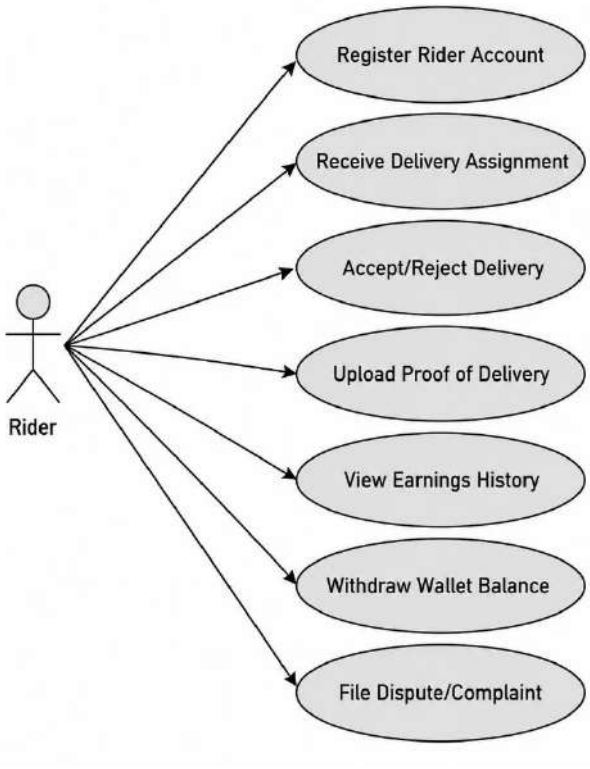


Figure 114: Rider Use Case Diagram

8.7.4 User Use Case Diagram

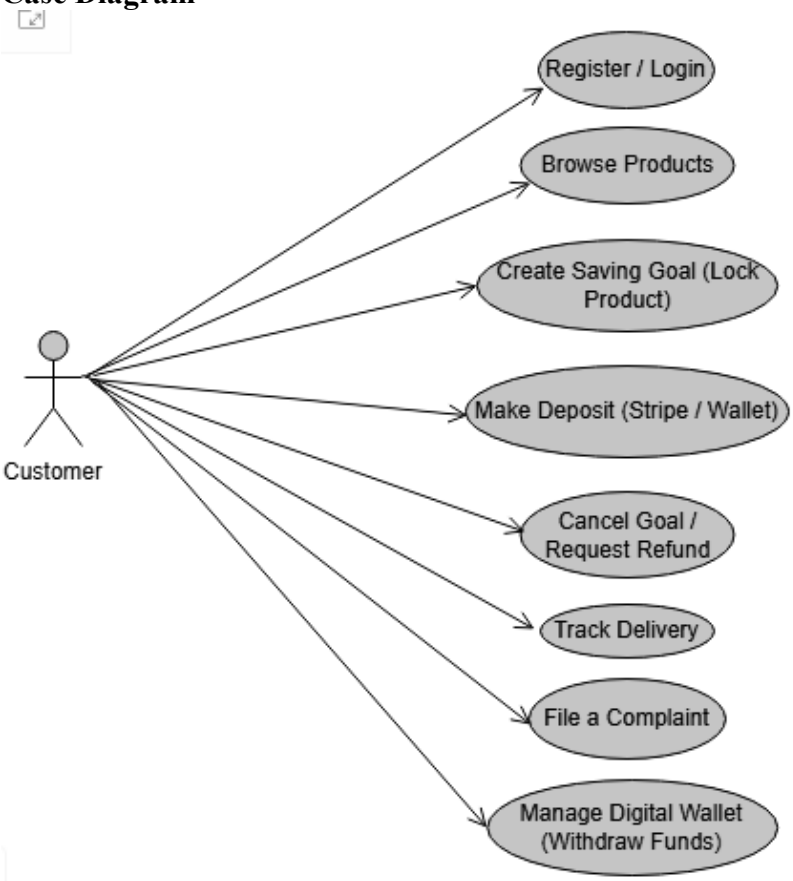


Figure 115: User Use Case Diagram

8.8 Component Diagram

8.8.1 Admin Component Diagram

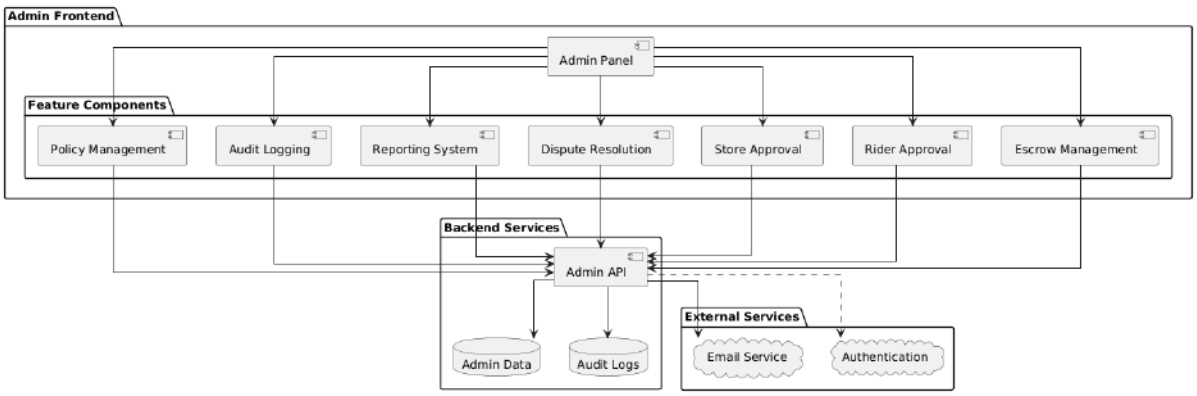


Figure 116: Admin Component Diagram

8.8.2 Store Component Diagram

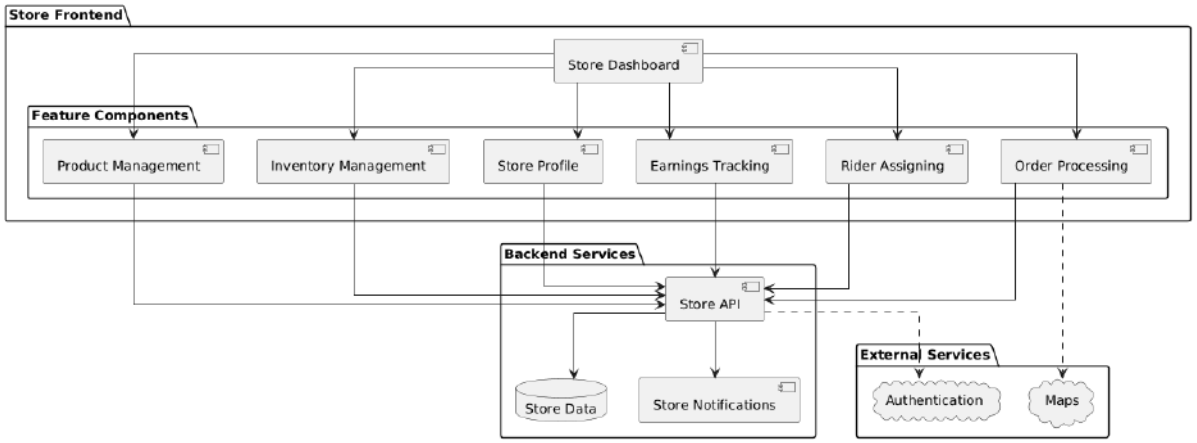


Figure 117: Store Component Diagram

8.8.3 User Component Diagram

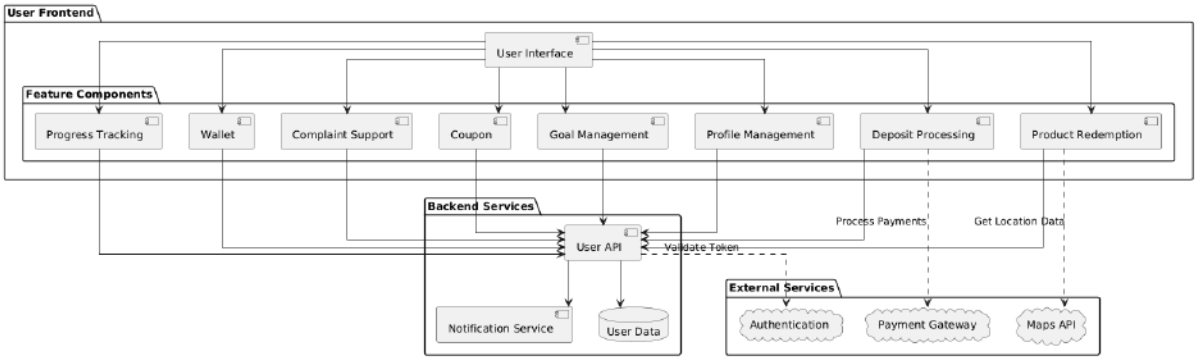


Figure 118: User Component Diagram

8.8.4 Rider Component Diagram

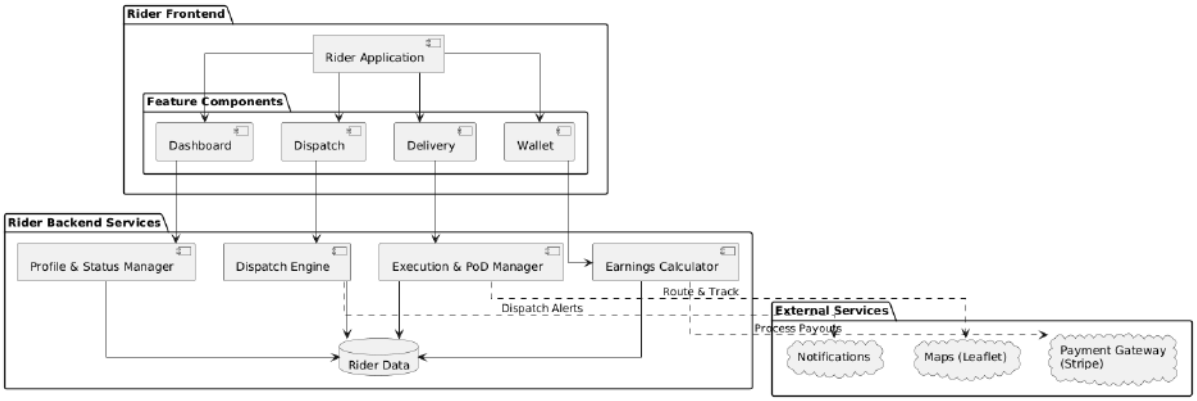


Figure 119: Rider Component Diagram

8.9 Deployment Diagram

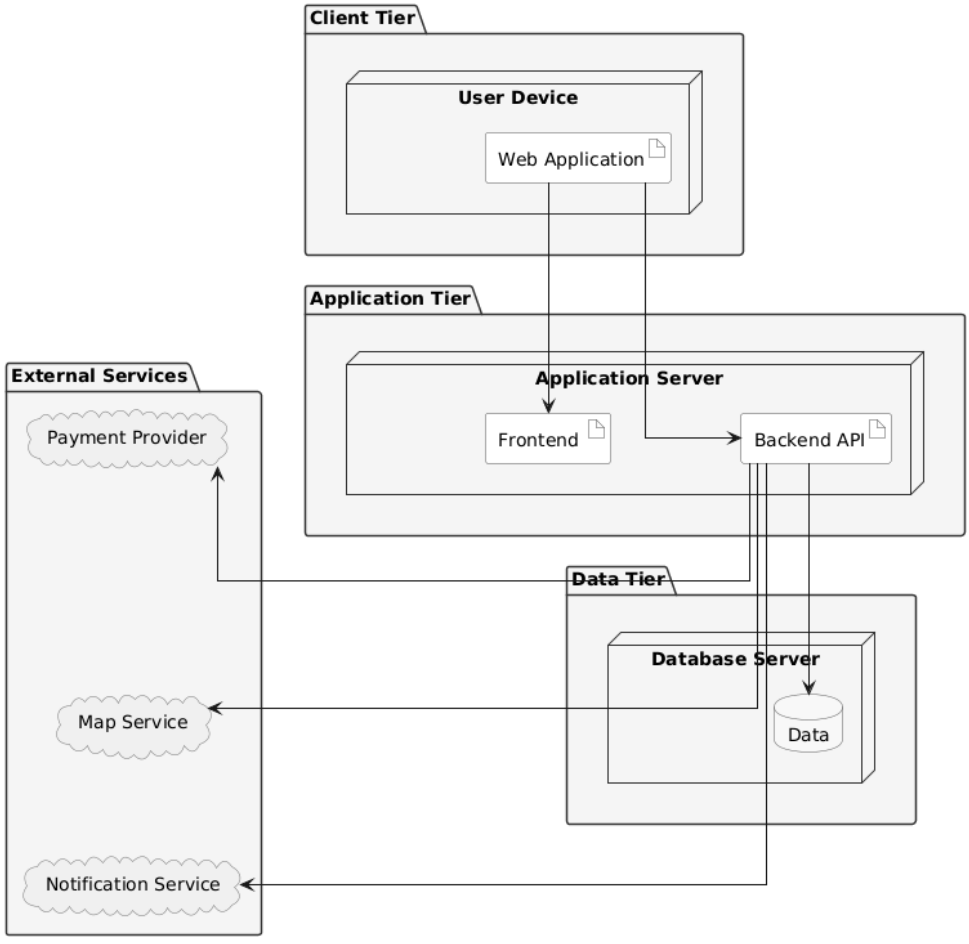


Figure 120: Deployment Diagram

8.10 System Block Diagram

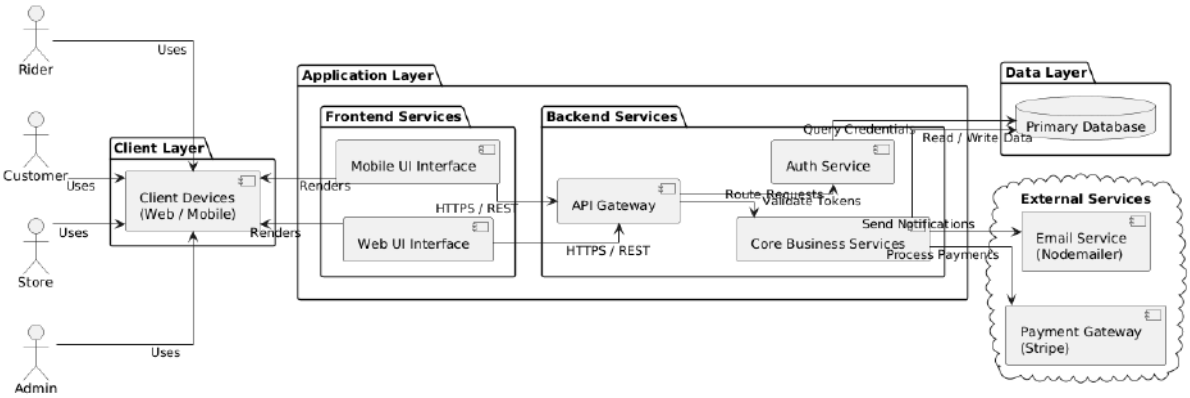


Figure 121: System Block Diagram

Test Cases

1. User Registration with Valid Data

Test Case ID	**P2-01-01**				
Test Case Name	User Registration with Valid Data	**Test Case Description**	User registers on the system, fills in valid personal data and confirms the account by email.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 Dec 2025

S. No	**Prerequisites:**
1	User is not registered.
2	User's email is valid.

Initial Execution

Test Scenario	The user registers on the system by filling in the necessary personal information, and then checks the personal information through the email verification code.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens the application.	Homepage loads successfully.	Loaded	Pass
2	User clicks on Sign up page.	Sign up page displayed.	Displayed	Pass
3	User provides a valid name and email.	Inputs accepted.	Accepted	Pass
4	User types in password and system responds "password is strong".	Passwords validated.	Validated	Pass
5	User enters Registration information.	An email with a verification code was sent to you to verify your address.	Sent	Pass
6	User clicks on verification email.	Verification code visible.	Visible	Pass
7	User types verification code.	Account activated.	Activated	Pass

Table 59: User Registration with Valid Data – Test Case

2. User Registration with Duplicate Email

Test Case ID	**P2-01-02**				
Test Case Name	User Registration with Duplicate Email	**Test Case Description**	Ensure that the system does not allow user to register using a duplicate email address and shows a suitable validation message.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 Dec 2025

S. No	Prerequisites:
1	Email already exists in system.

Initial Execution

Test Scenario	User tries to sign up with an email address that is already registered.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User clicks on Sign up page.	Page displayed.	Displayed	Pass
2	User enters name.	Accepted.	Accepted	Pass
3	User enters duplicate email.	System validates email.	Validated	Pass
4	User enters password.	Accepted.	Accepted	Pass
5	User submits form.	Error message shown.	No error shown.	Fail
6	User waits for response.	Registration blocked.	Not blocked.	Fail

Table 60: User Registration with Duplicate Email – Test Case

Re-Test Execution:

Test Scenario	User tries to sign up with an email address that is already registered.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User clicks on Sign up page.	Page displayed.	Displayed	Pass
2	User enters name.	Accepted.	Accepted	Pass
3	User enters duplicate email.	The message "email already exists" is shown.	Displayed	Pass
4	User edits e-mail field.	Field editable.	Editable	Pass
5	User types new email.	Accepted.	Accepted	Pass
6	User submits form	Verification email sent.	Sent	Pass
7	User verifies email	Account created	Created	Pass

Notes:

- Duplication in emails is now fixed.

3. User Login with Valid Credentials

Test Case ID	P2-01-03				
Test Case Name	User Login with Valid Credentials	**Test Case Description**	Confirm a registered and verified user can log in with a valid user ID and password, and gain access to the dashboard.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 Dec 2025

S. No	Prerequisites:
1	User account exists
2	Account is verified

Initial Execution

Test Scenario	Registered user logs in using valid credentials.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User logs in to the system.	Login screen displayed.	Displayed	Pass
2	User types in registered email.	Email accepted.	Accepted	Pass
3	User types in the correct password.	Password accepted.	Password accepted	Pass
4	User clicks on Login button.	Authentication successful.	Accepted	Pass
5	System redirects user.	Dashboard displayed.	No error shown.	Pass

Table 61: User Login with Valid Credentials – Test Case

4. User Login with Invalid Password

Test Case ID	P2-01-04				
Test Case Name	User Login with Invalid Password	**Test Case Description**	Confirm that the system displays an error message and denies login when a user logs in with the wrong password.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 Dec 2025

S. No	Prerequisites:
1	User account exists

Initial Execution

Test Scenario	Registered user logs in using valid credentials.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens login screen.	Login screen displayed.	Displayed	Pass
2	User types in valid email.	Email accepted.	Accepted	Pass
3	User enters a wrong password.	Password accepted.	Password accepted	Pass
4	User clicks Login.	Error message shown	No error message	Fail
5	User is stuck in the login screen.	Login blocked	Blocked	Pass
6	User retries to login	Allowed	Allowed	Pass

Table 62: User Login with Invalid Password – Test Case

Re-Test Execution:

Test Scenario	Registered user logs in using valid credentials.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens login screen.	Login screen displayed.	Displayed	Pass
2	User types valid email.	Email accepted.	Accepted	Pass
3	User enters a wrong password.	Error displayed.	Displayed	Pass
4	System displays error messages.	Invalid credentials.	Shown	Pass
5	User re enters password	Input accepted.	Accepted	Pass
6	User enters correct password.	Authenticated.	Authenticated	Pass
7	User redirected to home screen.	Screen visible.	Visible	Pass

Notes:

- Login validation logic was not properly handling errors if the password was incorrect. The error messages and authentication checks were fix during re-test.

5. Create Savings Goal

Test Case ID	**P2-01-05**		
Test Case Name	Create Savings Goal	**Test Case Description**	Ensure that a user with a logon account is able to set up a savings goal by selecting a product and confirming goal details.

Created By	Naqash Sachwani	Version	1.1	Date	15 Dec 2025

S. No	Prerequisites:
1	User is logged in

Initial Execution

Test Scenario	User sets up a new savings goal by choosing a product and inputting savings details.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens application.	Home screen displayed.	Displayed	Pass
2	User chooses a product.	Product selected.	Selected	Pass
3	User enters target amount.	Amount validated.	Validated	Pass
4	User chooses the length of time that they want to save.	Duration accepted.	Accepted	Pass
5	User review's goal details.	Details displayed.	Displayed	Pass
6	User verifies creation of goal.	Goal created.	Created	Pass
7	Goal is visible in the dashboard.	Goal listed.	Listed	Pass

Table 63: Create Savings Goal – Test Case

6. Store Registration with Valid Details

Test Case ID	P2-01-06				
Test Case Name	Store Registration with Valid Details	**Test Case Description**	Ensure that a Store can be registered with valid information.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	16 Dec 2025

S. No	Prerequisites:
1	Store is not registered

Initial Execution

Test Scenario	Store registers on the platform by providing valid details.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Store opens registration form.	Page displayed.	Displayed	Pass
2	Store enters details.	Inputs accepted.	Accepted	Pass

3	Store submits registration form.	Request submitted.	Submitted	Pass
4	System validates information	Validation successful.	Successful	Pass
5	Admin notified for approval	Notification sent.	Sent	Pass

Table 64: Store Registration with Valid Details – Test Case

7. Product Listing by Store

Test Case ID	**P2-01-07**				
Test Case Name	Product Listing by Store	**Test Case Description**	Make sure an approved Store is able to successfully add a product to their list.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**
1	Account is approved

Initial Execution

Test Scenario	Store newly adds a product to the platform.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Store logs in.	Dashboard displayed.	Displayed	Pass
2	Store opens product listing page.	Page opened.	Opened	Pass
3	Store enters product details.	Details accepted.	Accepted	Pass
4	Store uploads product images.	Images uploaded.	Uploaded	Pass
5	Product price is set by Store.	Price validated.	Validated	Pass
6	Store submits product.	Product listed.	Listed	Pass

Table 65: Product Listing by Store – Test Case

8. Product Listing without Price

Test Case ID	**P2-01-08**				
Test Case Name	Product Listing without Price	**Test Case Description**	Ensure that the system does not permit products to be listed without a price.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**

1	Tries to sell a product without a price.

Initial Execution

Test Scenario	Store introduces a new product on the platform.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Store logs in.	Dashboard displayed.	Displayed	Pass
2	Store opens product listing page.	Page opened.	Opened	Pass
3	Store writes product information.	Details accepted.	Accepted	Pass
4	Store skips price field.	Validation error shown.	No error	Fail
5	Store submits product.	Submission blocked.	Allowed	Fail
6	Product appears in listing.	Should not appear.	Appeared	Fail

Table 66: Product Listing without Price – Test Case

Re-Test Execution

Test Scenario	Store introduces a new product on the platform.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Store logs in.	Dashboard displayed.	Displayed	Pass
2	Store opens product listing page.	Page opened.	Opened	Pass
3	Store writes product information.	Details accepted.	Accepted	Pass
4	Store submits without price.	Error message shown.	Shown	Pass
5	System blocks submission.	Blocked.	Blocked	Pass
6	Store enters valid price.	Price accepted.	Accepted	Pass
7	Store resubmits product.	Submitted.	Submitted	Pass
8	Product listed successfully.	Listed.	Displayed	Pass

Notes:

- Price field validation missing when listing products. Validation rules added to make sure that product details are not published if they are incomplete.

9. Price Lock on Goal Creation

Test Case ID	**P2-01-09**				
Test Case Name	Price Lock on Goal Creation	**Test Case Description**	Ensure that product price is fixed when a savings goal is set.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**
1	User is logged in.
2	Product is available.

Initial Execution

Test Scenario	User sets a savings target and confirms that the product price is constant for the period of the savings target.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard.	Dashboard displayed.	Displayed	Pass
2	User selects a product.	Product details shown.	Shown	Pass
3	User notes product price.	Price visible.	Visible	Pass
4	User creates savings goal.	Goal created.	Created	Pass
5	User revisit's goal details.	Price unchanged.	Unchanged	Pass
6	User refreshes page.	Price persists.	Persisted	Pass

Table 67: Price Lock on Goal Creation – Test Case

10. Price Change After Goal Creation

Test Case ID	**P2-01-10**				
Test Case Name	Price Change After Goal Creation	**Test Case Description**	Check if an admin price change will impact on current goals.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**
1	User has an active goal.
2	Admin updates product price.

Initial Execution

Test Scenario	Admin changes product price after goal is created; user notices effect on goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin changes the product price.	New price saved.	Saved	Pass
2	User opens dashboard	Dashboard displayed.	Displayed	Pass
3	User opens active goal.	Goal details shown.	Shown	Pass
4	User checks product price.	Old Price must be displayed.	New price shown	Fail
5	User refreshes page.	Price remains locked.	Still changed	Fail
6	System recalculates goal.	Should not recalculate.	Recalculated	Fail

Table 68: Price Change After Goal Creation – Test Case

Re-Test Execution

Test Scenario	Admin changes product price after goal is created; user notices effect on goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin updates product price	New price saved.	Saved	Pass
2	User opens dashboard	Dashboard displayed.	Displayed	Pass
3	User opens active goal.	Goal details shown.	Shown	Pass
4	System shows locked price	Original price shown.	Shown	Pass
5	User refreshes page.	Price unchanged.	Unchanged	Pass
6	Goal calculations verified.	No change.	Verified	Pass

Notes:

- Corrected price locking logic.

11. Store Dashboard View

Test Case ID	**P2-01-11**		
Test Case Name	Store Dashboard View	**Test Case Description**	Ensure Store can get dashboard statistics right.

Created By	Naveed Asad Ali	Version	1.1	Date	16 Dec 2025

S. No	Prerequisites:
1	Store account is approved

Initial Execution

Test Scenario	Store Logs in Dashboard and View Dashboard metrics.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Store logs in.	Dashboard displayed.	Displayed	Pass
2	Store listes products.	Products shown.	Shown	Pass
3	Store views earnings summary.	Earnings visible.	Visible	Pass
4	Store views orders count.	Order count correct.	Correct	Pass
5	Store refreshes dashboard.	Data persists.	Persisted	Pass
6	Store logs out.	Logout successful.	Successful	Pass

Table 69: Store Dashboard View – Test Case

12. Admin Login

Test Case ID	P2-01-12				
Test Case Name	Admin Login	Test Case Description	Check that admin can log in using proper credentials.		
Created By	Naqash Sachwani	Version	1.1	Date	16 Dec 2025

S. No	Prerequisites:
1	Admin account exists.

Initial Execution

Test Scenario	The administrator logs into the system to gain administrative access.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin opens login page.	Login page displayed.	Displayed	Pass
2	Admin enters username.	Accepted.	Accepted	Pass
3	Admin enters password.	Accepted.	Accepted	Pass

4	Admin clicks Login.	Authenticated.	Authenticated	Pass
5	Admin dashboard loads.	Dashboard visible,	Visible	Pass
6	Admin views admin menu.	Menu accessible.	Accessible	Pass

Table 70: Admin Login – *Test Case*

13. Admin Approves Store Registration

Test Case ID	**P2-01-13**				
Test Case Name	Admin Approves Store Registration	**Test Case Description**	Ensure that admin is able to successfully approve a pending Store registration.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**
1	Admin is logged in.
2	Registration request is pending.

Initial Execution

Test Scenario	Admin checks Store details and grants the Store registration.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin opens dashboard.	Dashboard displayed.	Displayed	Pass
2	Admin clicks on Store requests.	Pending requests shown.	Shown	Pass
3	Admin selects Store request.	Store details displayed.	Displayed	Pass
4	Admin reviews details.	Details verified.	Verified	Pass
5	Admin clicks "Approve"	Store approved.	Approved	Pass
6	System updates – Store status.	The status will be set to Approved.	Updated	Pass
7	System sends notification.	Store notified.	Notified	Pass

Table 71: Admin Approves Store Registration – Test Case

14. User Logout Functionality

Test Case ID	**P2-01-14**		
Test Case Name	User Logout Functionality	**Test Case Description**	Ensure that user can successfully log out and session closed.

Created By	Naveed Asad Ali	Version	1.1	Date	16 Dec 2025

S. No	Prerequisites:
1	User is logged in.

Initial Execution

Test Scenario	User logs out from the application and tries to access the secured pages.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens home screen.	Home screen displayed	Displayed	Pass
2	User clicks on profile menu	Menu opens	Opened	Pass
3	User clicks Logout	Logout initiated	Initiated	Pass
4	System clears session	Session cleared	Cleared	Pass
5	User(s) redirected to login page	Login page shown	Shown	Pass
6	User clicks back button	Access denied	Denied	Pass
7	User refreshes page	Still logged out	Logged out	Pass
8	User clicks Logout	Logout initiated	Initiated	Pass

Table 72: User Logout Functionality – *Test Case*

15. Password Change Success

Test Case ID	P2-01-15				
Test Case Name	Password Change Success	**Test Case Description**	Check that user is able to change the password using the correct inputs.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	16 Dec 2025

S. No	Prerequisites:
1	User is logged in.

Initial Execution

Test Scenario	User changes password and logs in using the new password.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User accesses their profile settings.	Settings opened	Opened	Pass

2	User clicks on Change Password	Form displayed	Displayed	Pass
3	Enters current password	Validated	Validated	Pass
4	User enters new password	Accepted	Accepted	Pass
5	User confirms new password	Matched	Matched	Pass
6	User submits form	Password updated	Updated	Pass
7	System logs password change	Log created	Created	Pass
8	User logs out	Logged out	Logged out	Pass
9	User logs in with new password	Login successful	Successful	Pass

Table 73: Password Change Success – Test Case

16. Deposit Amount into Savings Goal

Test Case ID	**P2-01-16**				
Test Case Name	Deposit Amount into Savings Goal	**Test Case Description**	Check if a user is able to add funds to an active savings goal.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	18 Dec 2025

S. No	**Prerequisites:**
1	User is logged in.
2	An active savings goal exists.

Initial Execution

Test Scenario	User deposits funds into a pre-existing savings objective by a legitimate payment strategy.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens application.	Home screen displayed.	Displayed	Pass
2	User selects an active savings goal.	Goal shown.	Shown	Pass
3	User clicks the "Deposit" button.	Deposit form displayed.	Displayed	Pass
4	User types in amount of deposit.	Amount validated.	Validated	Pass
5	User chooses how to pay.	Payment option selected.	Selected	Pass
6	User confirms payment.	Payment processed successfully.	Processed	Pass

7	System update's goal balance	Updated balance displayed.	Displayed	Pass

Table 74: Deposit Amount into Savings Goal – Test Case

17. Track Savings Progress

Test Case ID	**P2-01-17**				
Test Case Name	Track Savings Progress	**Test Case Description**	Ensure that savings progress is accurately displayed for an active goal.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	18 Dec 2025

S. No	**Prerequisites:**
1	User is logged in.
2	Deposits has been made

Initial Execution

Test Scenario	User views progress details of an active savings goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens application.	Home screen displayed.	Displayed	Pass
2	User clicks on an active savings goal.	Goal shown.	Displayed	Pass
3	User views progress bar.	Progress shown correctly.	Shown	Pass
4	User checks deposited amount.	Amount accurate.	Accurate	Pass
5	User checks remaining amount.	Remaining amount shown.	Shown	Pass
6	User refreshes page	Data persists.	Persisted	Pass

Table 75: Track Savings Progress – *Test Case*

18. Cancel Savings Goal without Condition Acceptance

Test Case ID	**P2-01-18**				
Test Case Name	Cancel Savings Goal without Condition Acceptance	**Test Case Description**	Verify that the system does not allow cancellation of a savings goal unless the cancellation conditions is accepted.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	16 Dec 2025

S. No	**Prerequisites:**
1	User is logged in.

2	An active savings goal exists.

Initial Execution

Test Scenario	User attempts to delete an active savings goal without accepting the cancellation terms then retries after the bug is patched.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard	Dashboard displayed.	Displayed	Pass
2	The User selects an active savings goal.	The details page for the Goal was opened.	Opened	Pass
3	User click "Cancel Goal".	Condition popup should appear.	Popup not shown	Fail
4	User clicks confirm cancellation.	The cancellation should be disabled.	Cancellation allowed	Fail
5	System processes cancellation.	Goal should stay on.	Goal cancelled	Fail
6	System removes goal.	Status unchanged.	Status changed	Fail

Table 76: Cancel Savings Goal without Condition Acceptance – Test Case

Re-Test Execution

Test Scenario	User attempts to delete an active savings goal without accepting the cancellation terms then retries after the bug is patched.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard	Dashboard displayed.	Displayed	Pass
2	The User selects an active savings goal.	Details page for the goal was opened.	Opened	Pass
3	User click "Cancel Goal" again.	Policy popup displayed.	Displayed	Pass
4	User reads cancellation conditions.	Policy readable.	Readable	Pass
5	User accepts conditions, terms.	Acceptance recorded.	Recorded	Pass
6	User confirms cancellation.	Goal cancelled successfully.	Cancelled	Pass
7	System removes goal.	The status is now Cancelled.	Updated	Pass

Notes:

- All the conditions for cancellation are enforced correctly.

19. Redeem Product Before Goal Completion

Test Case ID	**P2-01-19**				
Test Case Name	Redeem Product Before Goal Completion	**Test Case Description**	Ensure system does not allow product redemption if savings goal is not met.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	20 Jan 2026

S. No	**Prerequisites:**
1	User is logged in.
2	Saving plan is set but not fulfilled.

Initial Execution

Test Scenario	User tries to redeem a product before reaching the savings goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard.	Dashboard displayed.	Displayed	Pass
2	User chooses active savings goal.	Goal details shown.	Shown	Pass
3	User clicks "Redeem".	Error message displayed.	No error shown	Fail
4	User confirms redemption.	Redemption blocked.	Allowed	Fail
5	System processes request.	Request rejected.	Processed	Fail
6	Goal status checked.	Should remain active.	Unchanged	Pass

Table 77: Redeem Product Before Goal Completion – Test Case

Re-Test Execution

Test Scenario	User tries to redeem a product before reaching the savings goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard.	Dashboard displayed.	Displayed	Pass
2	User chooses active savings goal.	Goal details shown.	Shown	Pass
3	User clicks "Redeem".	"Goal not completed" message.	Displayed	Pass
4	System blocks redemption.	Blocked.	Blocked	Pass

5	User closes message.	Message dismissed.	Dismissed	Pass
6	User returns to dashboard.	Dashboard shown.	Shown	Pass
7	Goal status verified.	Still active	Active	Pass

Notes:

- Eligibility for redemption was not checked correctly. Validation of completion status before redemption has been added.

20. Redeem Product After Goal Completion

Test Case ID	**P2-01-20**				
Test Case Name	Redeem Product after Goal Completion	**Test Case Description**	Confirm if a user can redeem a product since the user reached a savings target.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	20 Jan 2026

S. No	**Prerequisites:**
1	User is logged in.
2	Savings goal is completed

Initial Execution

Test Scenario	User redeems the product of a completed savings goal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User opens dashboard.	Dashboard displayed.	Displayed	Pass
2	User chooses completed savings goal.	Goal details displayed.	Displayed	Pass
3	User click Redeem button.	Redemption page opened.	Opened	Pass
4	User confirms redemption request.	Request submitted.	Submitted	Pass
5	System notifies Store.	Store notified.	Notified	Pass
6	System update's goal status.	Marked as Redeemed.	Updated	Pass
7	User is notified of confirmation.	Confirmation shown.	Shown	Pass

Table 78: Redeem Product after Goal Completion – Test Case

21. Monitor Escrow by Admin

Test Case ID	**P2-01-21**				
Test Case Name	Monitor Escrow by Admin	**Test Case Description**	Confirm that admin has access to and is able to track all transactions in the system.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	20 Jan 2026

S. No	**Prerequisites:**
1	Admin is logged in.

Initial Execution

Test Scenario	Admin reviews transaction records for monitoring purposes.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin opens dashboard.	Dashboard displayed.	Displayed	Pass
2	Admin clicks on escrow management.	Transaction list shown.	Shown	Pass
3	Admin filters transactions.	Filtered results displayed.	Displayed	Pass
4	Admin views details.	Details accurate.	Accurate	Pass
5	Admin exports escrow report.	Report generated.	Generated	Pass
6	Admin logs activity.	Action logged.	Logged.	Pass

Table 79: Monitor Escrow by Admin – Test Case

22. Dispute Resolution Failure

Test Case ID	**P2-01-22**				
Test Case Name	Dispute Resolution Failure	**Test Case Description**	Test system behavior in the event of a dispute.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	10 Feb 2026

S. No	**Prerequisites:**
1	Dispute exists in system.
2	Admin is logged in.

Initial Execution

Test Scenario	The transactions are monitored by Admin.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin creates complain section.	Dispute list shown.	Shown	Pass
2	Admin selects dispute.	Details displayed.	Displayed	Pass
3	Admin enters resolution.	Resolution accepted.	Accepted	Pass
4	Admin submits resolution.	Status updated.	Not updated	Fail
5	System logs resolution.	Log created.	Not logged	Fail
6	User notified.	Notification sent.	Not sent	Fail

Table 80: Dispute Resolution Failure – Test Case

Re-Test Execution

Test Scenario	The transactions are monitored by Admin.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin opens dispute section.	Dispute list shown.	Shown	Pass
2	Admin selects dispute.	Details displayed.	Displayed	Pass
3	Admin restates resolution.	Status updated.	Updated	Pass
4	System logs resolution	Log created.	Created	Pass
5	User notified	Notification sent.	Sent	Pass
6	Dispute marked resolved	Resolved.	Resolved	Pass
7	Admin refreshes page.	Status persists.	Persisted	Pass

Notes:

- Update of status/dispute failed. Fixed back-end transaction handling and notification triggers.

23. Refund Processing by Admin

Test Case ID	**P2-01-23**				
Test Case Name	Refund Processing by Admin	**Test Case Description**	Ensure that an admin can be successful in processing a refund request.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	10 Feb 2026

S. No	**Prerequisites:**
1	Refund request exists.
2	Admin is logged in.

Initial Execution

Test Scenario	Admin reviews the request for a refund and checks the transaction reversal.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin approves refunds.	Requests displayed.	Displayed	Pass
2	Admin clicks on refund request.	Details shown.	Shown	Pass
3	Admin reviews transaction.	Transaction verified.	Verified	Pass
4	Admin approves refund.	Refund processed.	Processed	Pass
5	System reverses transaction.	Amount reversed.	Reversed	Pass
6	User informed of refund.	Notification sent.	Sent	Pass

Table 81: Refund Processing by Admin – Test Case

24. Notification Service Down

Test Case ID	**P2-01-24**				
Test Case Name	Notification Service Down	**Test Case Description**	Test system operation without a notification service.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	2 March 2026

S. No	**Prerequisites:**
1	Notification service is down
2	Reminder trigger exists

Initial Execution

Test Scenario	An attempt was made to send the notification when the notification service was not available.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Notification event is triggered by System.	Event triggered.	Triggered	Pass
2	System is now connected to notification service.	Connection attempt.	Failed	Fail
3	System retries sending notification.	Retry mechanism activated.	Not activated	Fail
4	System queues notification.	Notification queued.	Not queued	Fail
5	Error logged.	Error logged.	Not logged	Fail
6	User notified later.	Deferred notification.	Not sent	Fail
7	System alerts admin.	Admin alerted.	Not alerted	Fail

Table 82: Notification Service Down – Test Case

Re-Test Execution

Test Scenario	An attempt was made to send the notification when the notification service was not available.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Notification service restored.	Service available.	Available	Pass
2	System triggers notification event.	Event triggered.	Triggered	Pass
3	System queued notifications to be retried.	Retry successful.	Successful	Pass
4	Notification sent	Sent.	Sent	Pass
5	User receives notification	Received.	Received	Pass
6	Error logs updated	Logs updated.	Updated	Pass

Notes:

- No retry and queueing in system during service outage. Implement error handling and notification queue.

25. Transaction Email Generation

Test Case ID	P2-01-23				
Test Case Name	Transaction Email Generation	**Test Case Description**	Check that system sends and mails confirmation of transaction following successful payment.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	2 March 2026

S. No	Prerequisites:
1	Successful payment completed

Initial Execution

Test Scenario	System creates a receipt & sends an email to the user upon deposit.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	System detects successful payment.	Payment confirmed	Confirmed	Pass
2	System generates receipt.	Receipt generated.	Generated	Pass
3	Deposit saved in database.	Saved successfully.	Saved	Pass
4	System prepares email.	Email content created.	Created	Pass
5	System sends email.	Email sent.	Sent	Pass
6	User receives confirmation email.	email received.	Received	Pass

Table 83: Transaction Email Generation – Test Case

26. Map Tracking after Dispatch

Test Case ID	P2-01-26				
Test Case Name	Map Tracking after Dispatch	**Test Case Description**	Ensure that users can locate a product after dispatch.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 March 2026

S. No	Prerequisites:
1	Product dispatched.
2	Tracking enabled

Initial Execution

Test Scenario	The integration of maps was used to send user tracks to product.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	System updates status of dispatch.	Status updated.	Updated	Pass
2	User opens tracking page.	Map loaded.	Loaded	Pass
3	System retrieves location information.	Data retrieved.	Retrieved	Pass
4	Current location on Map.	Location shown.	Shown	Pass
5	User refreshes map.	Location updated.	Updated	Pass
6	User exits tracking.	Session ended.	Ended	Pass

Table 84: Map Tracking after Dispatch – Test Case

27. Session Timeout Failure

Test Case ID	**P2-01-27**				
Test Case Name	Session Timeout Failure	**Test Case Description**	Test system behavior when sessions are not timed out.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 March 2026

S. No	**Prerequisites:**
1	User is logged in.

Initial Execution

Test Scenario	User does not use the system within the timeout period.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User logs in	Login successful	Successful	Pass
2	User remains inactive	Timer starts	Started	Pass
3	Session timeout reached	User logged out	Still logged in	Fail
4	User refreshes page	Redirect to login	Dashboard shown	Fail
5	User navigates secured page	Access denied	Access allowed	Fail
6	System logs timeout	Timeout logged	Not logged	Fail

7	Security alert triggered	Alert triggered	Not triggered	Fail

Table 85: Session Timeout Failure – Test Case

Re-Test Execution

Test Scenario	User does not use the system within the timeout period.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User logs in	Login successful	Successful	Pass
2	User inactive (passed timeout).	Session expires	Expired	Pass
3	System logs user out.	Logged out	Logged out	Pass
4	User refreshes page	Redirected to login	Redirected	Pass
5	Timeout event logged	Log created	Created	Pass
6	Security alert sent	Alert sent	Sent	Pass

Notes:

- The session timeout setting was not properly implemented.

28. Payment Gateway Timeout

Test Case ID	**P2-01-28**				
Test Case Name	Payment Gateway Timeout	**Test Case Description**	Check the system handling if payment gateway times out.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	23 Dec 2026

S. No	**Prerequisites:**
1	Payment initiated

Initial Execution

Test Scenario	Payment gateway is not responding for timeout.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User submits payment	Processing started	Started	Pass
2	Gateway timeout occurs	Retry initiated	No retry	Fail

3	User notified	Timeout message	Not shown	Fail
4	Payment reversed	Reversed	Pending	Fail
5	Transaction logged	Logged	Not logged	Fail

Table 86: Payment Gateway Timeout – Test Case

Final Execution

Test Scenario	Payment gateway is not responding for timeout.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User submits payment	Processing started	Started	Pass
2	Gateway timeout detected	Detected	Detected	Pass
3	Deposit rolled back	Rolled back	Rolled back	Pass
4	User notified	Notification sent	Sent	Pass
5	Retry option shown	Shown	Shown	Pass
6	Failure logged	Logged	Logged	Pass

Notes:

- Timeout handling implemented.

29. Accept Delivery Assignment

Test Case ID	**P2-01-29**				
Test Case Name	Accept Delivery Assignment	**Test Case Description**	Confirm a rider's ability to take and accept a delivery assignment.		
Created By	Naveed Asad Ali	**Version**	1.1	**Date**	15 May 2026

S. No	**Prerequisites:**
1	The rider status will be 'Online'.
2	An order has been packed by a store and passed to the dispatch engine.

Initial Execution

Test Scenario	New delivery alert sent to rider and the new delivery is successfully associated with the rider when accepted.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**

1	Dispatch engine matches delivery to Rider.	Match found.	Matched	Pass
2	System creates DeliveryAssignment (PENDING).	Record created.	Created	Pass
3	Push notification alert to Rider.	Alert received.	Received	Pass
4	Rider taps 'Accept'.	Acceptance registered.	Registered	Pass
5	System updates assignment to 'ACCEPTED'.	Status updated.	Updated	Pass
6	UI loads navigation route to the Store.	Route displayed.	Displayed	Pass

Table 87: Accept Delivery Assignment – Test Case

30. Proof of Delivery Upload

Test Case ID	**P2-01-30**				
Test Case Name	Proof of Delivery Upload	**Test Case Description**	Ensure the delivery is made successfully, accept any photo evidence of drop off.		
Created By	Naqash Sachwani	**Version**	1.1	**Date**	15 May 2026

S. No	**Prerequisites:**
1	Delivery status is In Transit.
2	Rider is at the customer's drop-off location.

Initial Execution

Test Scenario	There was a problem with the delivery system and the PoD photo could not be uploaded.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Rider takes a picture of the package.	Photo captured.	Captured	Pass
2	System uploads the photo to Storage.	Upload successful.	Upload Timeout	Fail
3	System creates ProofOfDelivery DB record.	Record created.	Not Created	Fail
4	Delivery status changed to 'DELIVERED'.	Status updated.	Not Updated	Fail

5	System calculates fee & credits payout.	Payout initiated .	Not Credited	Fail
6	System sends notification to Customer.	Notification sent.	Not Sent	Fail

Table 88: Proof of Delivery Upload – Test Case

Re-Test Execution

Test Scenario	Successfully uploads photo, marks delivery complete and credits payouts in system.			
Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Rider takes a picture of the package.	Photo captured.	Captured	Pass
2	System uploads the picture to Storage.	Upload successful.	Successful	Pass
3	System generates a ProofOfDelivery DB record.	Record created.	Created	Pass
4	Delivery status changed to 'DELIVERED'.	Status updated.	Updated	Pass
5	System calculates fee & payouts.	Payout initiated.	Credited	Pass
6	System sends notification to Customer.	Notification sent.	Sent	Pass

Notes:

- Client side image compression has been implemented and the workflow run successfully.

User Manual

1. Introduction

1.1 Purpose

This manual aims to give an overview of how to navigate and use DreamSaver. It explains what users can do with which user role, and how they can do it.

1.2 Scope

This document describes the full scope of operations of the DreamSaver web application. It encompasses setting up savings objectives, inventory control, payment handling via secure escrow system, rider logistics and tracking, and admin management.

1.3 Intended Audience

- Customers (End Users): Individuals who are interested in buying products in the digital layaway model.
- Sellers (Stores): The stores which are responsible for the management of their products, the progression of layaway, and dispatching orders.
- Logistics Personnel (Riders): Drivers in charge of their current deliveries and viewing proofs of delivery.
- Administrators are platform managers responsible for managing the process of approving stores, verifying riders and handling disputes.

1.4 System Overview

DreamSaver is a modern, responsive Web Application that will replace the traditional layaway purchasing concept. It enables the customer to secure the price of an item, and then make flexible repayments for it. Safe payment processes, role based dashboards and fully integrated logistics mapping system.

2. System Requirements

2.1 Hardware Requirements

DreamSaver is a web-hosted application and doesn't need much hardware:

- Memory: 2 GB or more
- Storage: Minimal (Browser cache only)
- Additional: A good internet connection. Riders need to use a mobile device camera to take Proof of Delivery.

2.2 Software Requirements

- Operating System: Windows, macOS, Linux, iOS, or Android.
- Supported Browsers: Google Chrome, Mozilla Firefox, Apple Safari, or Microsoft Edge (latest versions are recommended).

3. Installation & Access

3.1 Accessing the Application

The end-users don't need to install DreamSaver anywhere.

1. Go to a trusted web browser.=.
2. Go to the official DreamSaver URL.
3. Click Sign In / Sign Up to sign in and sign up securely at the log in portal.

3.2 Account Configuration

- Customers: No additional configuration is required after registration.
- Stores: To access the Seller Dashboard, must first submit a Store Application (Business Name, CNIC, Address) and await Admin Approval.
- Riders: Riders should provide a rider profile (Vehicle Type, License Plate, CNIC), and await admin approval prior to accepting deliveries.

4. System Features

- Digital Layaway & Savings Goals: No credit check is required for customers to reserve products and pay over time.
- Price Lock Guarantee: Holds the price guaranteed for buyers as they work towards a savings target, safeguarding them against price changes.
- Multi-Role Dashboards: Separate, secure areas for Customers, Stores, Riders and Admins.
- Integrated Logistics Map: Real-time routing and tracking for deliveries with the use of the map.
- Secure Escrow & Wallet: Payments are held in escrow and are safe until delivery is confirmed through a secure system.

5. User Interface Guide

5.1 Navigation Overview

5.1.1 User Interface

5.1.1.1 Sign Up

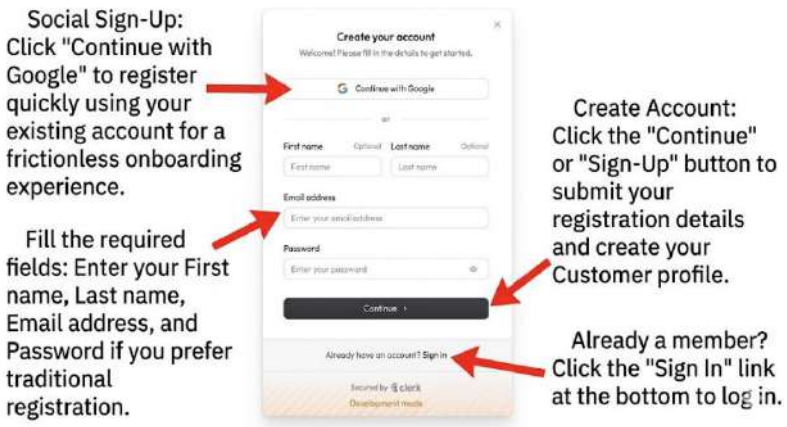


Figure 122: Sign Up – Navigation

5.1.1.2 Sign In

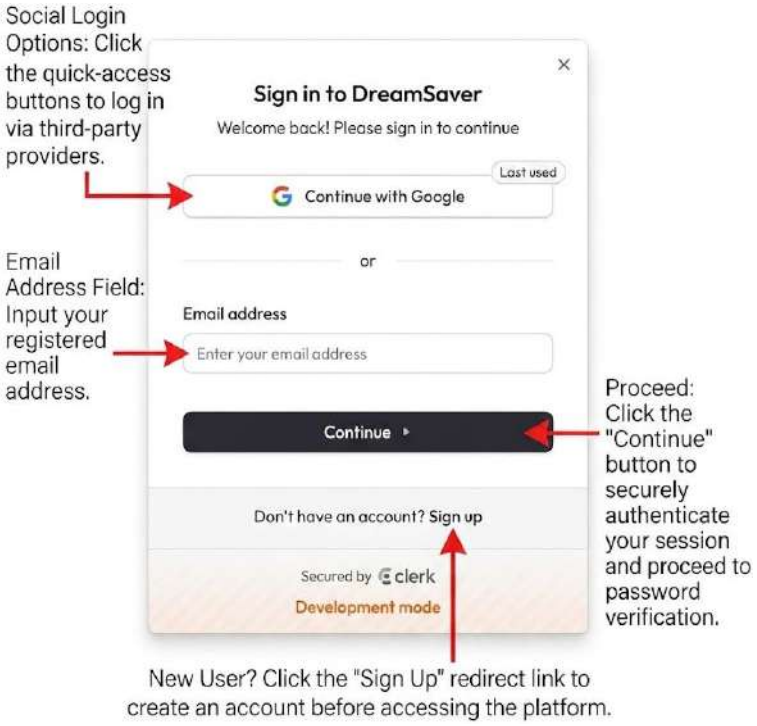


Figure 123: Sign In – Navigation

5.1.1.3 Home Page (Complete Overview)

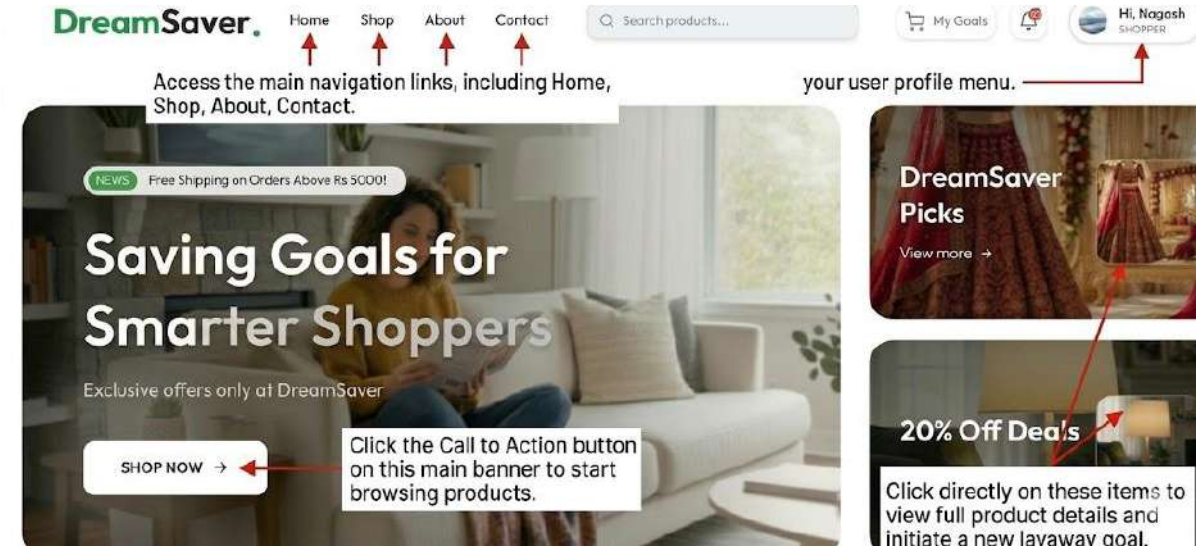


Figure 124: Home Page – Navigation

5.1.1.4 Shop

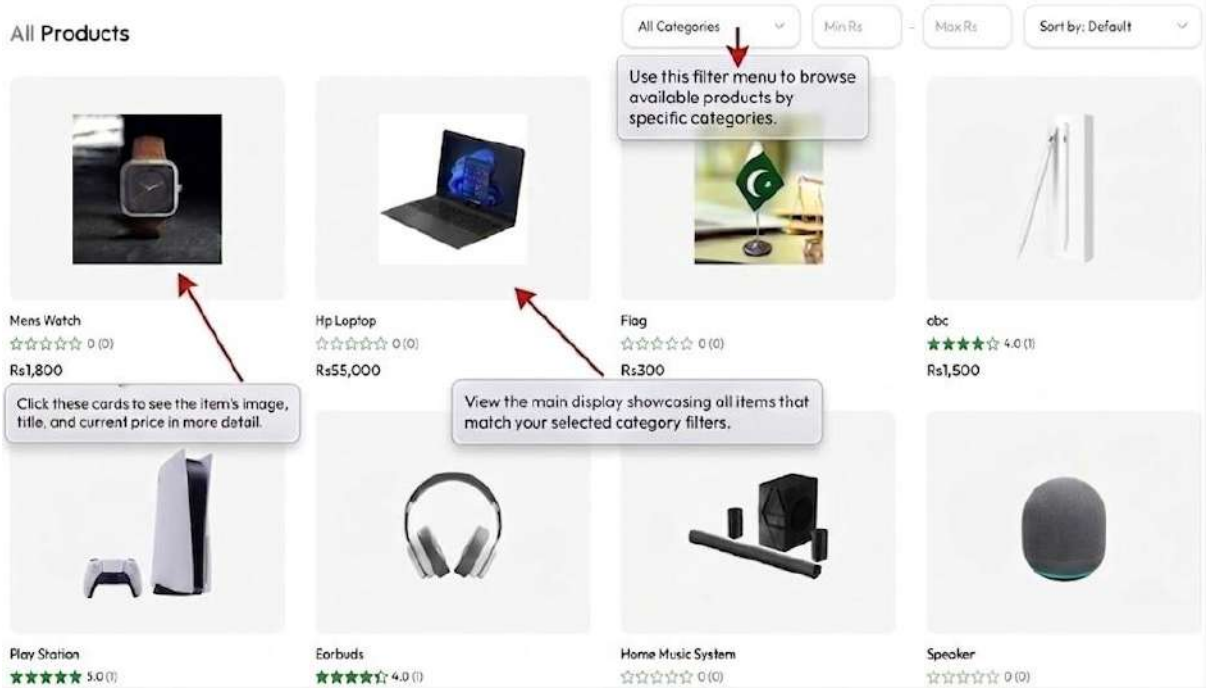


Figure 125: Shop – Navigation

5.1.1.5 About

Figure 126: About – Navigation

5.1.1.6 Contact

Figure 127: Contact – Navigation

5.1.1.7 Product

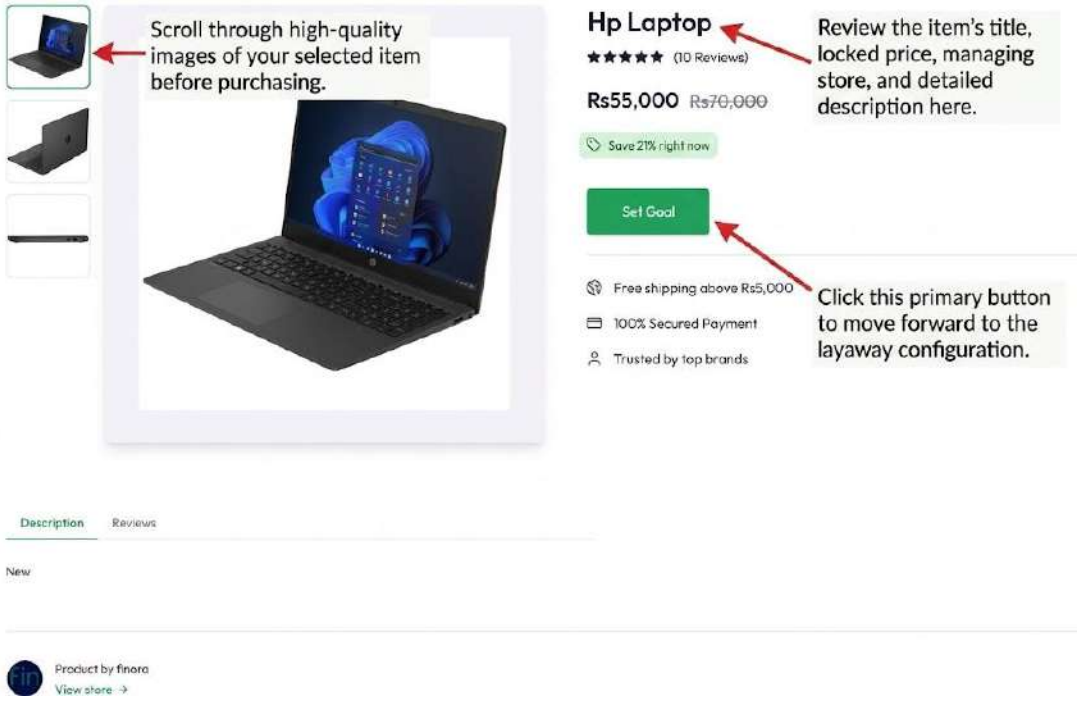


Figure 128: Product – Navigation

5.1.1.8 Set Goal

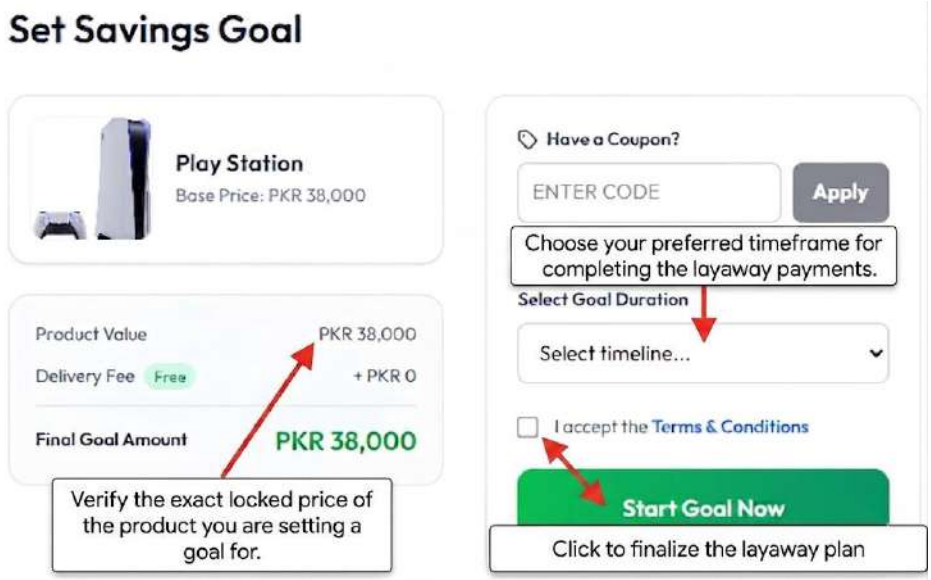


Figure 129: Set Goal – User Navigation

5.1.1.9 Payment Process (Initialization & Gateway)

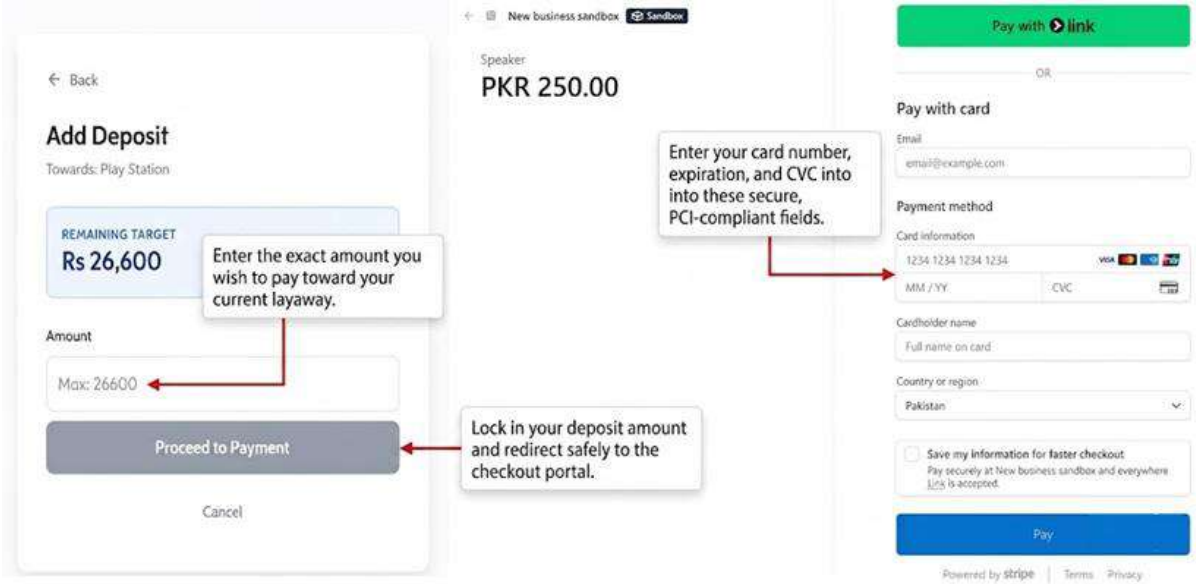


Figure 130: Payment Process – Navigation

5.1.1.10 Goal Progress

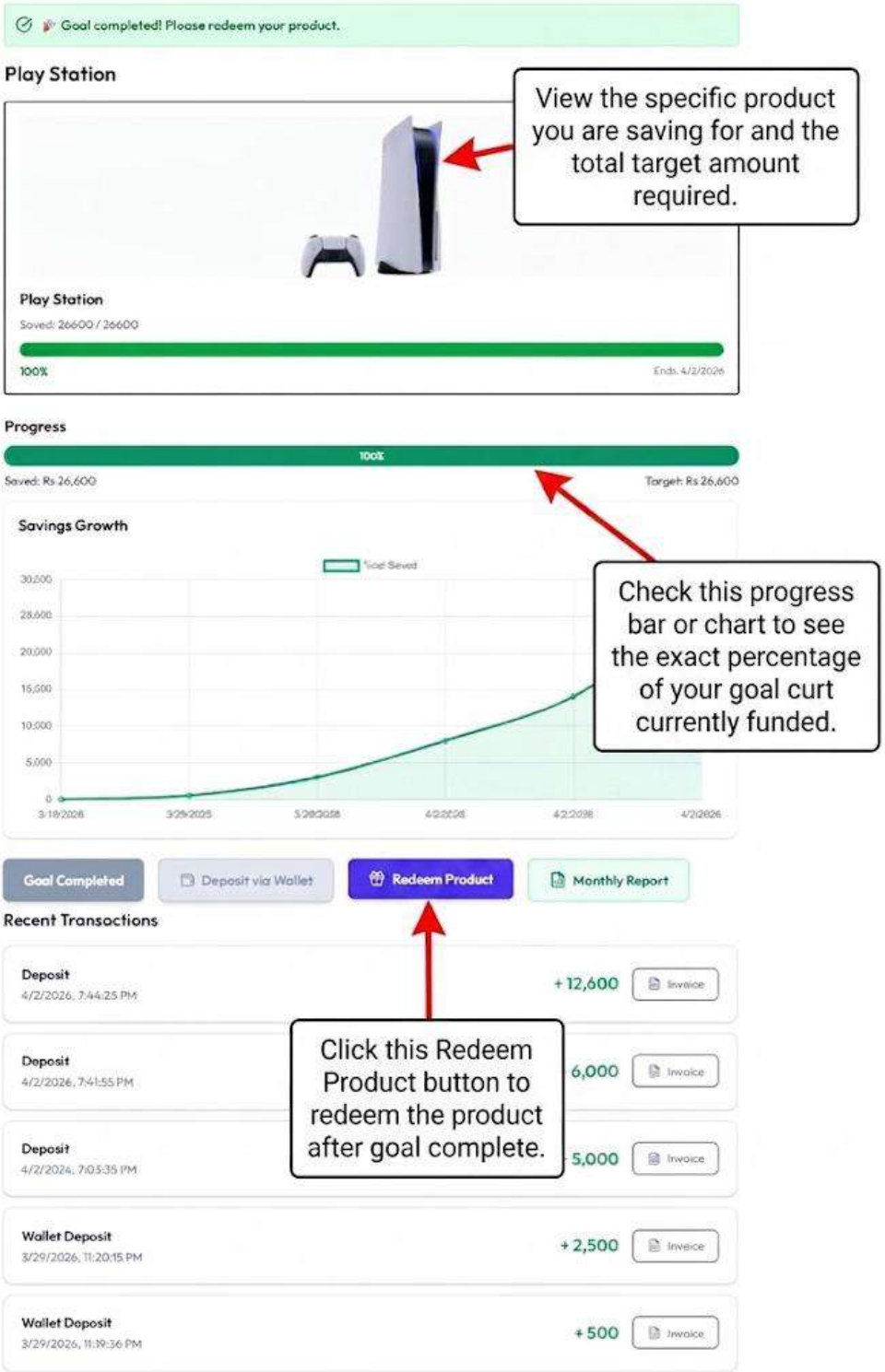


Figure 131: Goal Progress – Navigation

5.1.1.11 Goal Cart

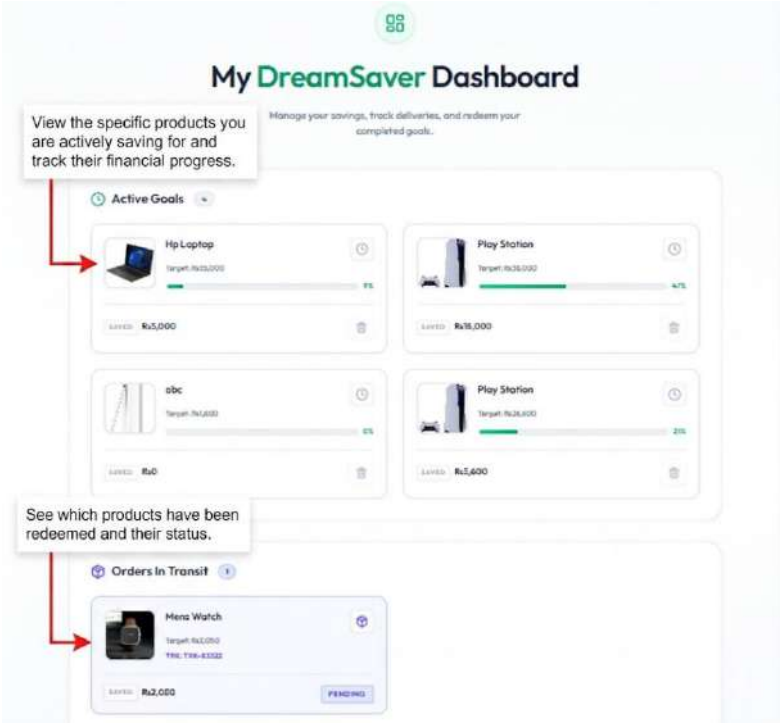


Figure 132: Goal Cart – Navigation

5.1.1.12 Wallet

Figure 133: Wallet – Navigation

5.1.1.13 Support & Complaints

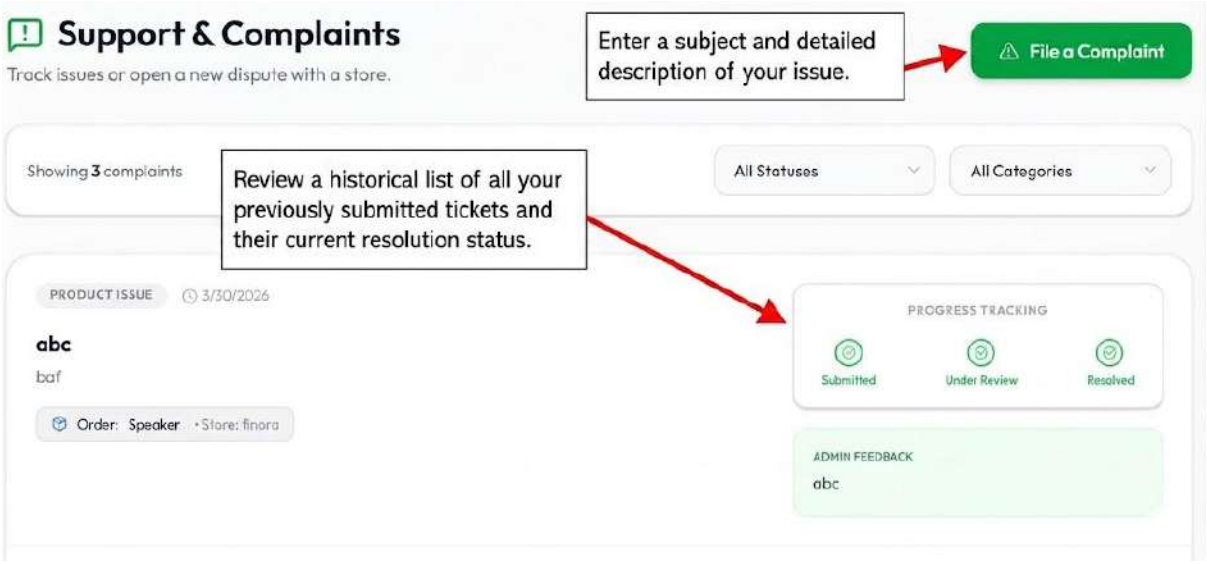


Figure 134: Support & Complaints – Navigation

5.1.1.14 Tracking

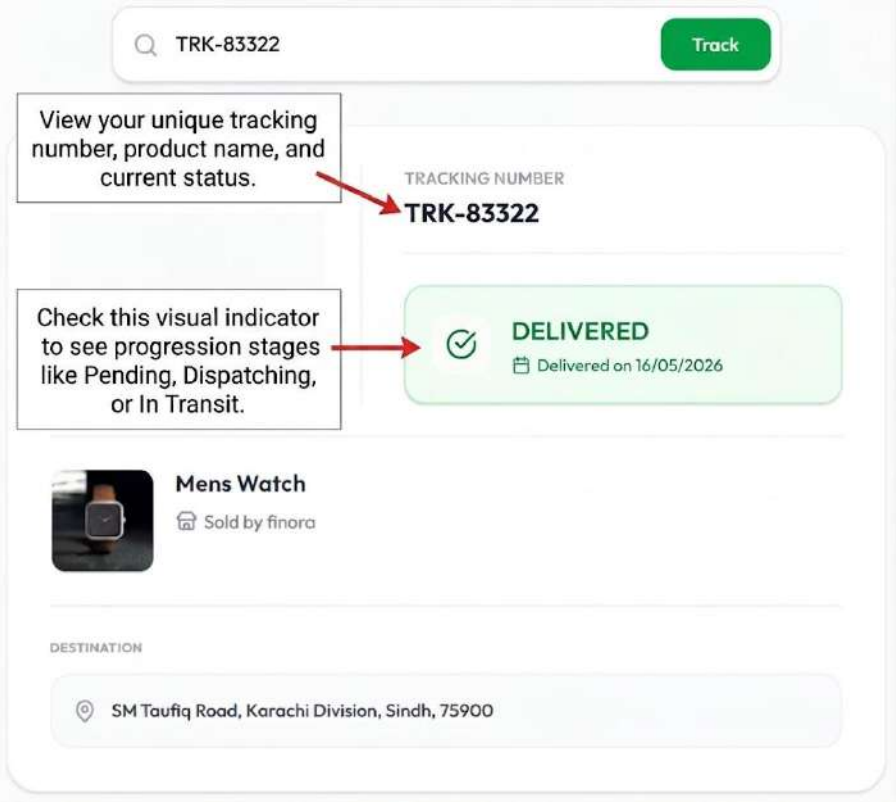


Figure 135: Tracking – Navigation

5.1.1.15 Goal History

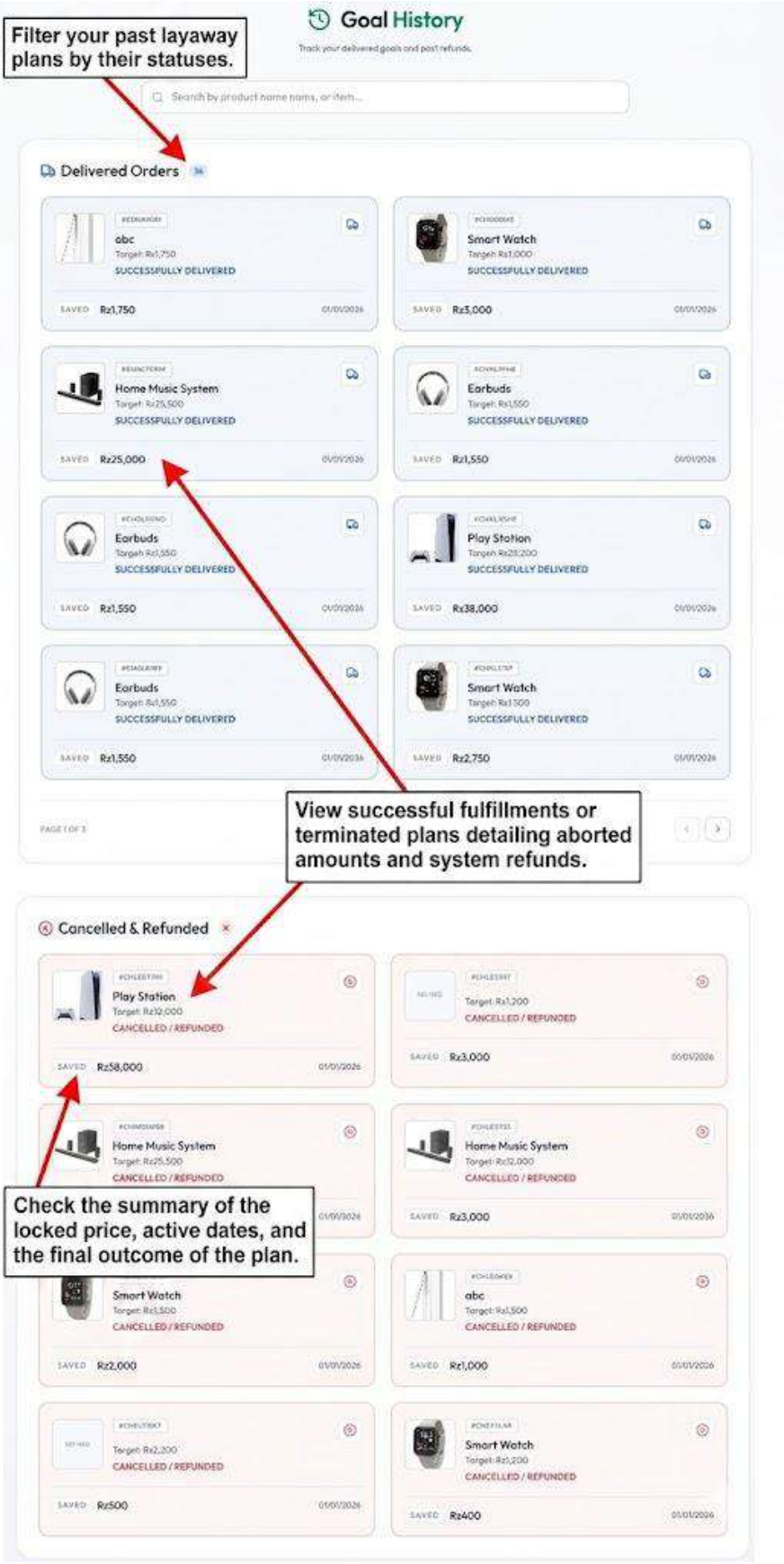


Figure 136: Goal History – Navigation

5.1.1.16 My Coupons

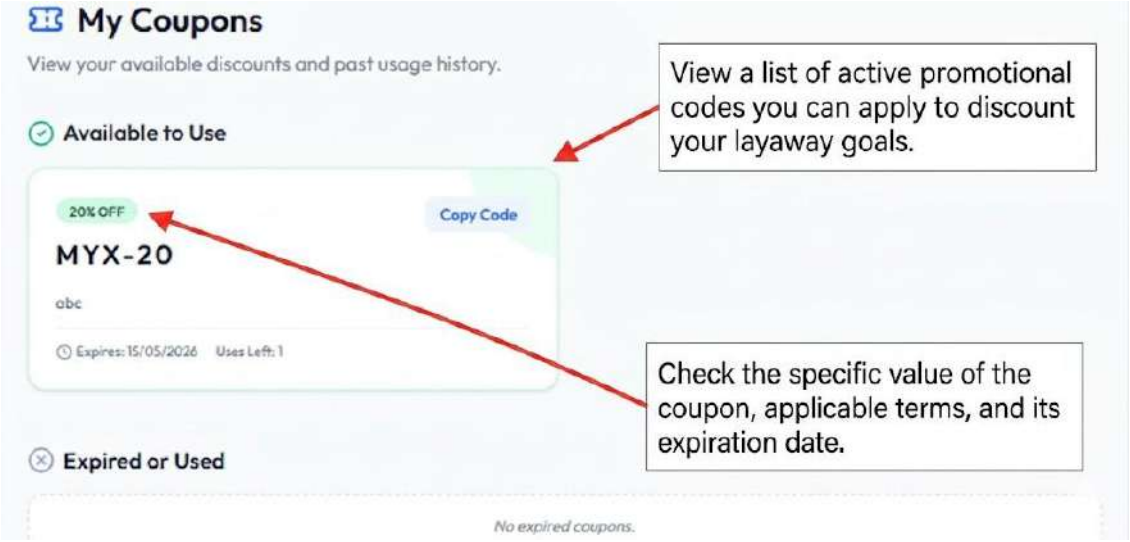


Figure 137: My Coupon – Navigation

5.1.1.17 Map Tracking

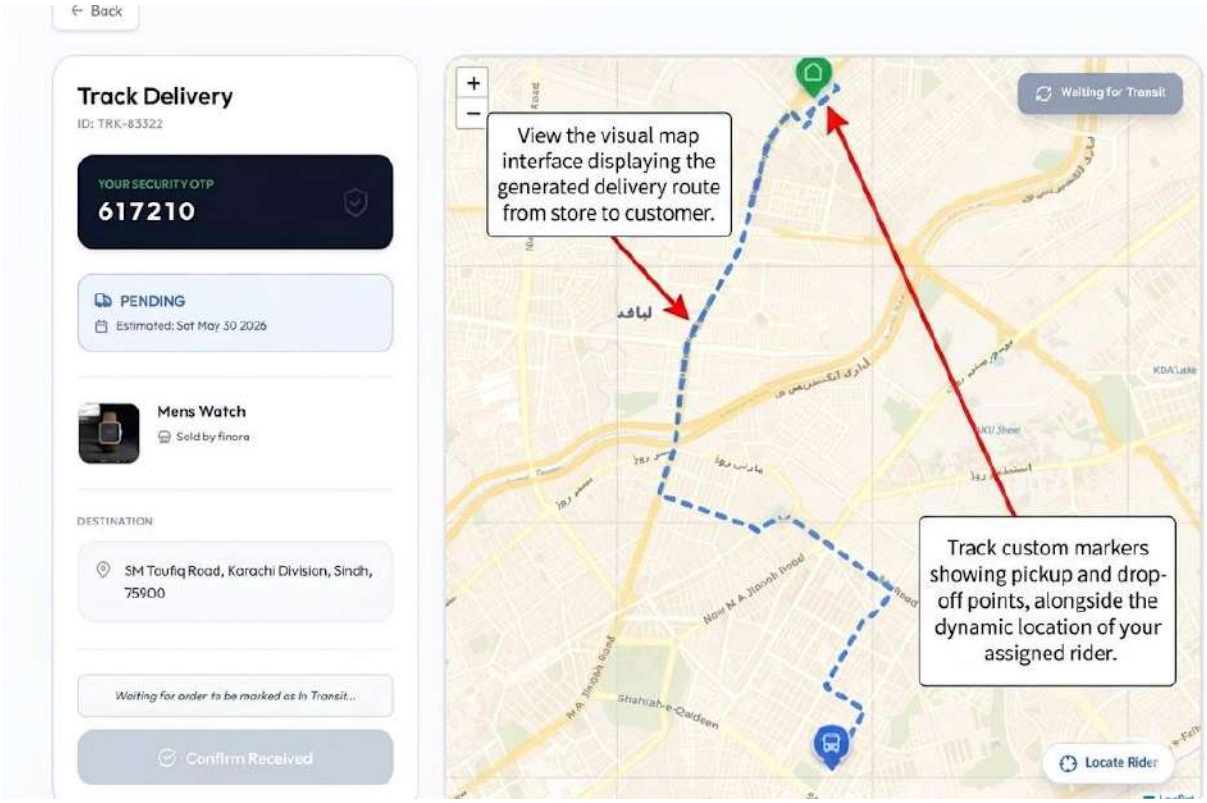


Figure 138: Map Tracking – Navigation

5.1.2 Rider Interface

5.1.2.1 Rider Registration

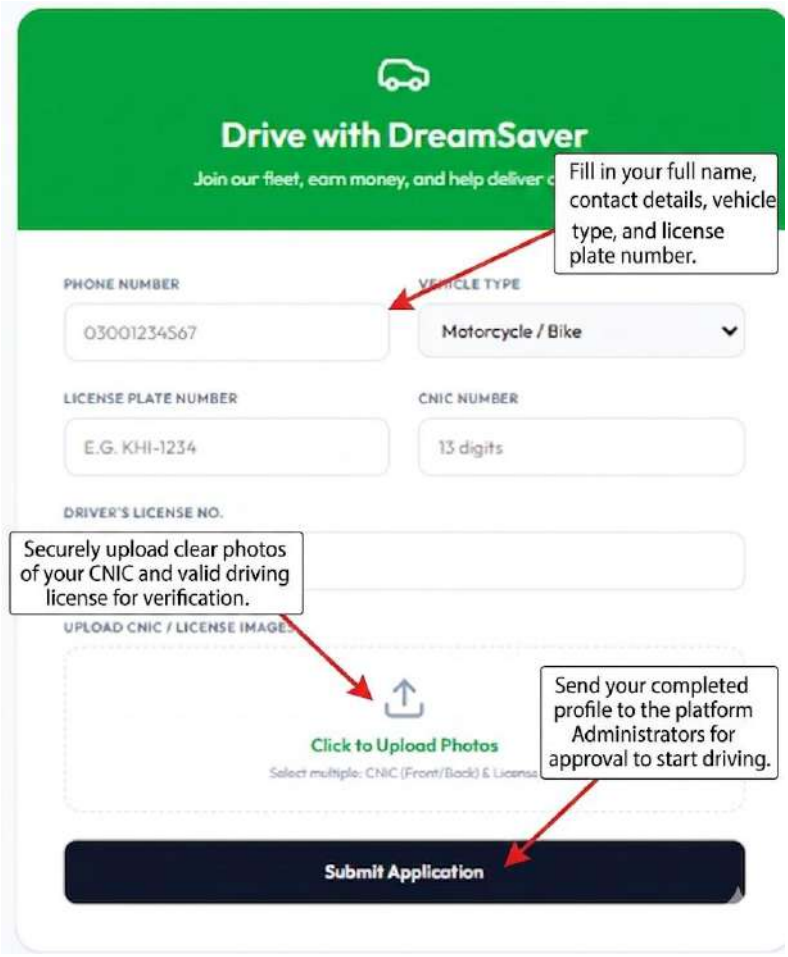


Figure 139: Rider Registration – Navigation

5.1.2.2 Rider Dashboard

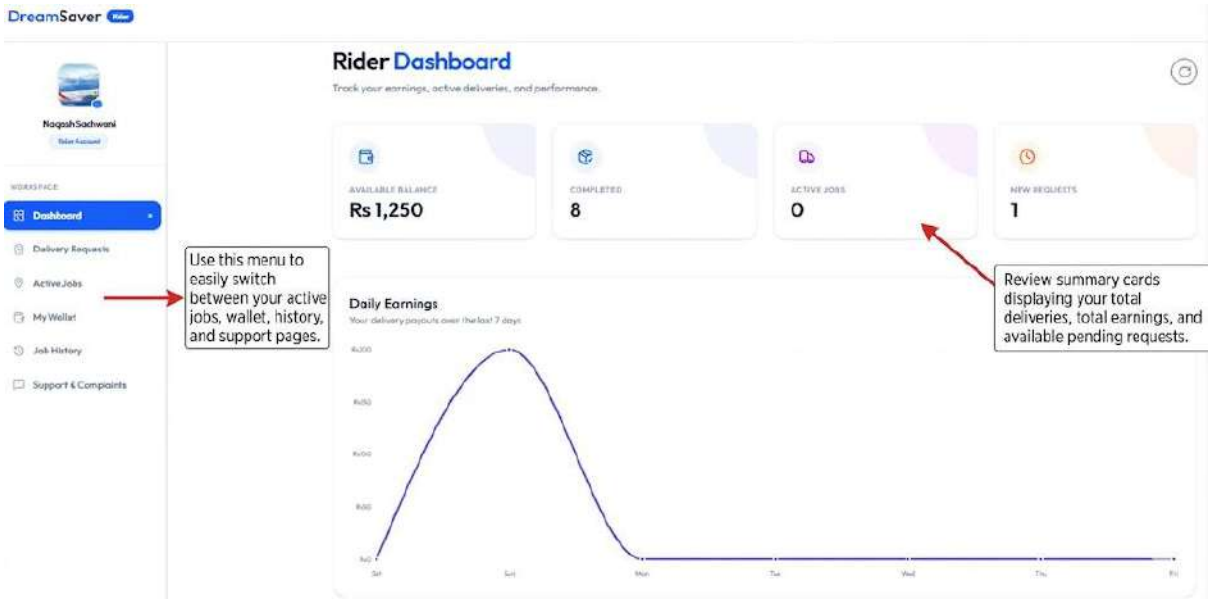


Figure 140: Rider Dashboard – Navigation

5.1.2.3 Rider Delivery Requests

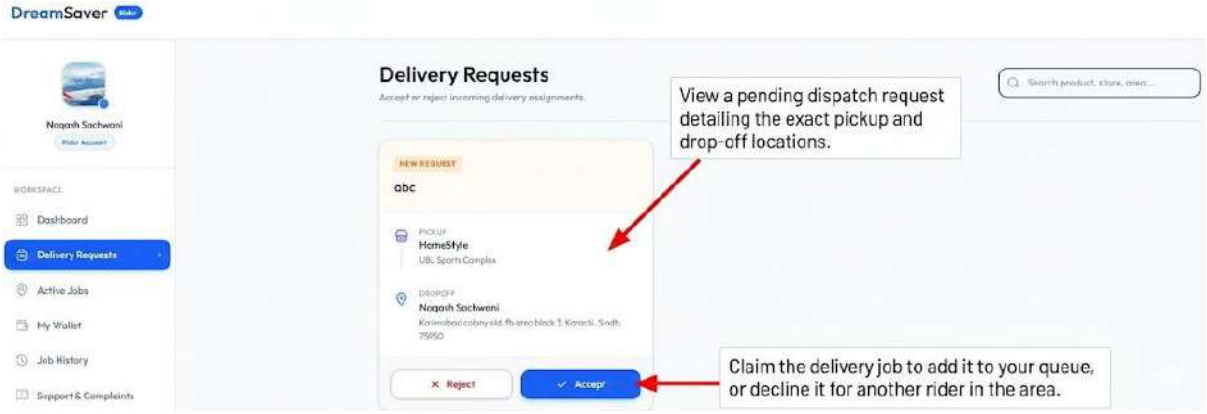


Figure 141: Rider Delivery Request – Navigation

5.1.2.4 Rider Active Jobs

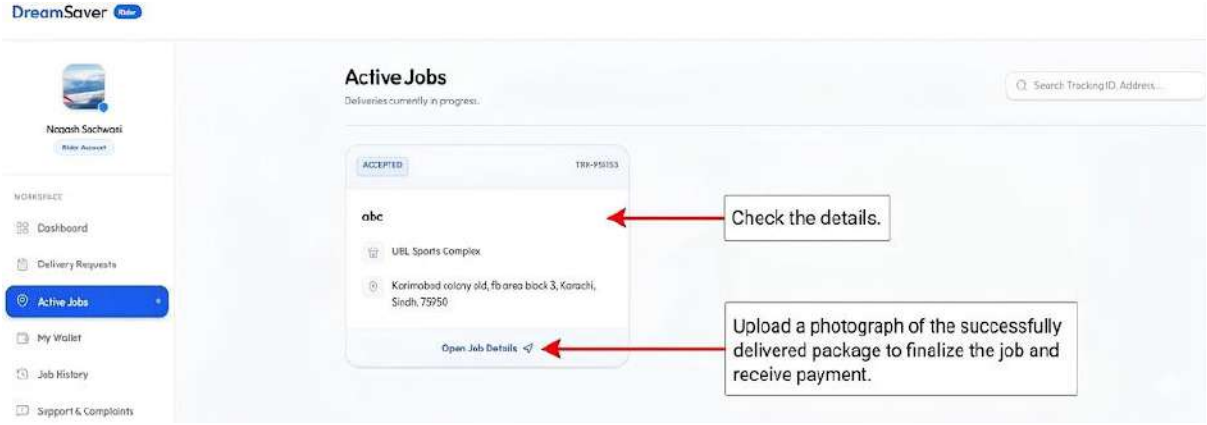


Figure 142: Rider Active Jobs – Navigation

5.1.2.5 Rider Wallet

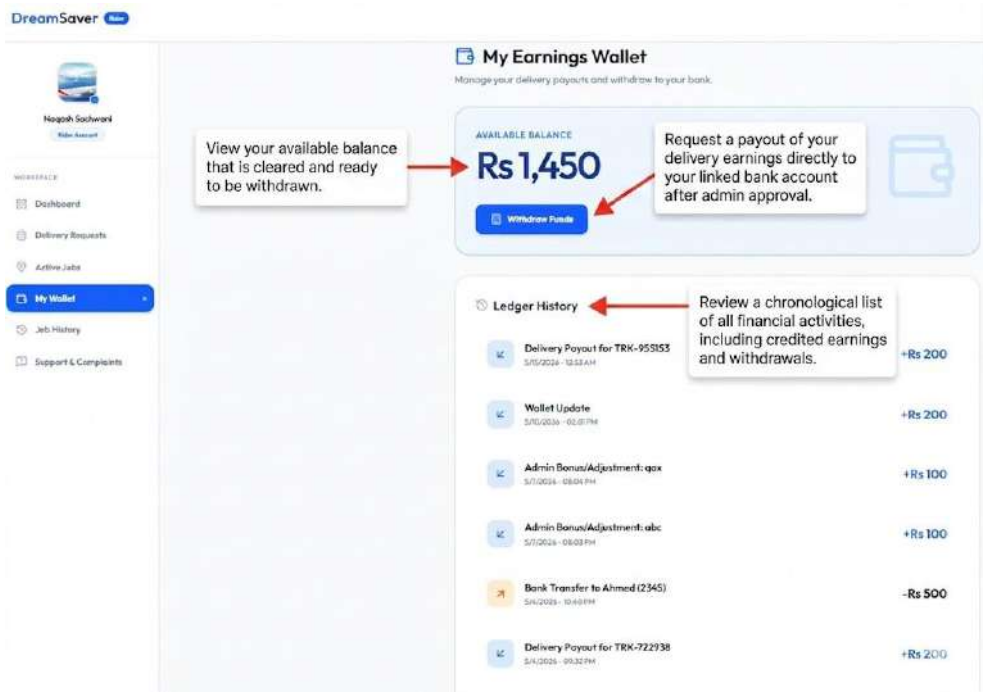


Figure 143: Rider Wallet – Navigation

5.1.2.6 Rider Job History

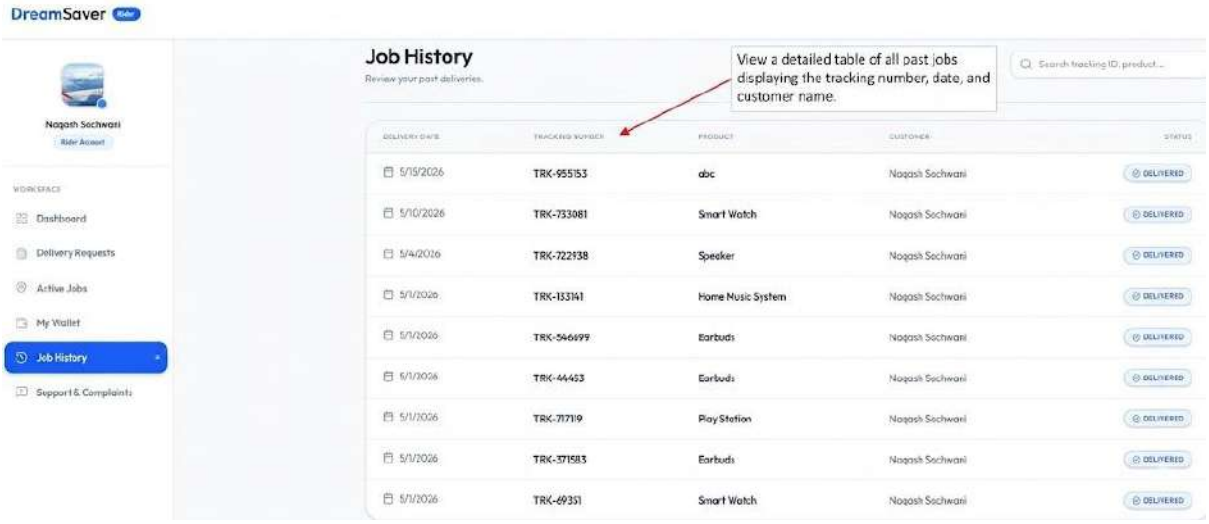


Figure 144: Rider Job History – Navigation

5.1.2.7 Rider Support & Complaints

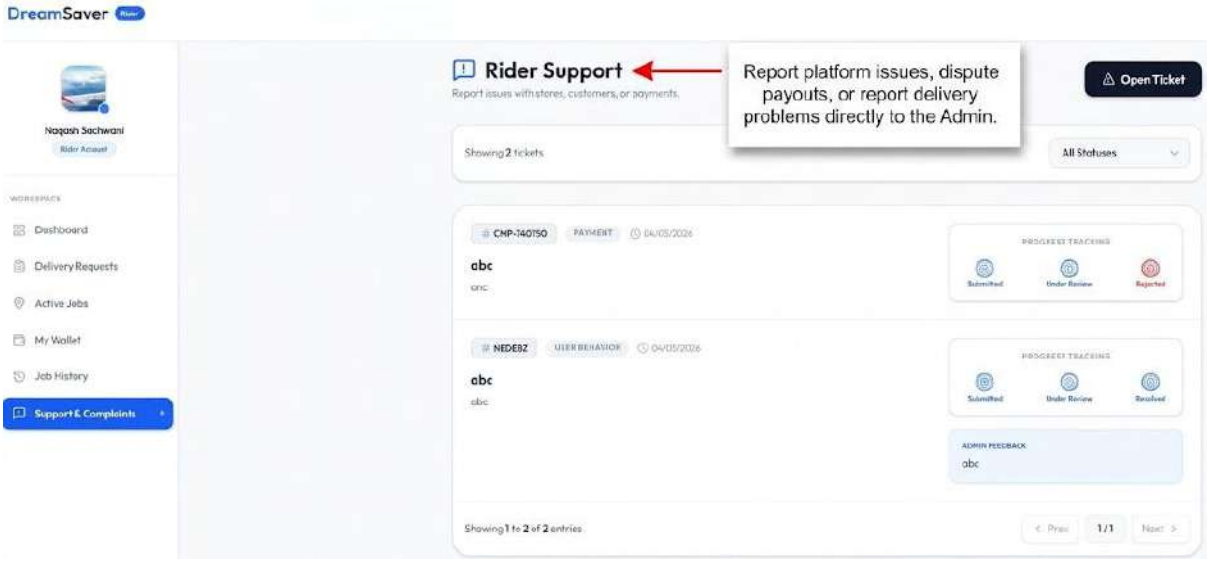


Figure 145: Rider Support & Complaints – Navigation

5.1.3 Store Interface

5.1.3.1 Store Creation Form

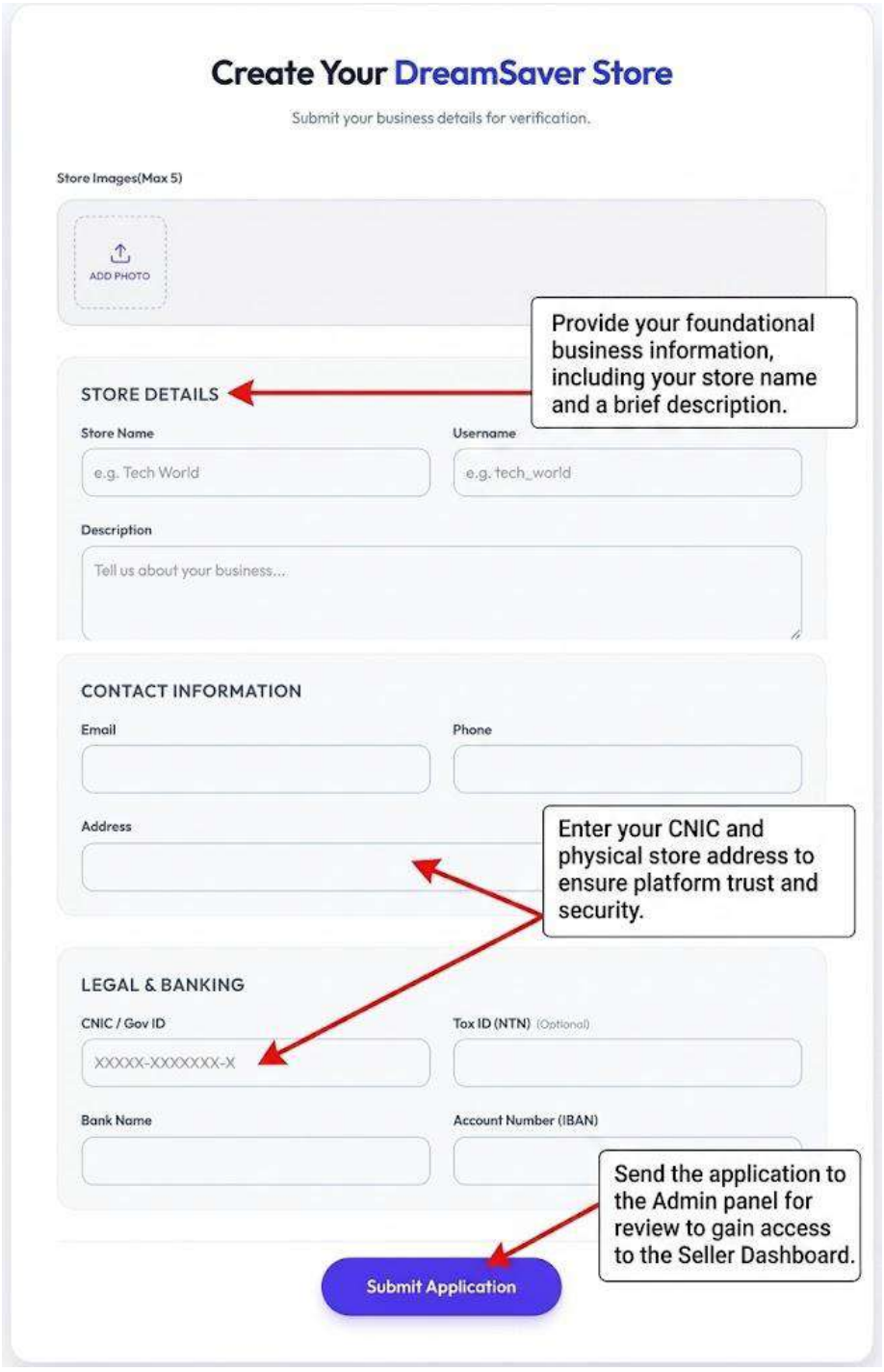


Figure 146: Store Creation Form – Navigation

5.1.3.2 Store Dashboard

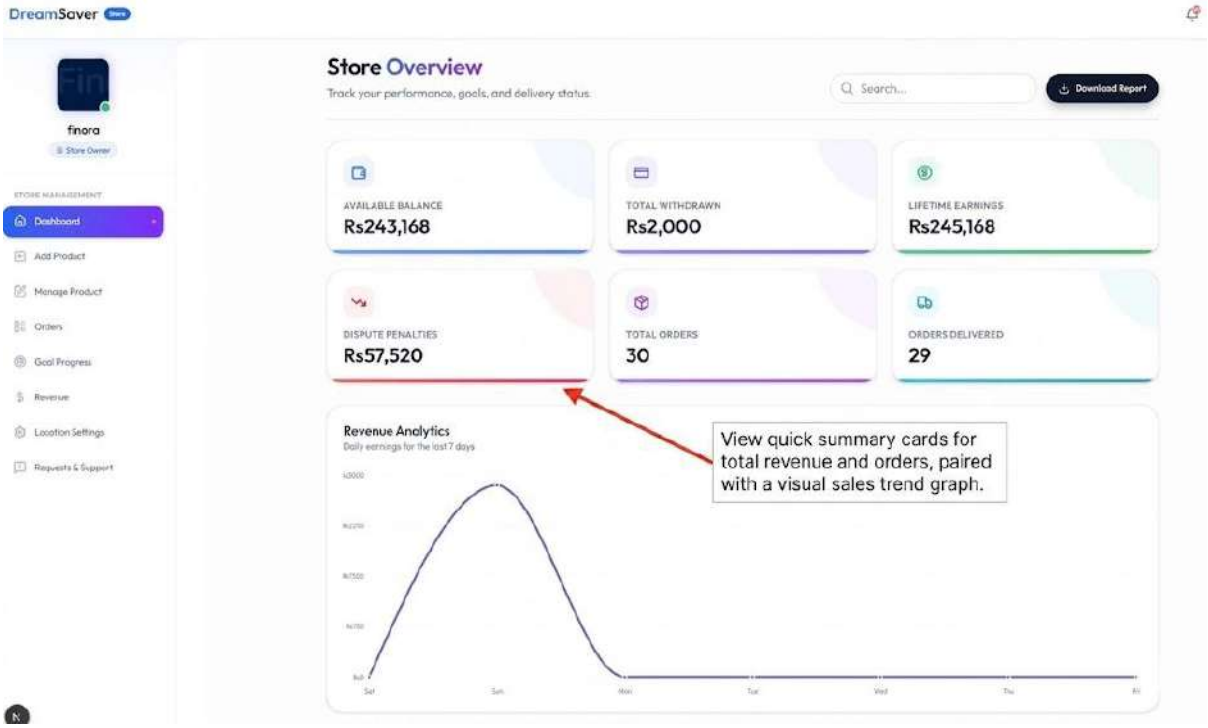


Figure 147: Store Dashboard – Navigation

5.1.3.3 Store Add Product

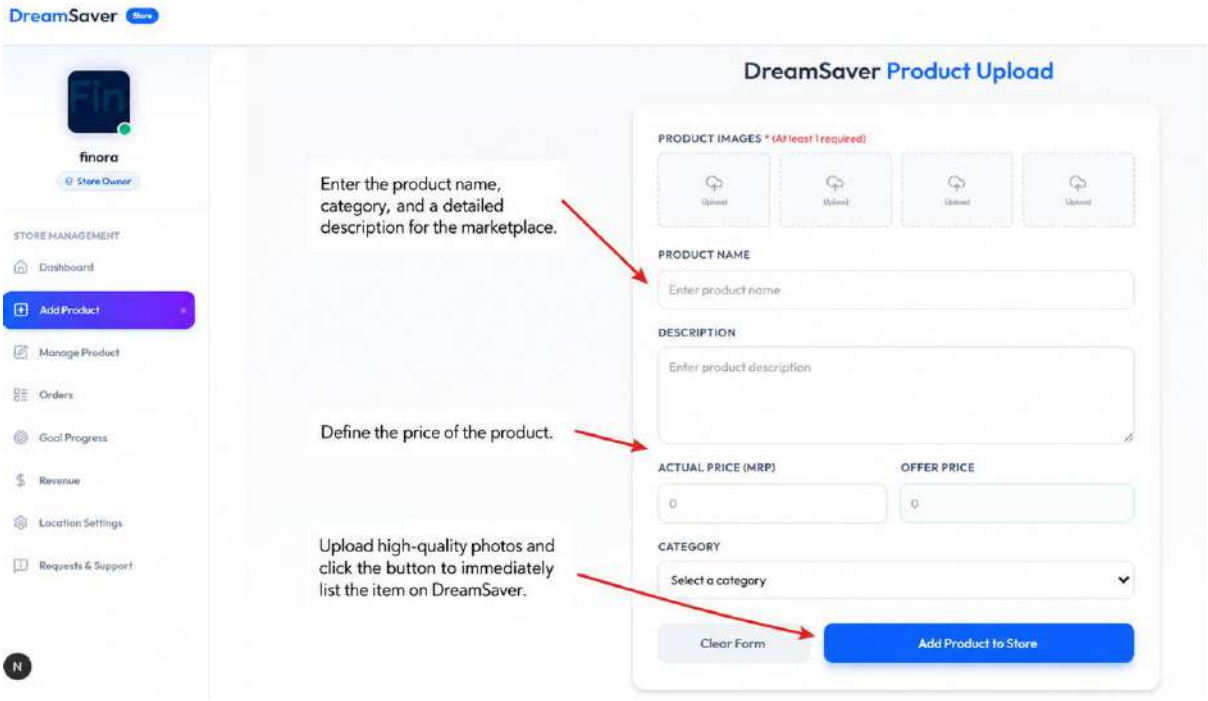


Figure 148: Store Add Product – Navigation

5.1.3.4 Store Manage Product

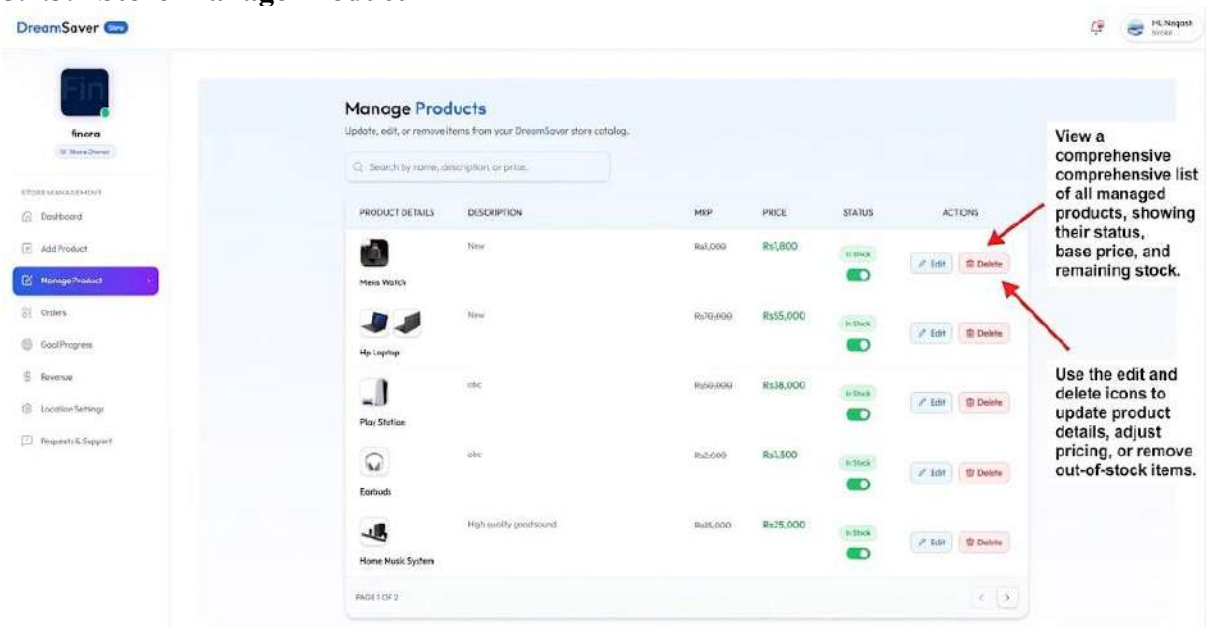


Figure 149: Store Manage Product – Navigation

5.1.3.5 Store Order

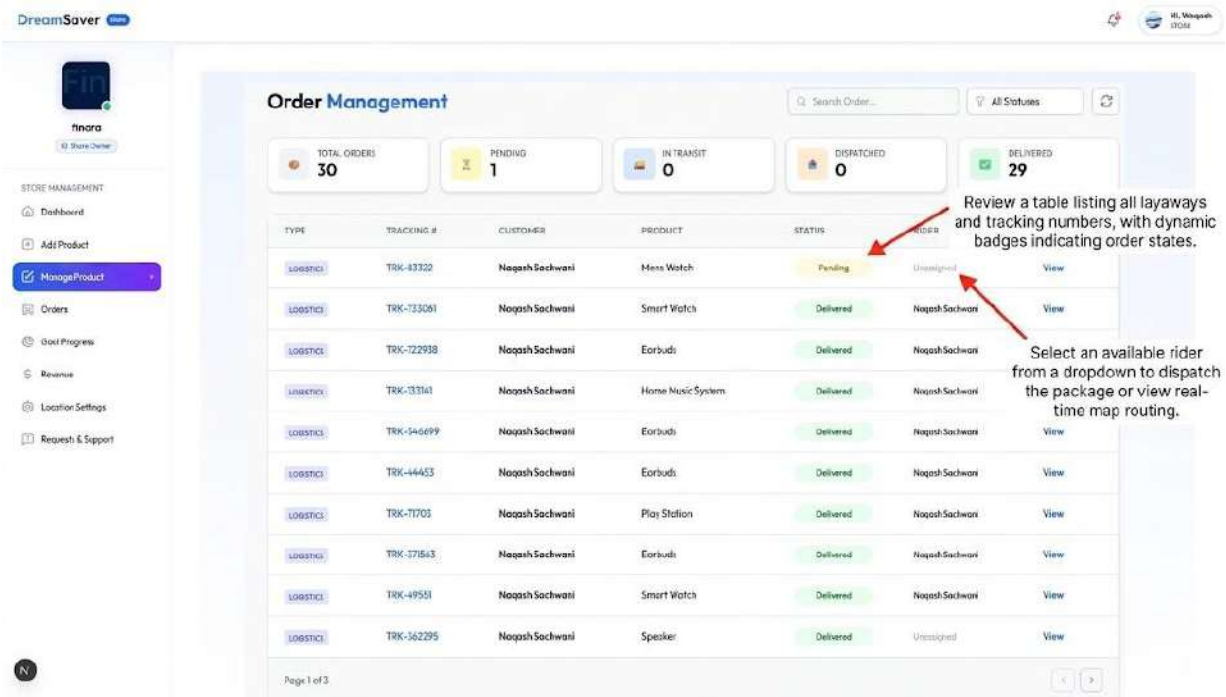


Figure 150: Store Order – Navigation

5.1.3.6 Store Location Setting

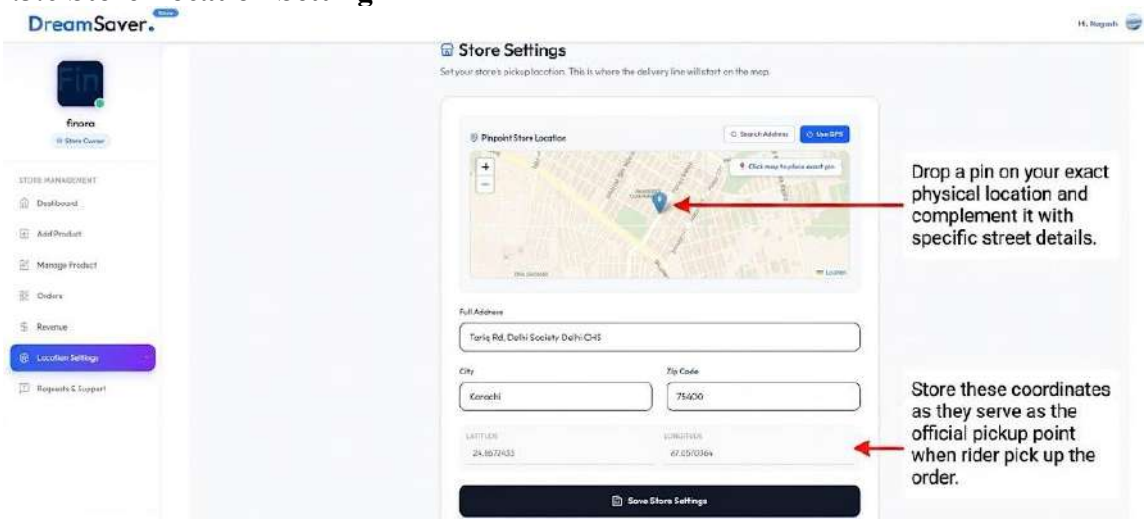


Figure 151: Store Location Setting – Navigation

5.1.3.7 Store Request & Support

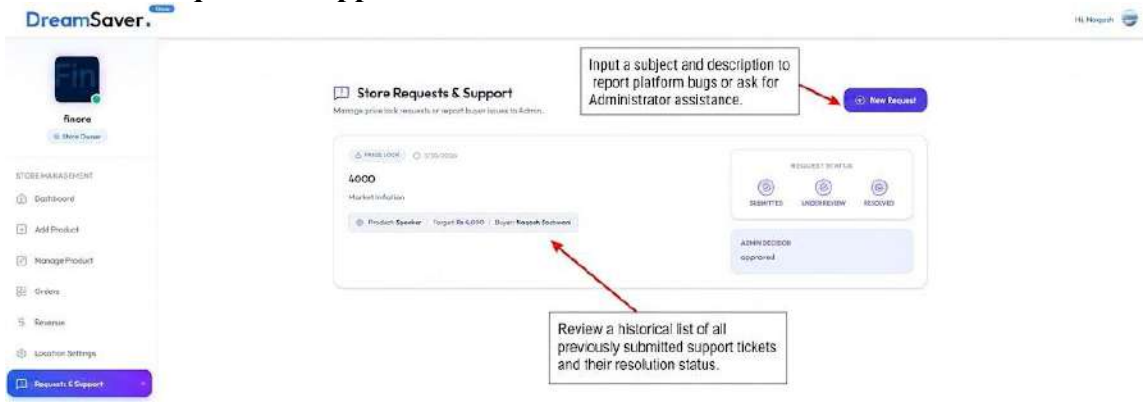


Figure 152: Store Request & Support – User Manual

5.1.3.8 Store Goal Progress

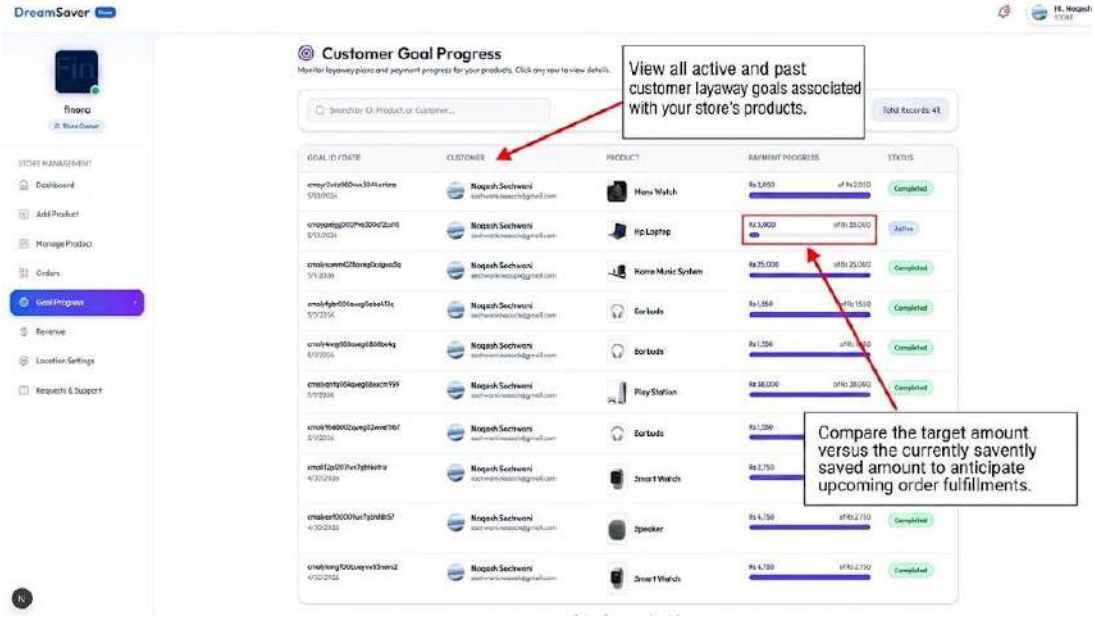


Figure 153: Store Goal Progress – Navigation

5.1.3.9 Store Revenue

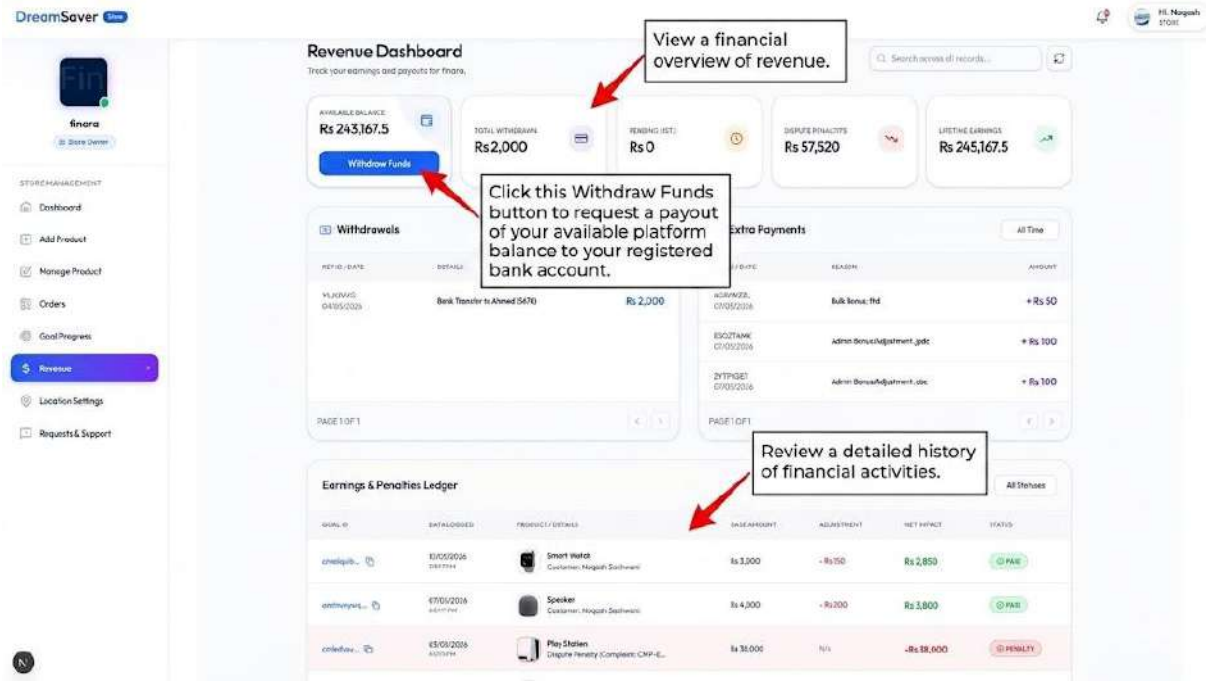


Figure 154: Store Revenue – Navigation

5.1.4 Admin Interface

5.1.4.1 Admin Dashboard

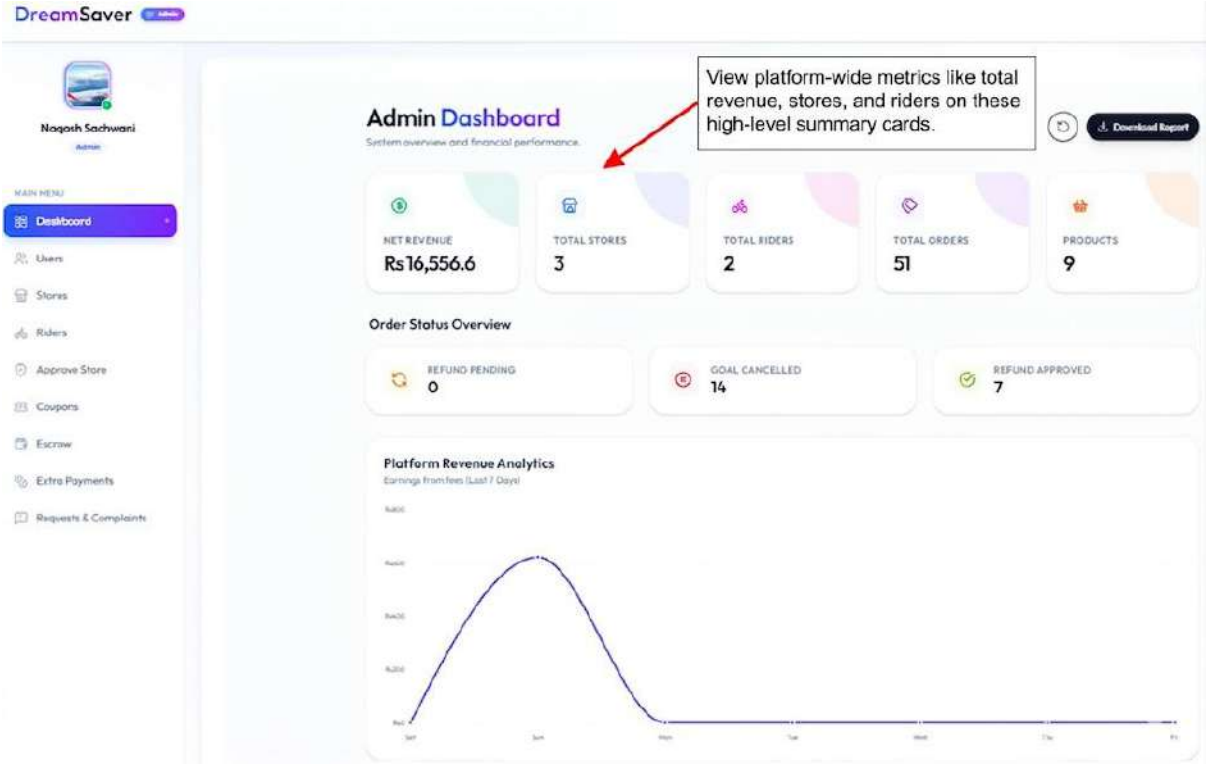


Figure 155: Admin Dashboard – Navigation

5.1.4.2 Admin Store Management

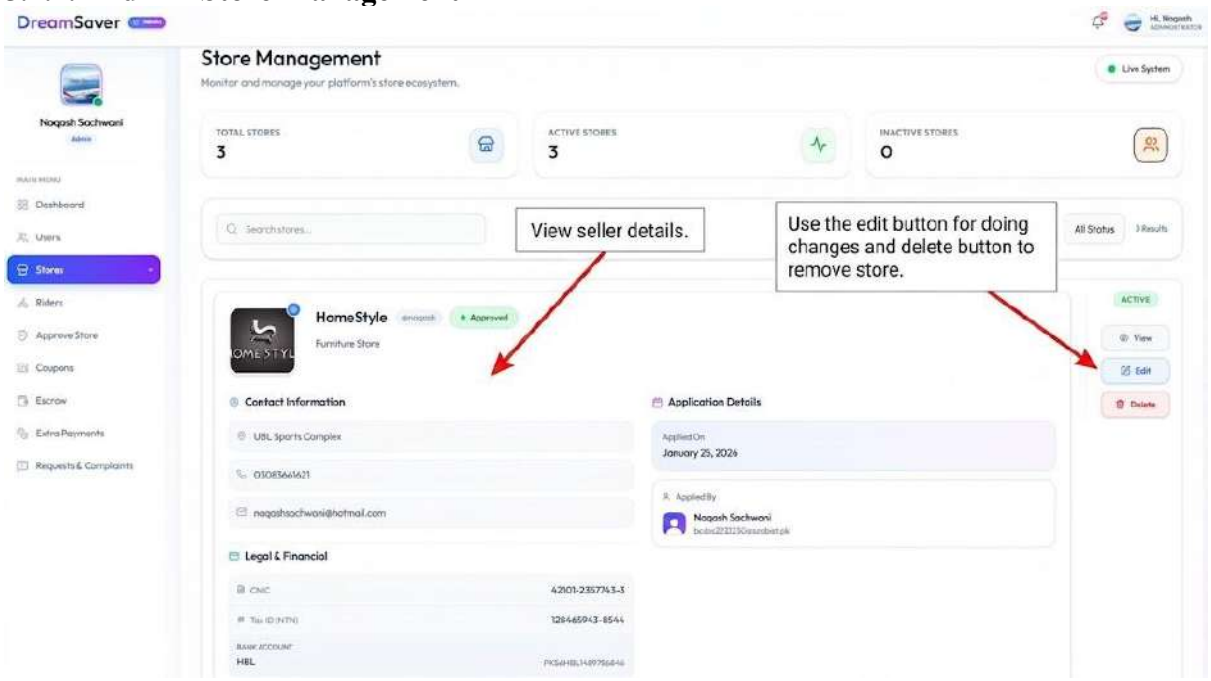


Figure 156: Admin Store Management– Navigation

5.1.4.3 Admin Store Approval

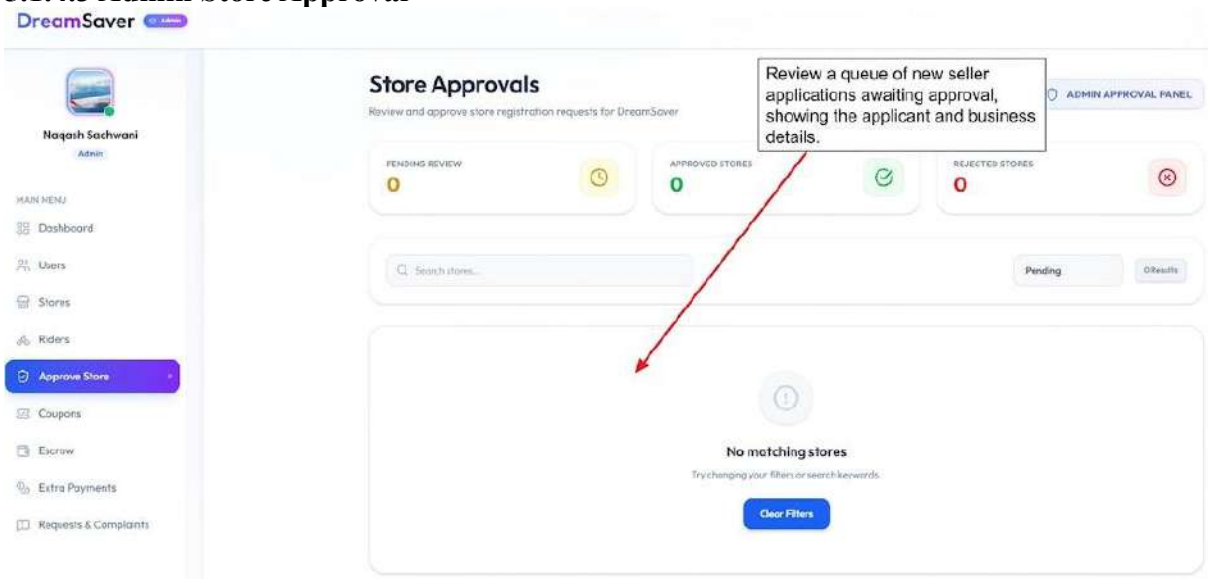


Figure 157: Admin Store Approval – Navigation

5.1.4.4 Admin User Management

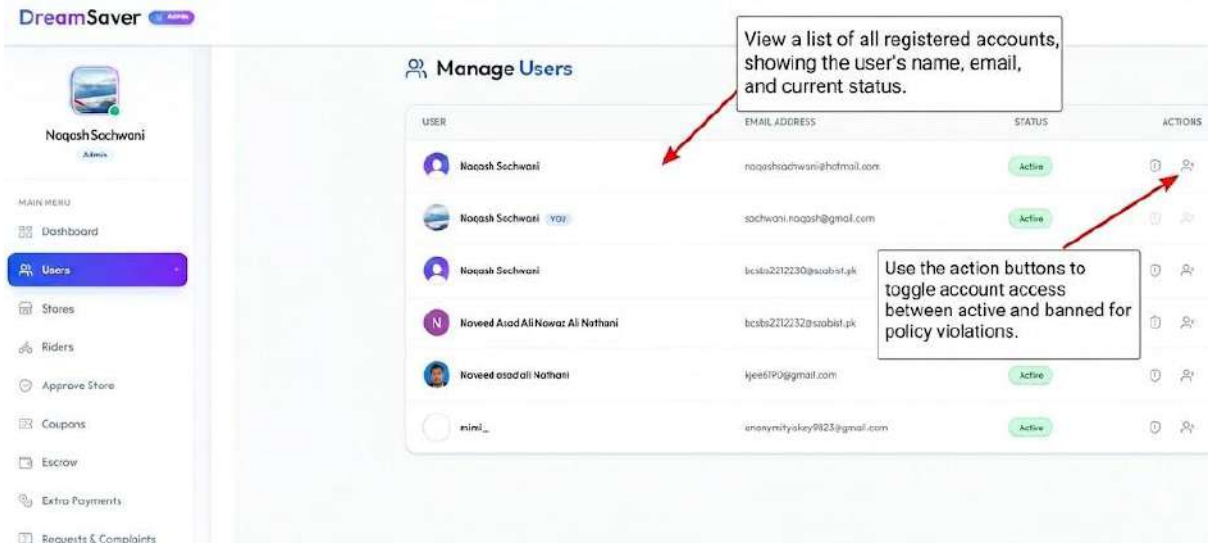


Figure 158: Admin User Management – Navigation

5.1.4.5 Admin Escrow Management

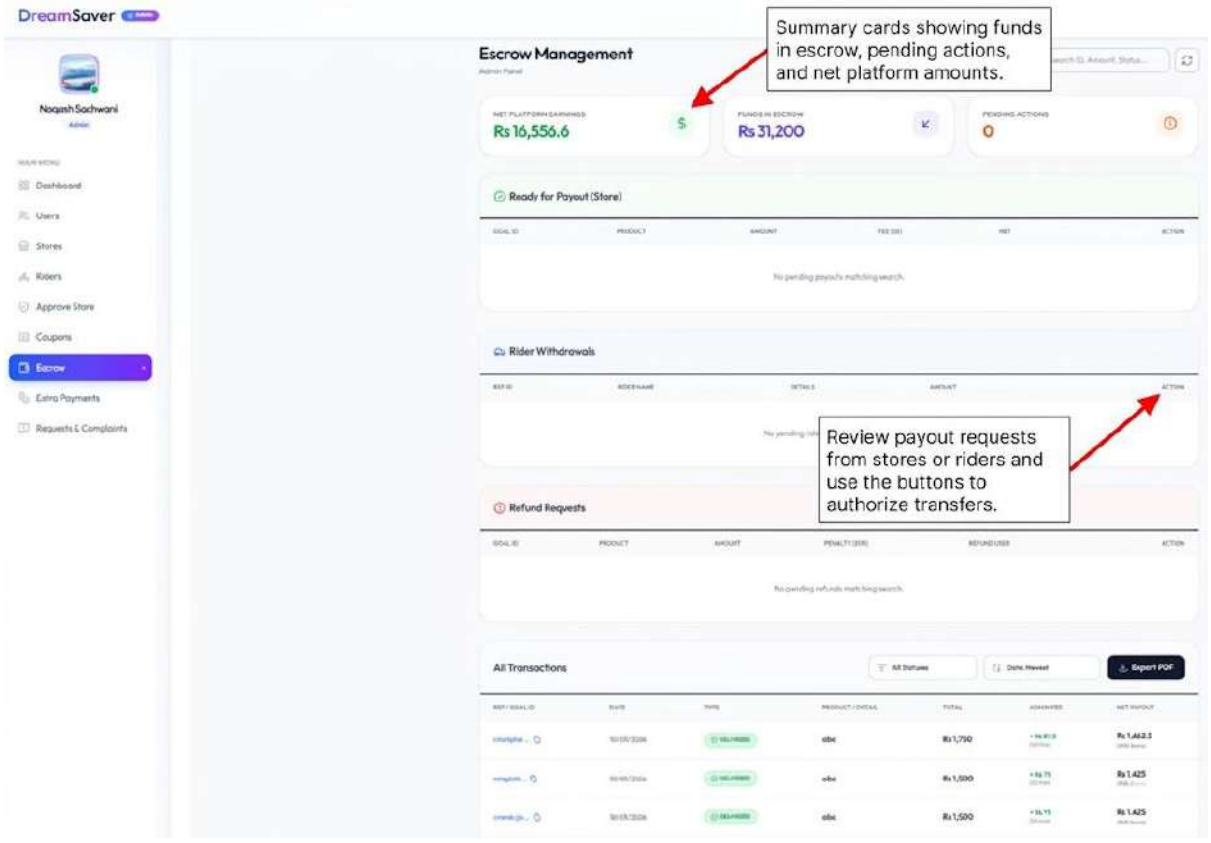


Figure 159: Admin Escrow Management – Navigation

5.1.4.6 Admin Dispute Management

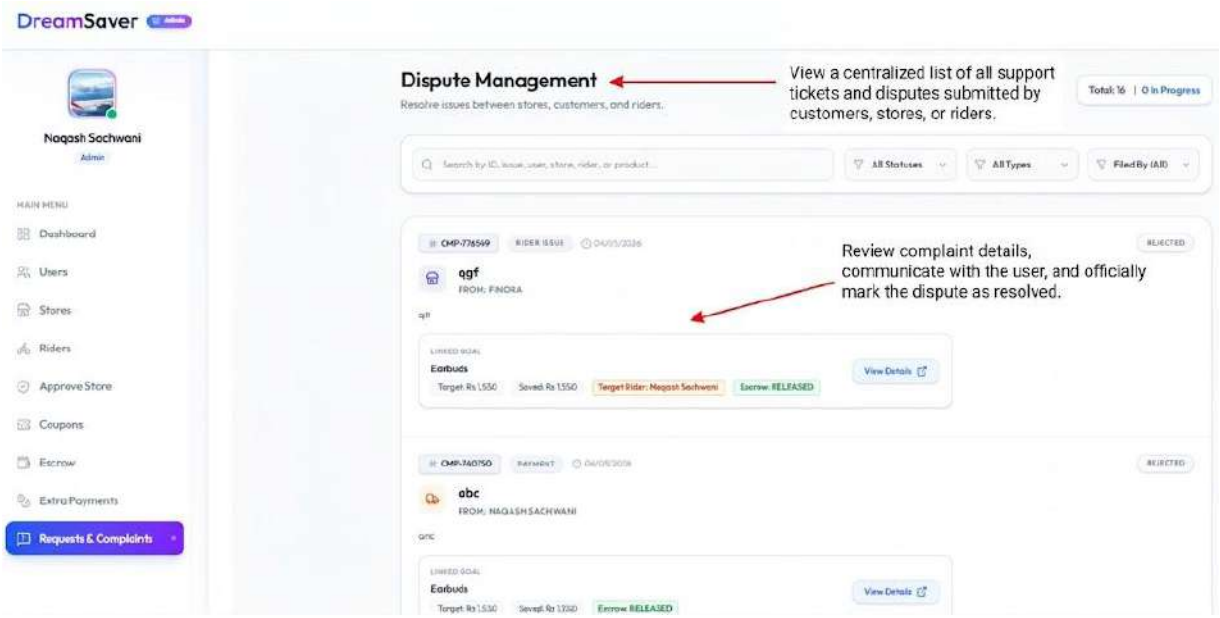


Figure 160: Admin Dispute Management – Navigation

5.1.4.7 Admin Rider Fleet Management

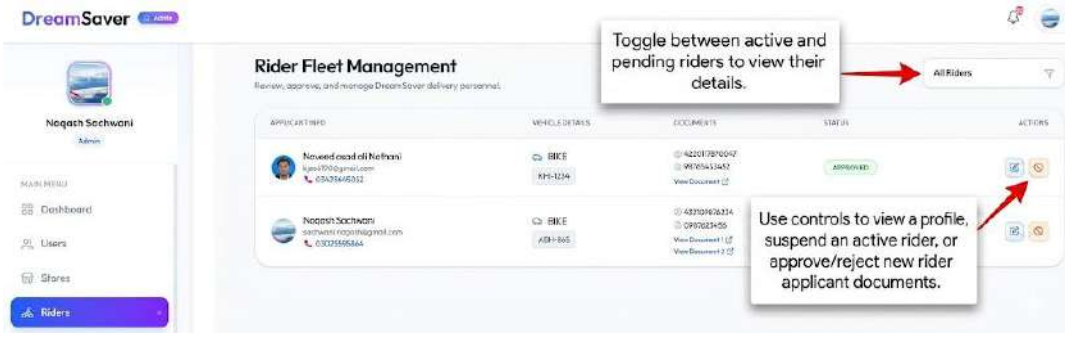


Figure 161: Admin Rider Fleet Management – Navigation

5.1.4.8 Admin Coupon Creation

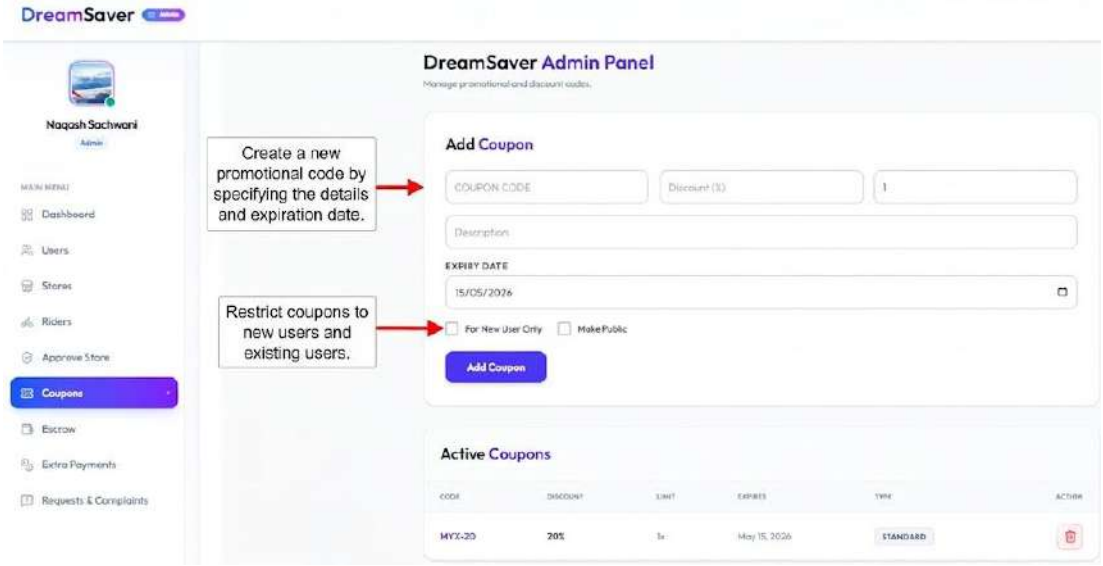


Figure 162: Admin Coupon Creation – Navigation

5.1.4.9 Admin Extra Payments

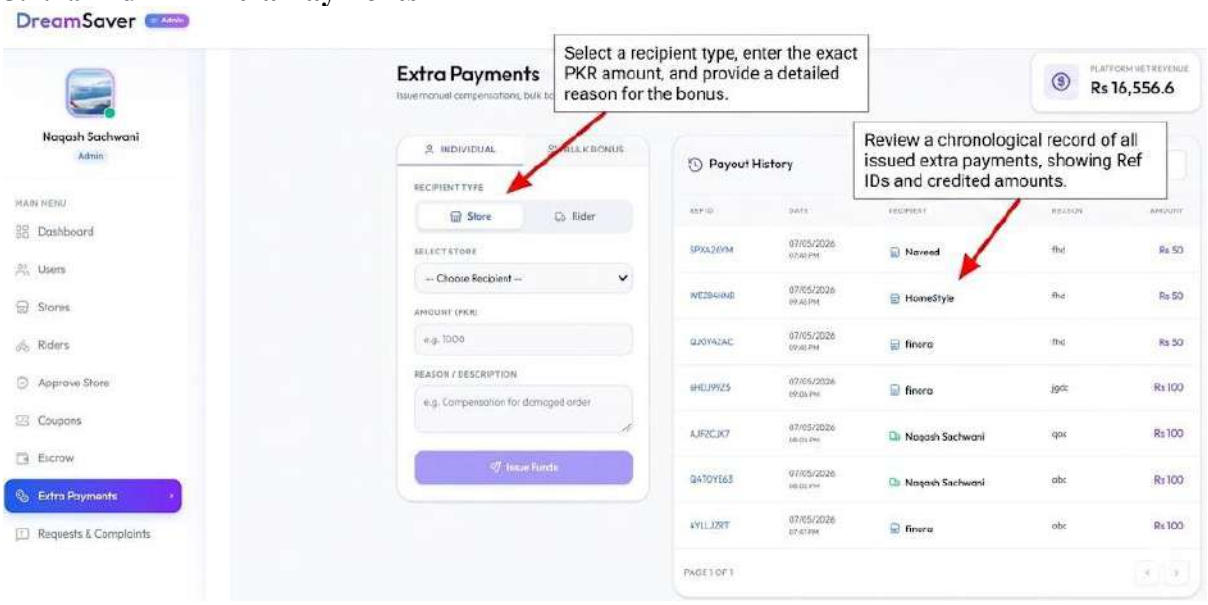


Figure 163: Admin Extra Payments – Navigation

5.2 Screens / Modules

5.2.1 User Interface

5.2.1.1 Sign Up

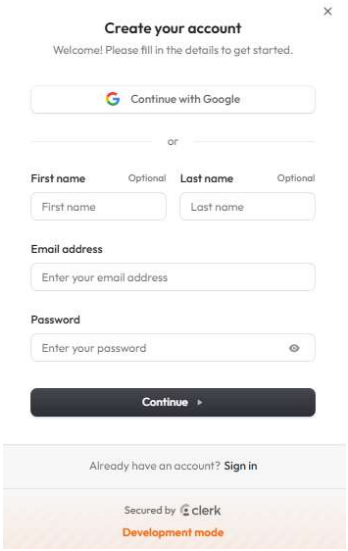


Figure 164: Sign Up – User Manual

5.2.1.2 Sign In

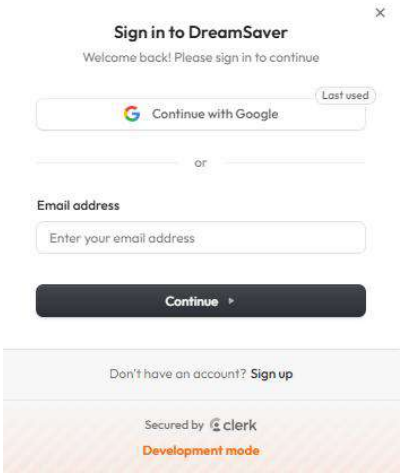


Figure 165: Sign In – User Manual

5.2.1.3 Home Page (Complete Overview)

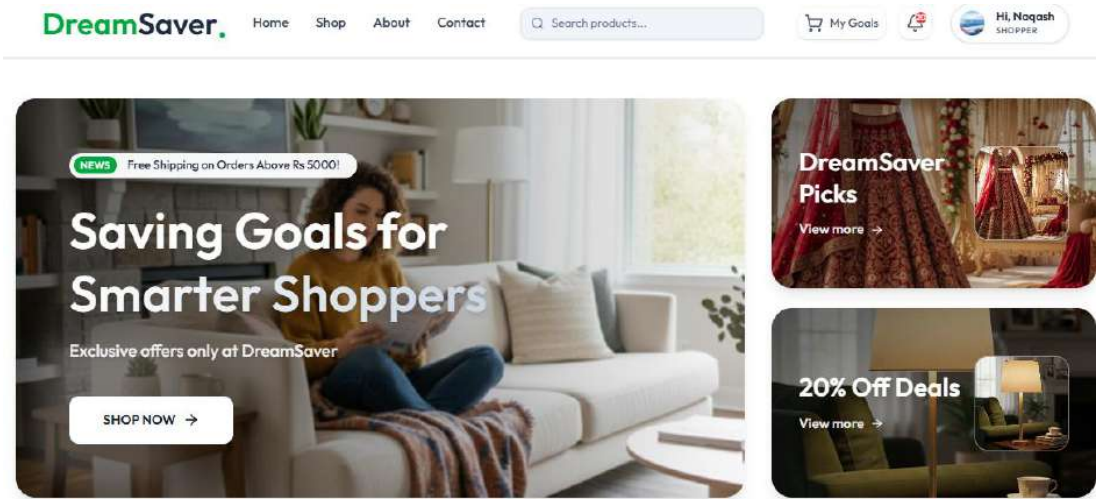


Figure 166: Home Page – User Manual

5.2.1.4 Shop

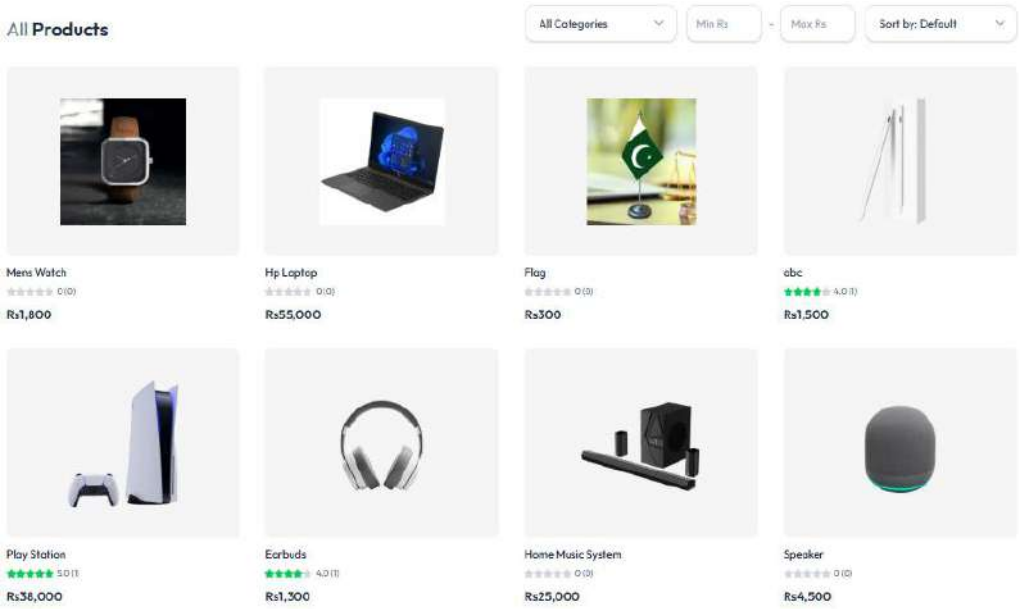


Figure 167: Shop – User Manual

5.2.1.5 About

Figure 168: About – User Manual

5.2.1.6 Contact

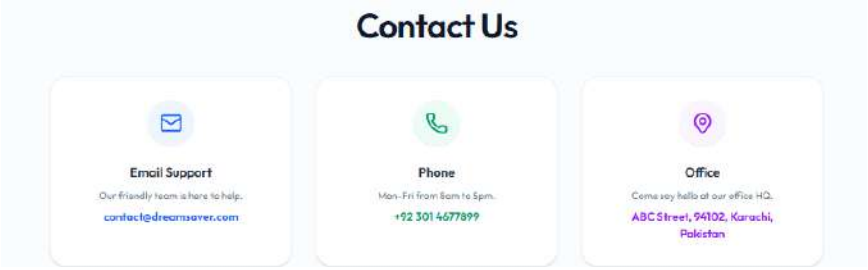


Figure 169: Contact – User Manual

5.2.1.7 Product

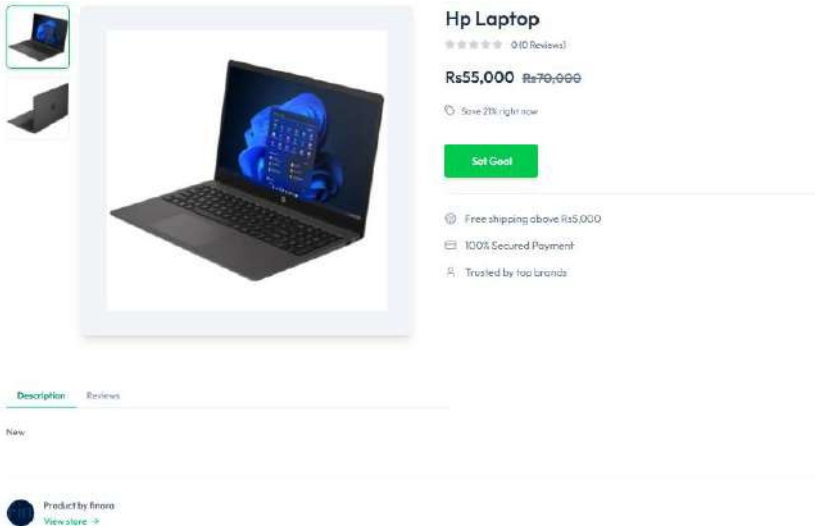


Figure 170: Product – User Manual

5.2.1.8 Set Goal

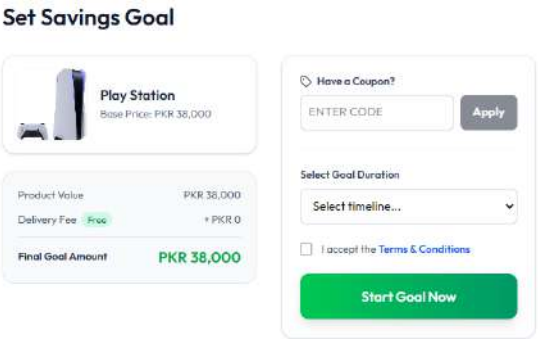


Figure 171: Set Goal – User Manual

5.2.1.9 Payment Process (Initialization & Gateway)

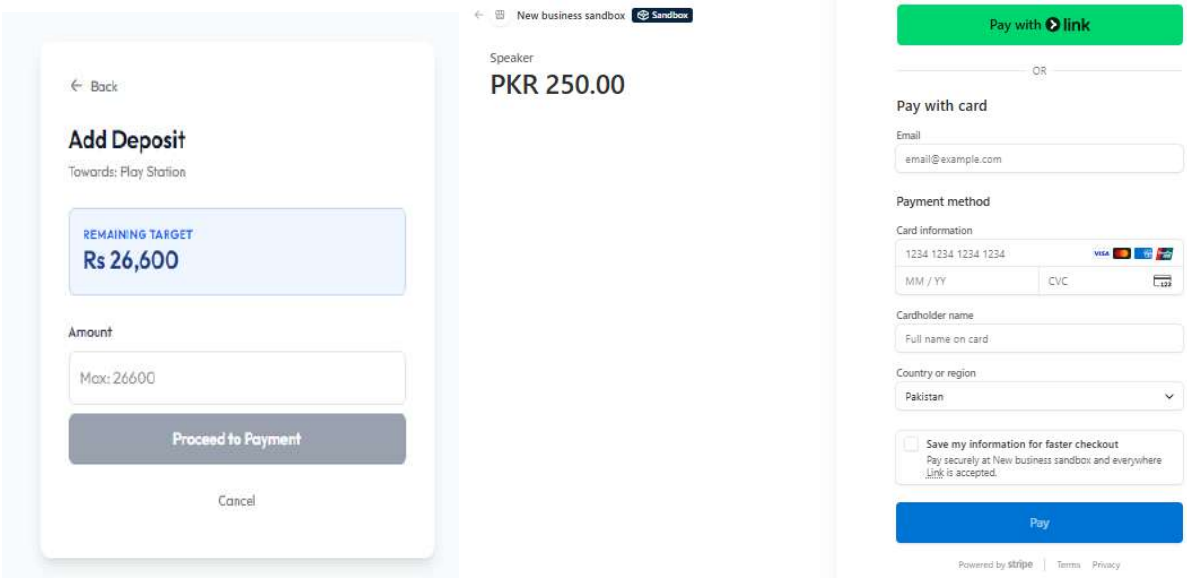


Figure 172: Payment Process – User Manual

5.2.1.10 Goal Progress

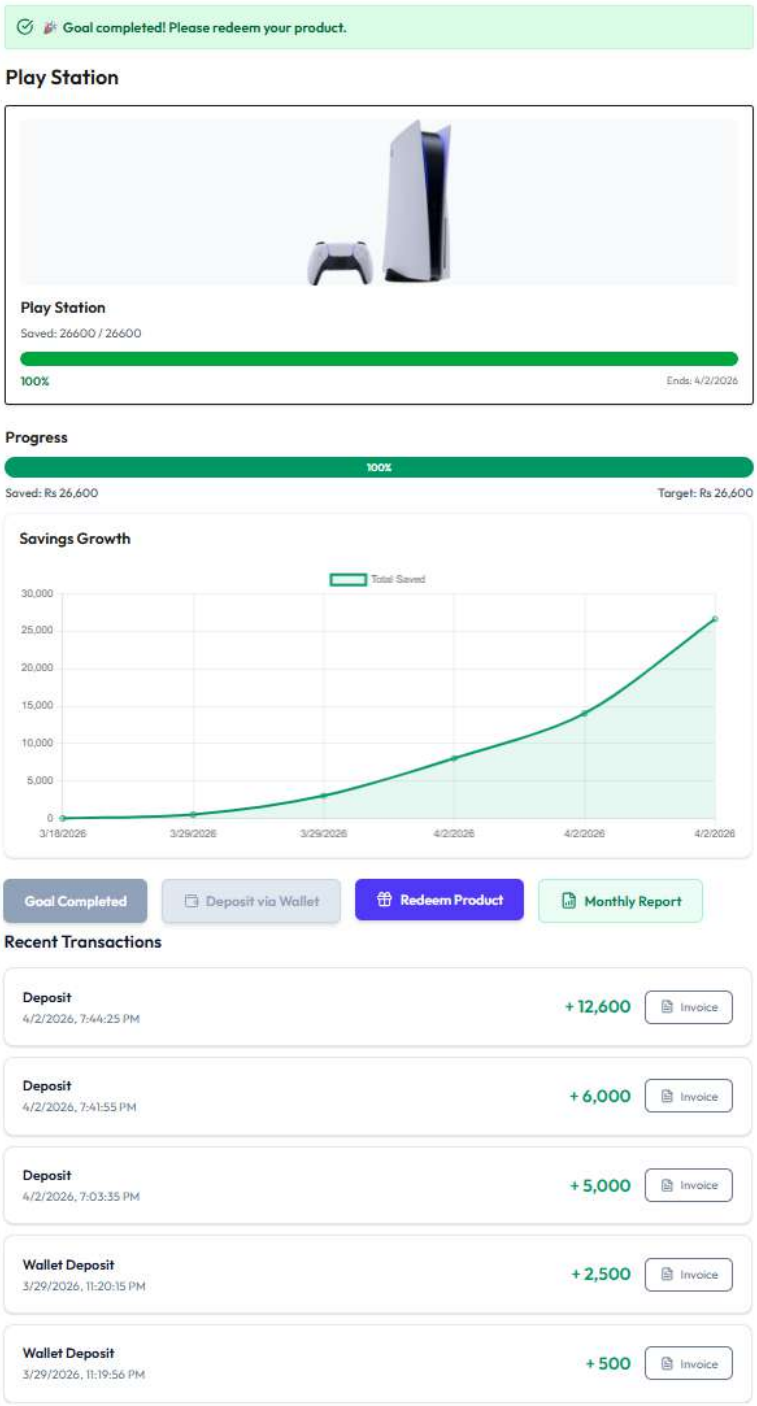


Figure 173: Goal Progress – User Manual

5.2.1.11 Goal Cart

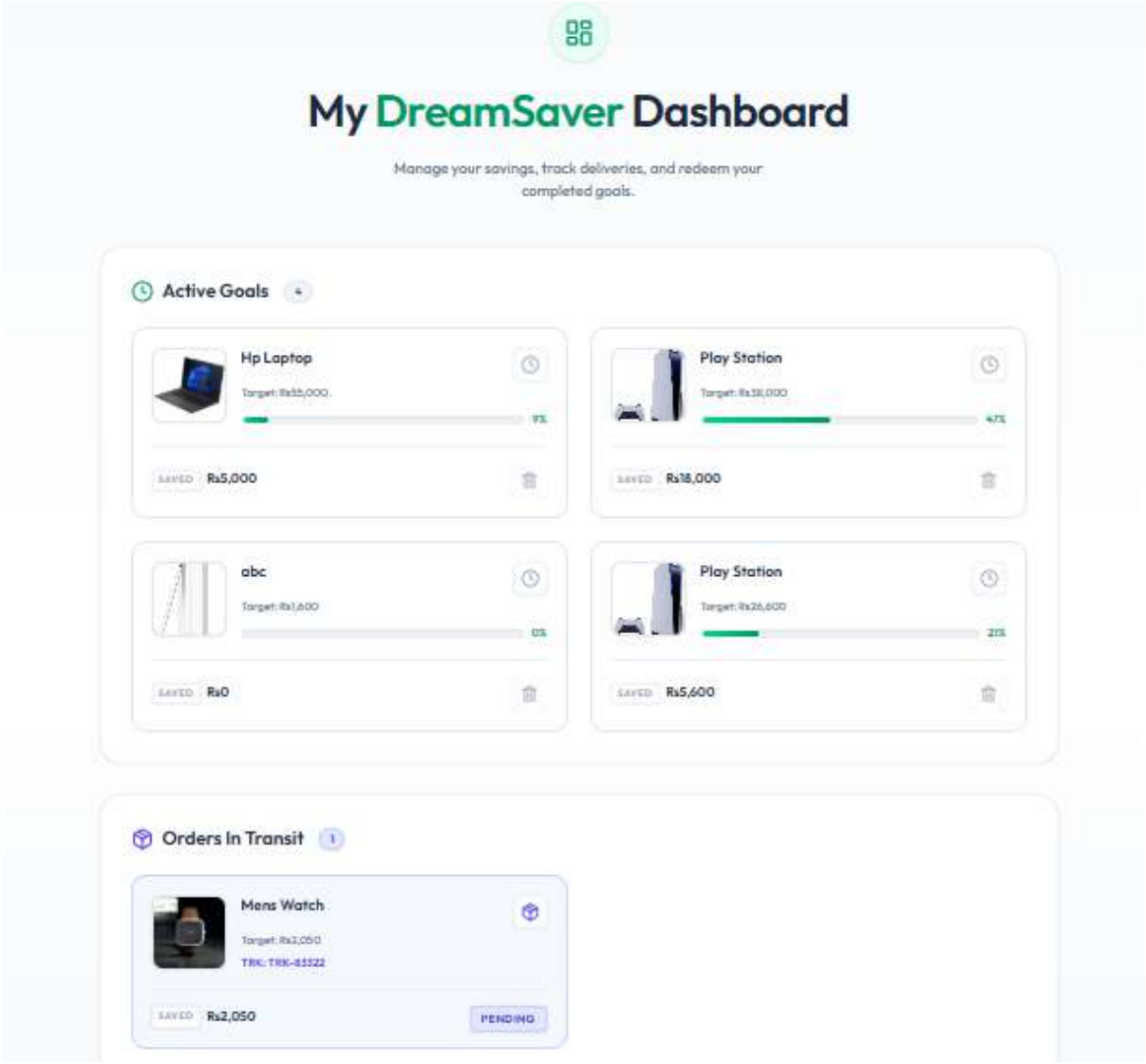


Figure 174: Goal Cart – User Manual

5.2.1.12 Wallet

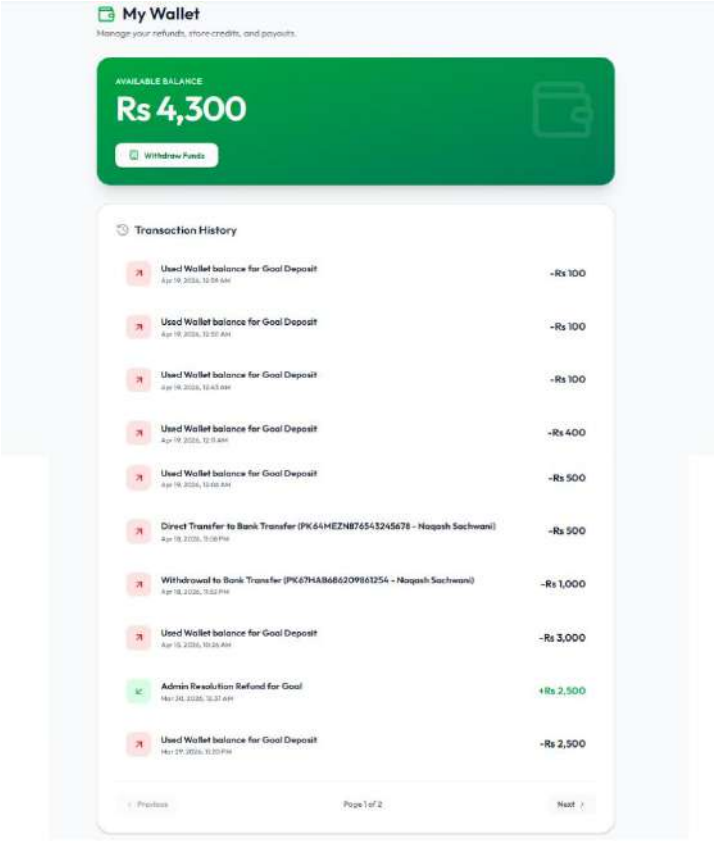


Figure 175: Wallet – User Manual

5.2.1.13 Support & Complaints

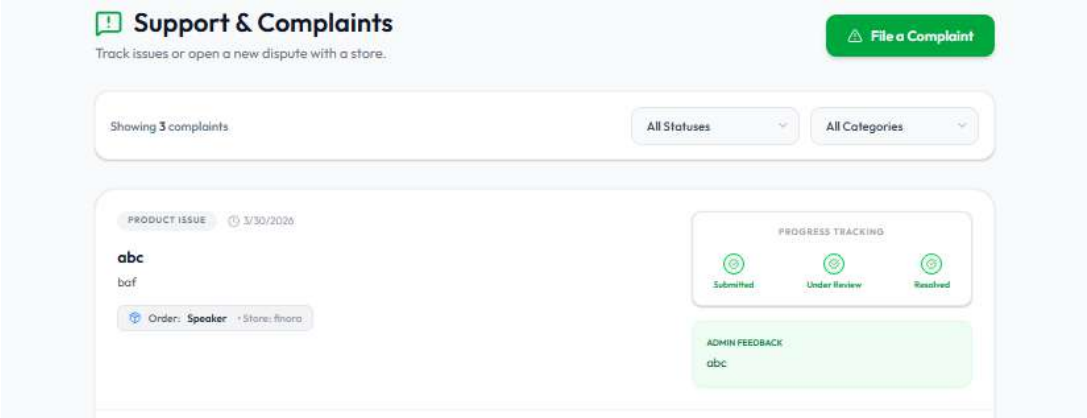


Figure 176: Support & Complaints – User Manual

5.2.1.14 Goal History

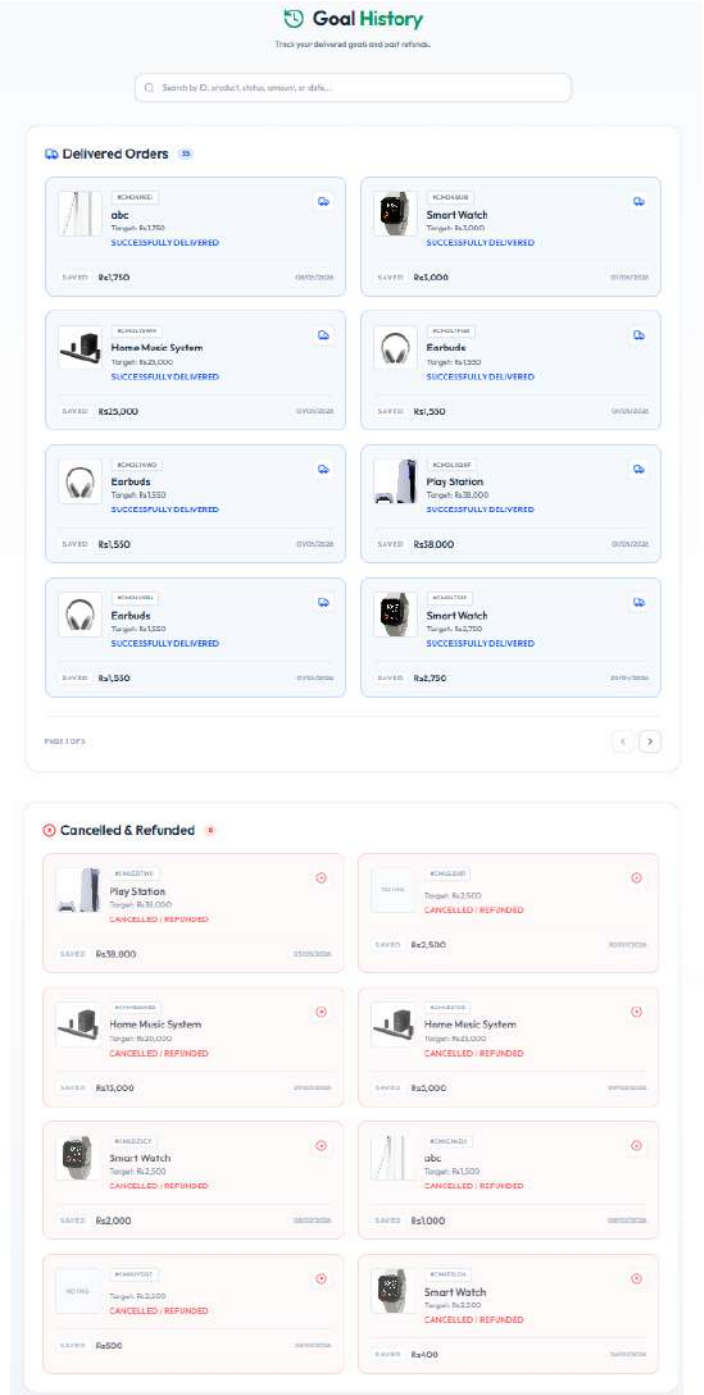


Figure 177: Goal History – User Manual

5.2.1.15 Tracking

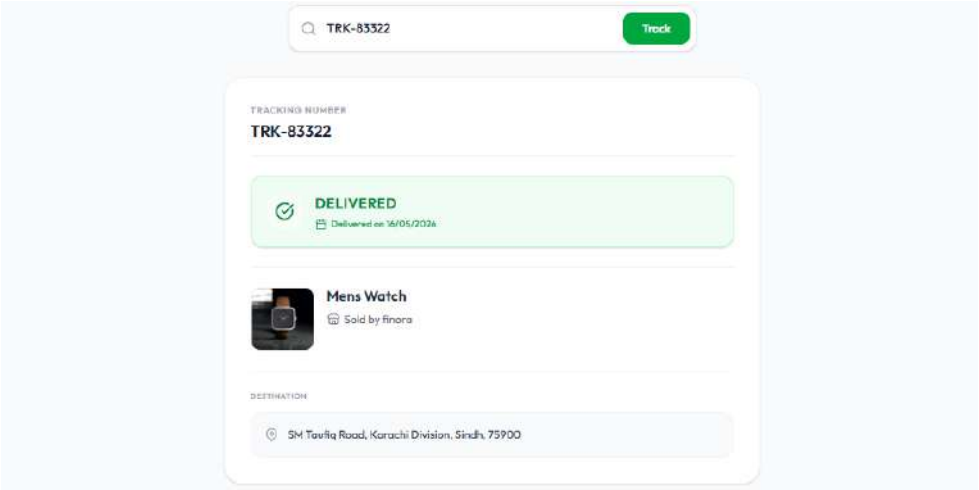


Figure 178: Tracking – User Manual

5.2.1.16 My Coupons

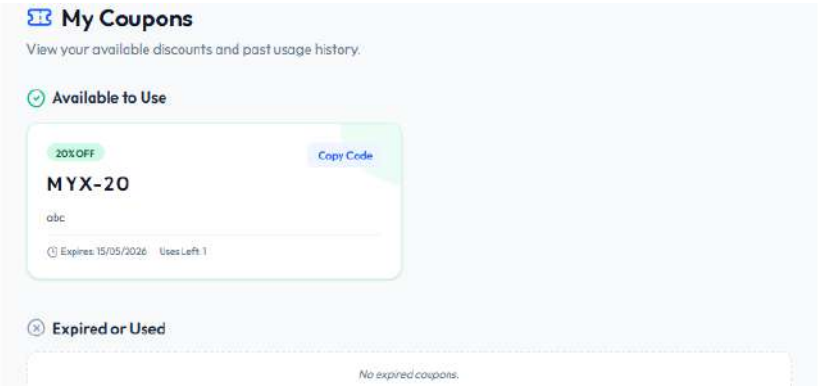


Figure 179: My Coupon – User Manual

5.2.1.17 Map Tracking

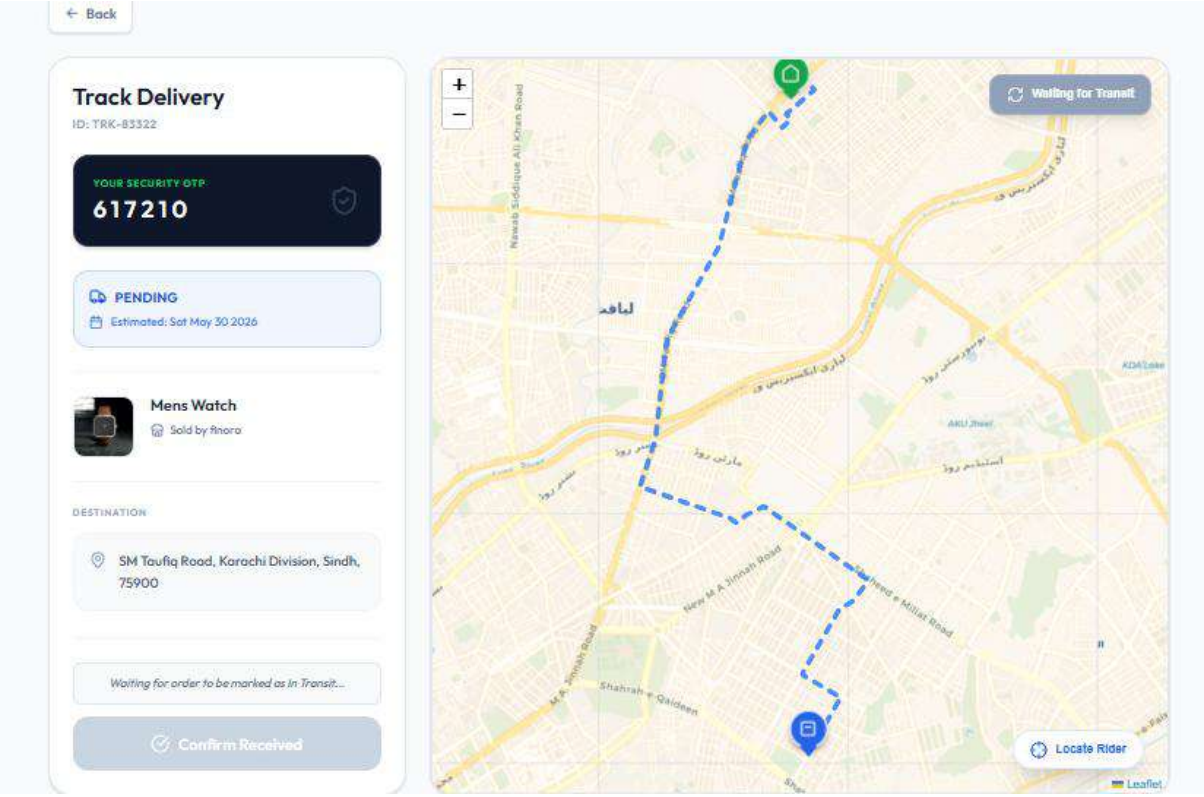


Figure 180: Map Tracking – User Manual

5.2.2 Rider Interface

5.2.2.1 Rider Registration

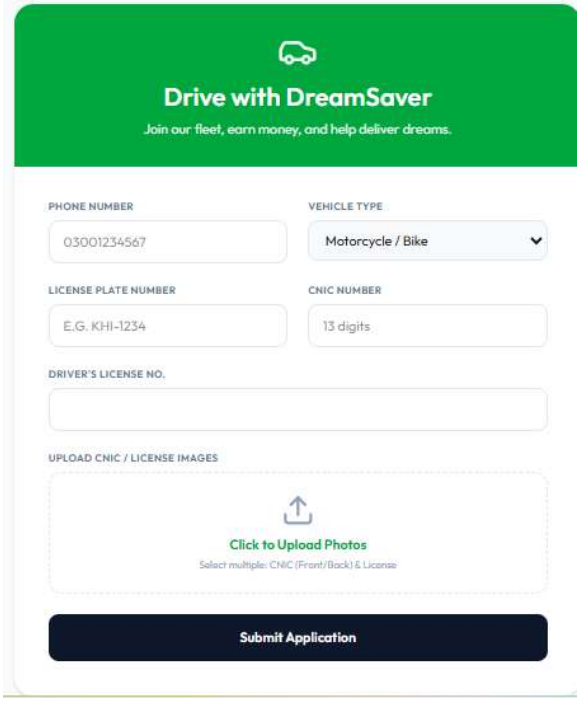


Figure 181: Rider Registration – User Manual

5.2.2.2 Rider Dashboard

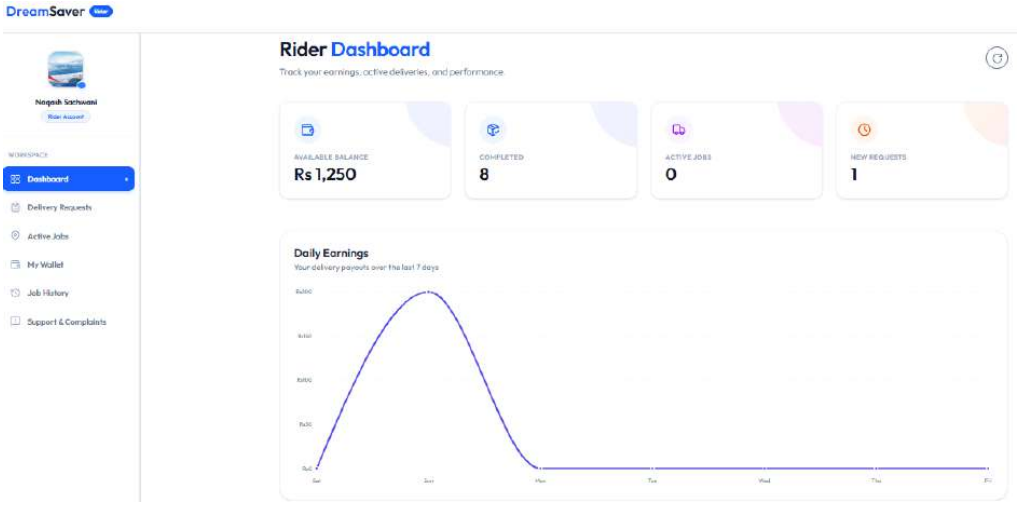


Figure 182: Rider Dashboard – User Manual

5.2.2.3 Rider Delivery Requests

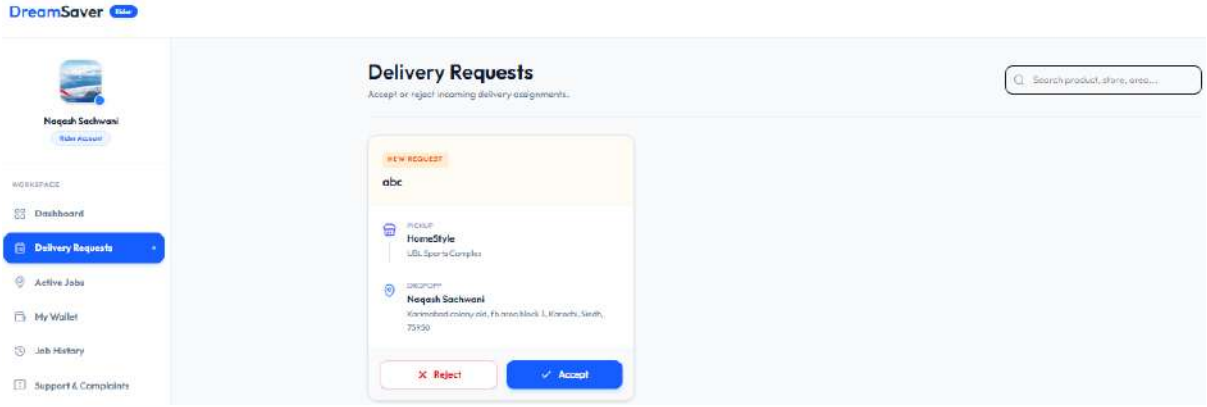


Figure 183: Rider Delivery Request – User Manual

5.2.2.4 Rider Active Jobs

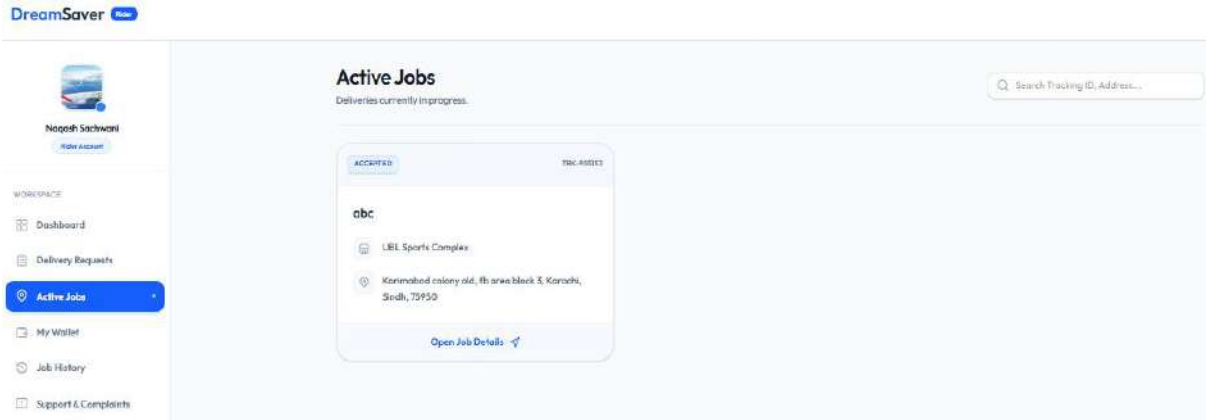


Figure 184: Rider Active Jobs – User Manual

5.2.2.5 Rider Wallet

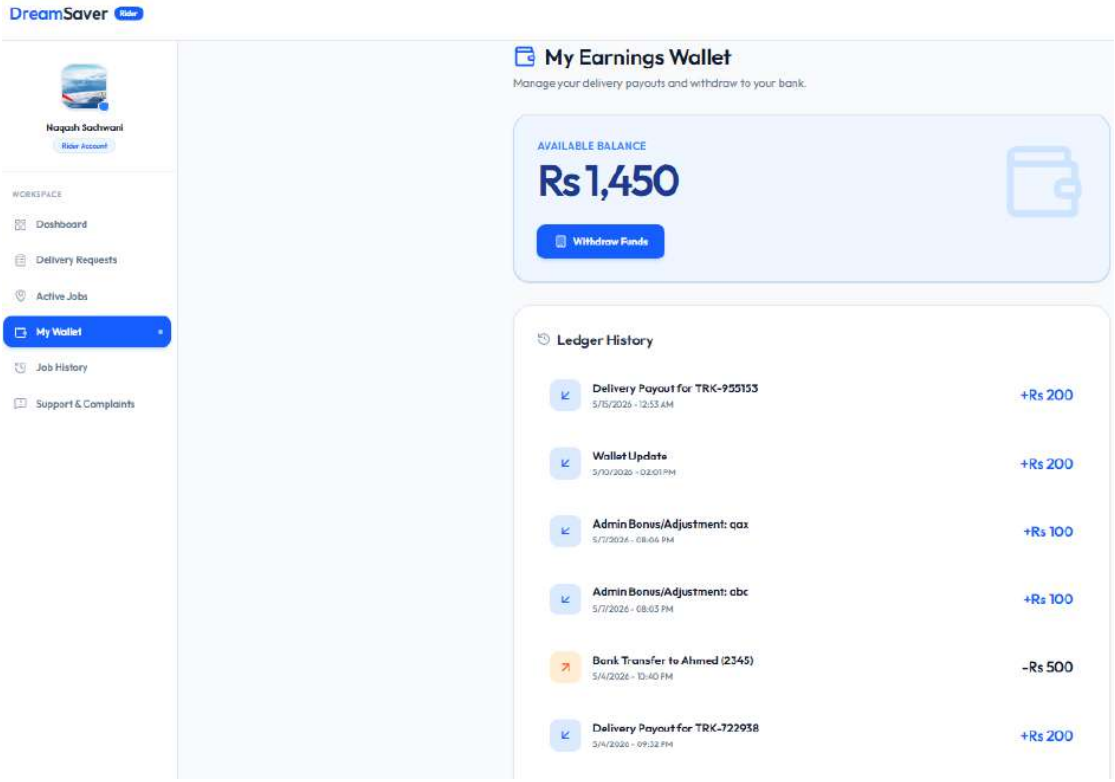


Figure 185: Rider Wallet – User Manual

5.2.2.6 Rider Job History

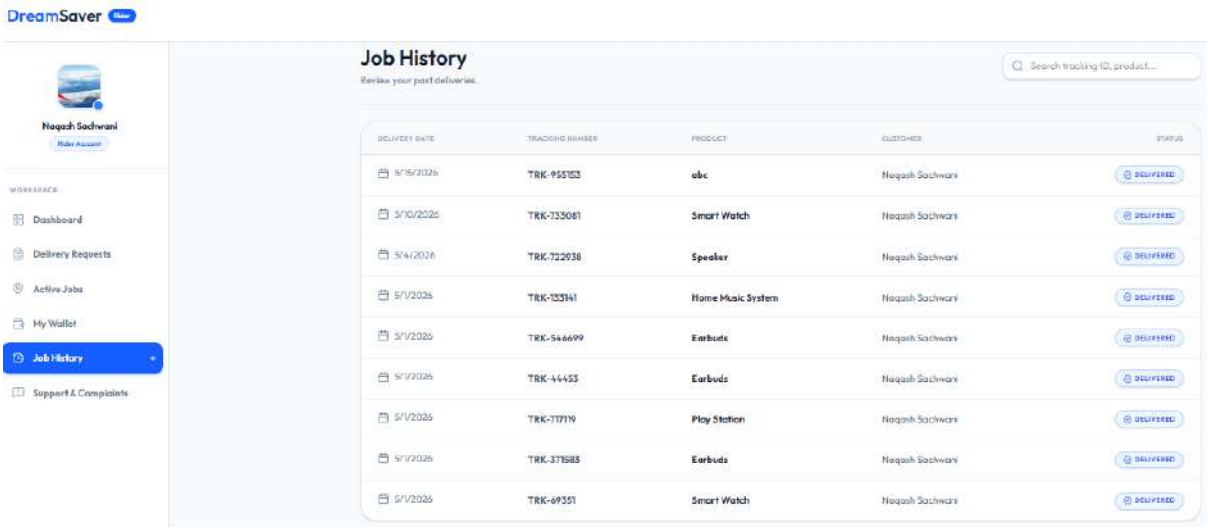


Figure 186: Rider Job History – User Manual

5.2.2.7 Rider Support & Complaints

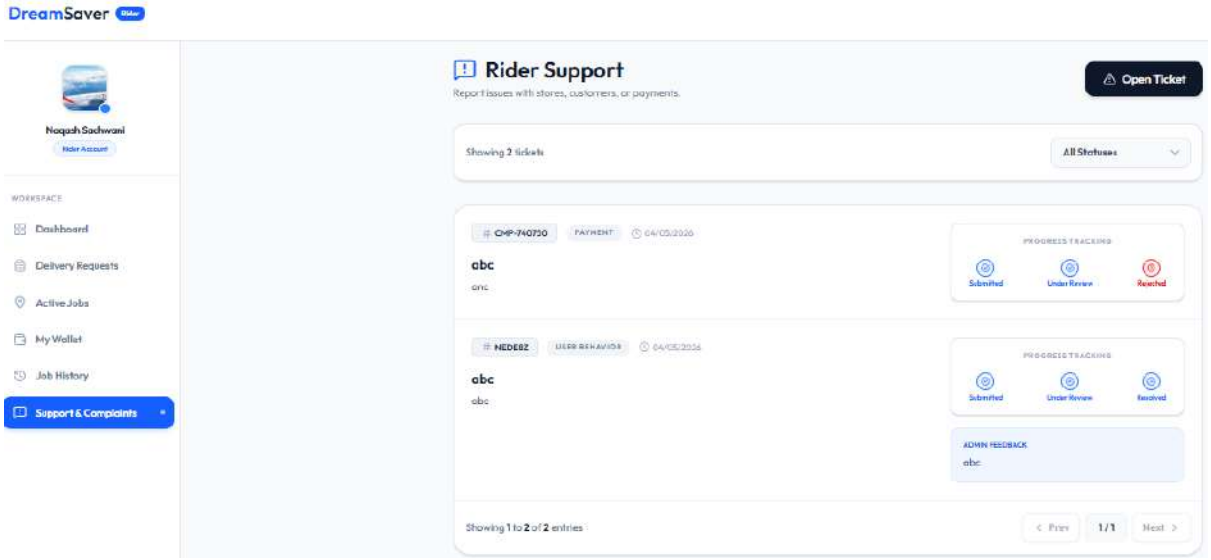


Figure 187: Rider Support & Complaints – User Manual

5.2.3 Store Interface

5.2.3.1 Store Creation Form

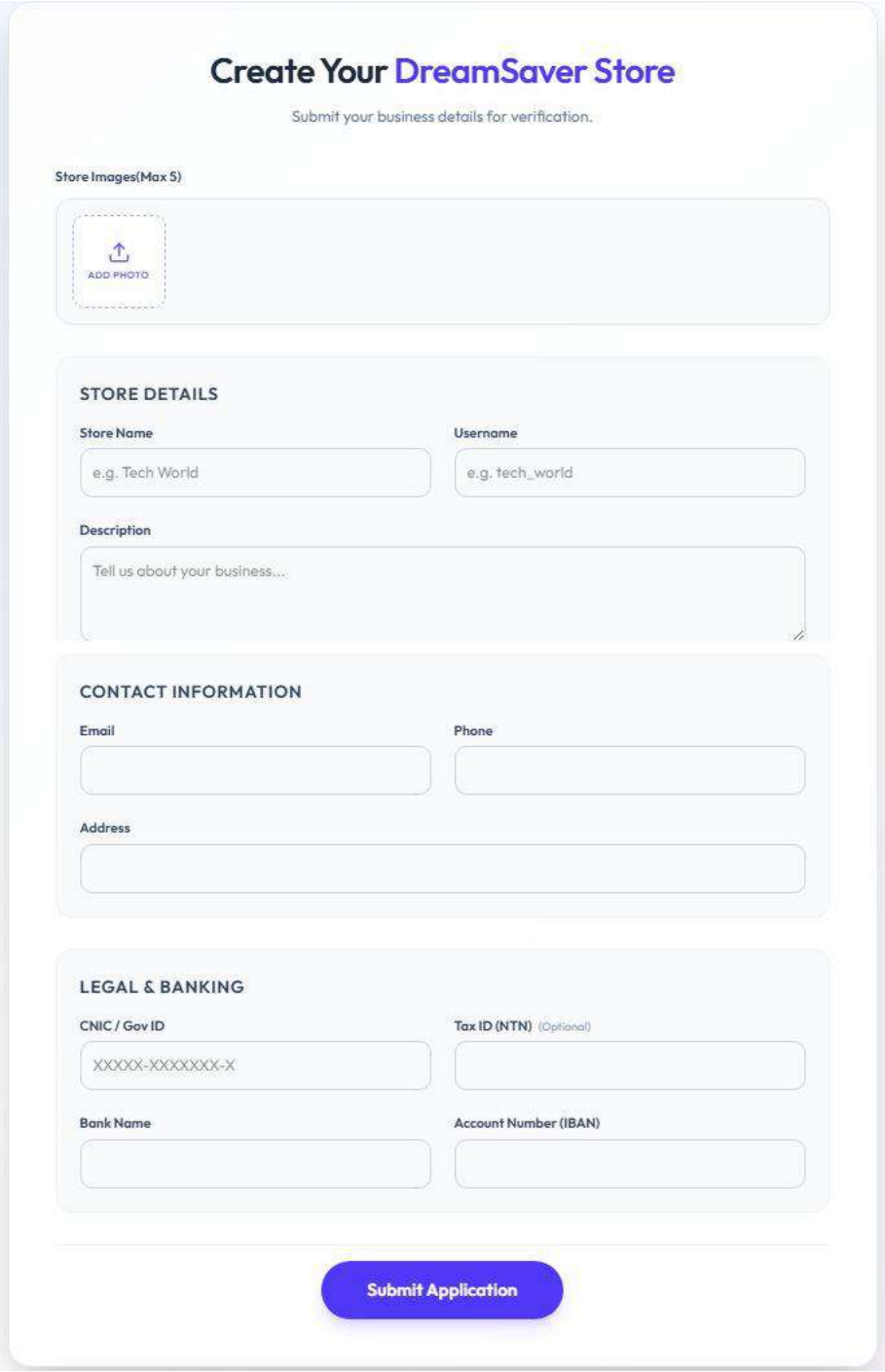


Figure 188: Store Creation Form – User Manual

5.2.3.2 Store Dashboard

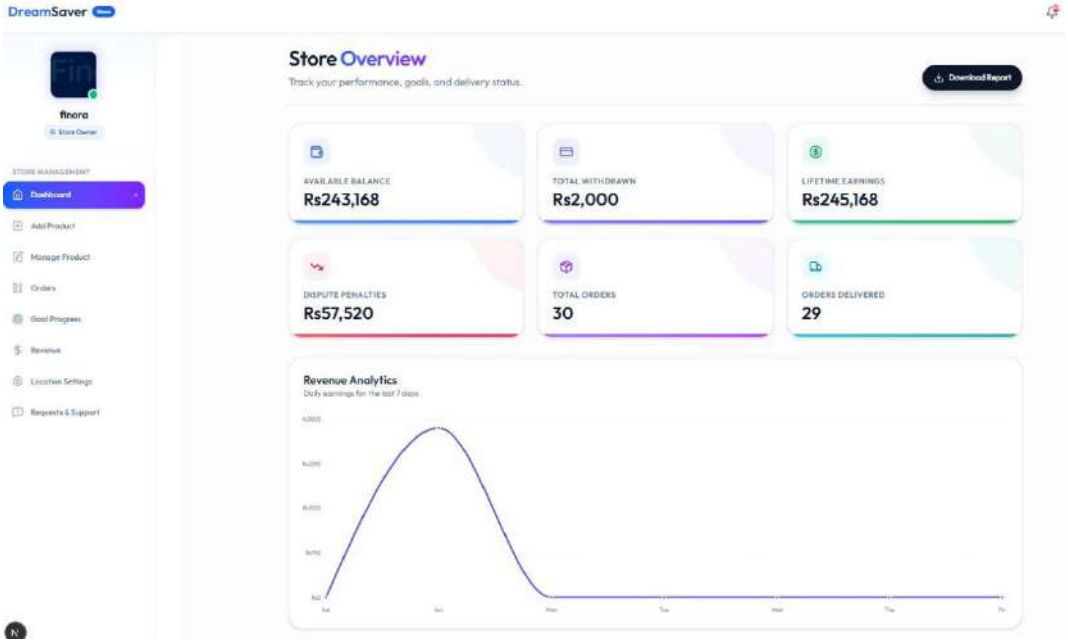


Figure 189: Store Dashboard – User Manual

5.2.3.3 Store Add Product

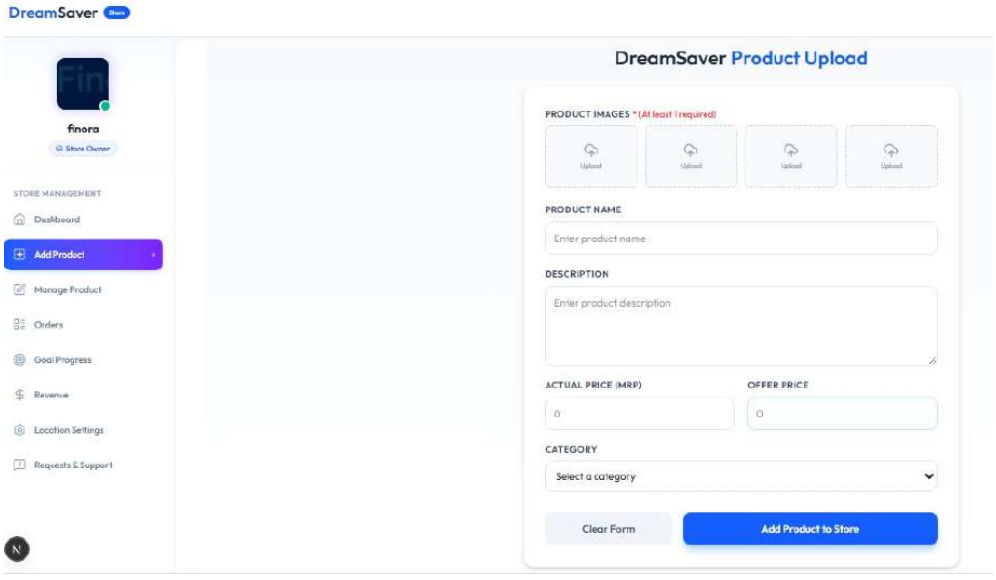


Figure 190: Store Add Product – User Manual

5.2.3.4 Store Manage Product

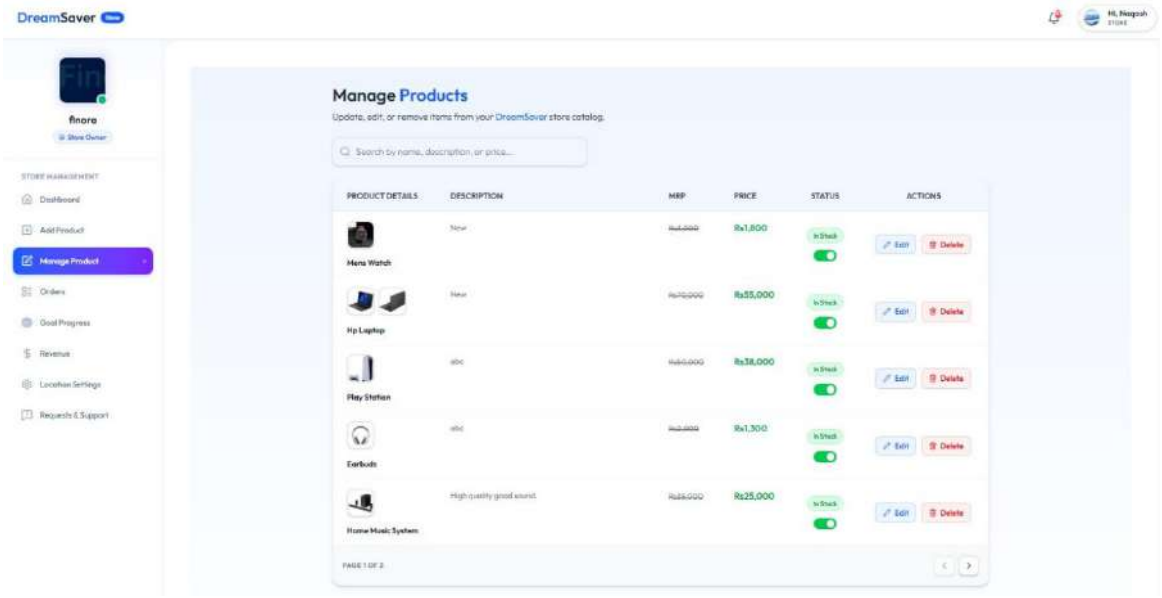


Figure 191: Store Manage Product – User Manual

5.2.3.5 Store Order

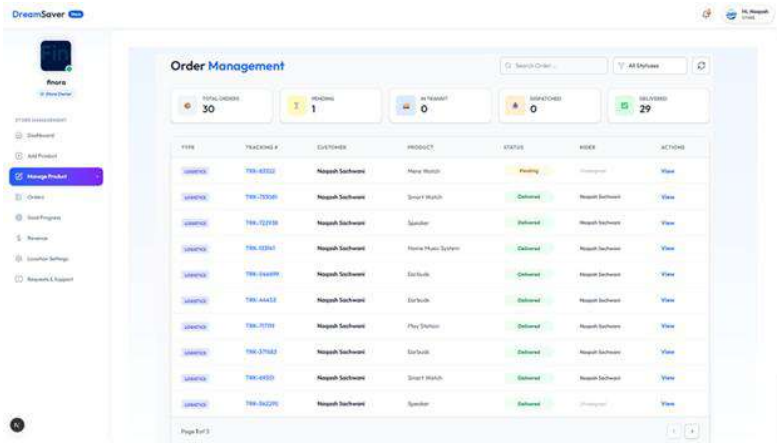


Figure 192: Store Order – User Manual

5.2.3.6 Store Location Setting

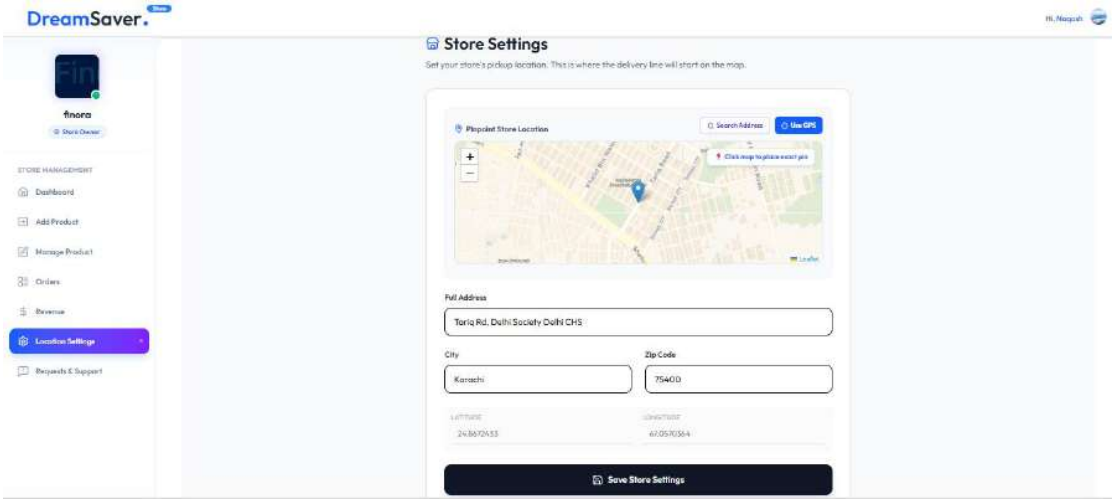


Figure 193: Store Location Setting – User Manual

5.2.3.7 Store Request & Support

Figure 194: Store Request & Support – User Manual

5.2.3.8 Store Goal Progress

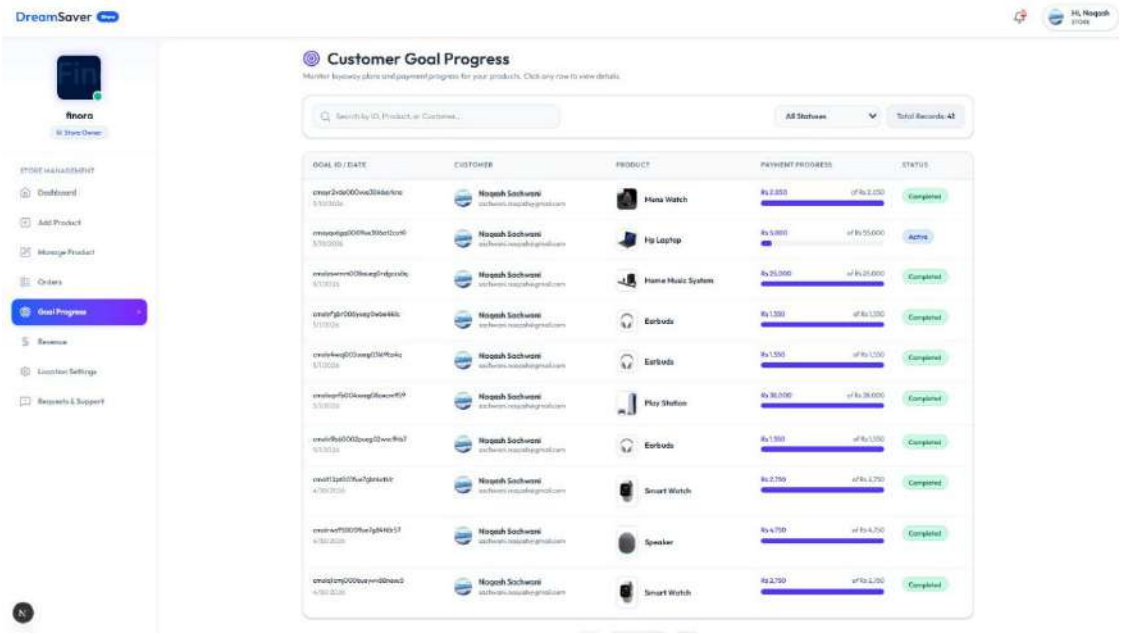


Figure 195: Store Goal Progress – User Manual

5.2.3.9 Store Revenue

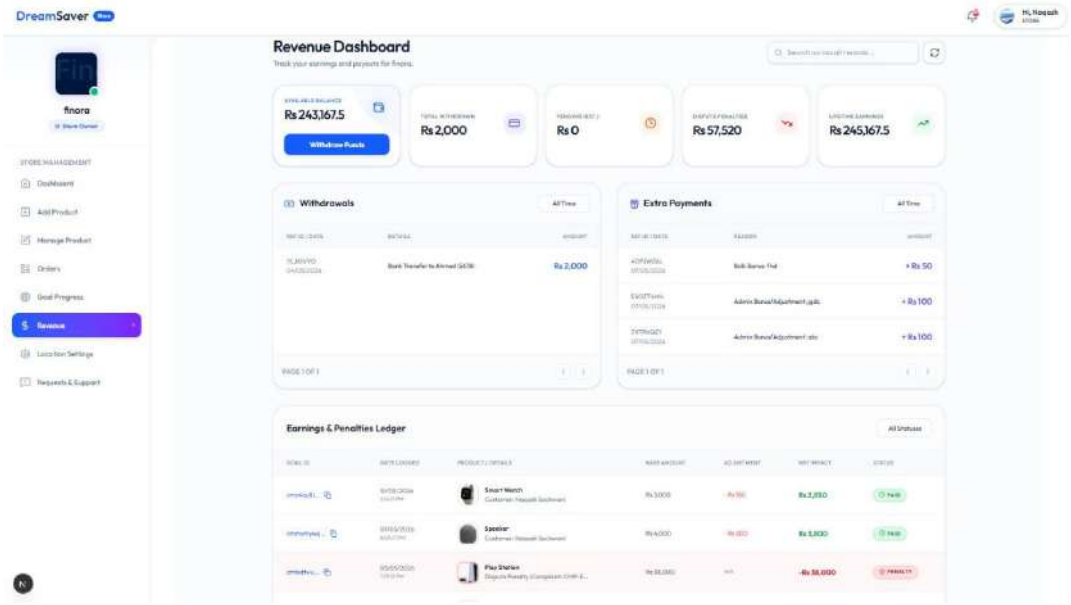


Figure 196: Store Revenue – User Manual

5.2.4 Admin Interface

5.2.4.1 Admin Dashboard

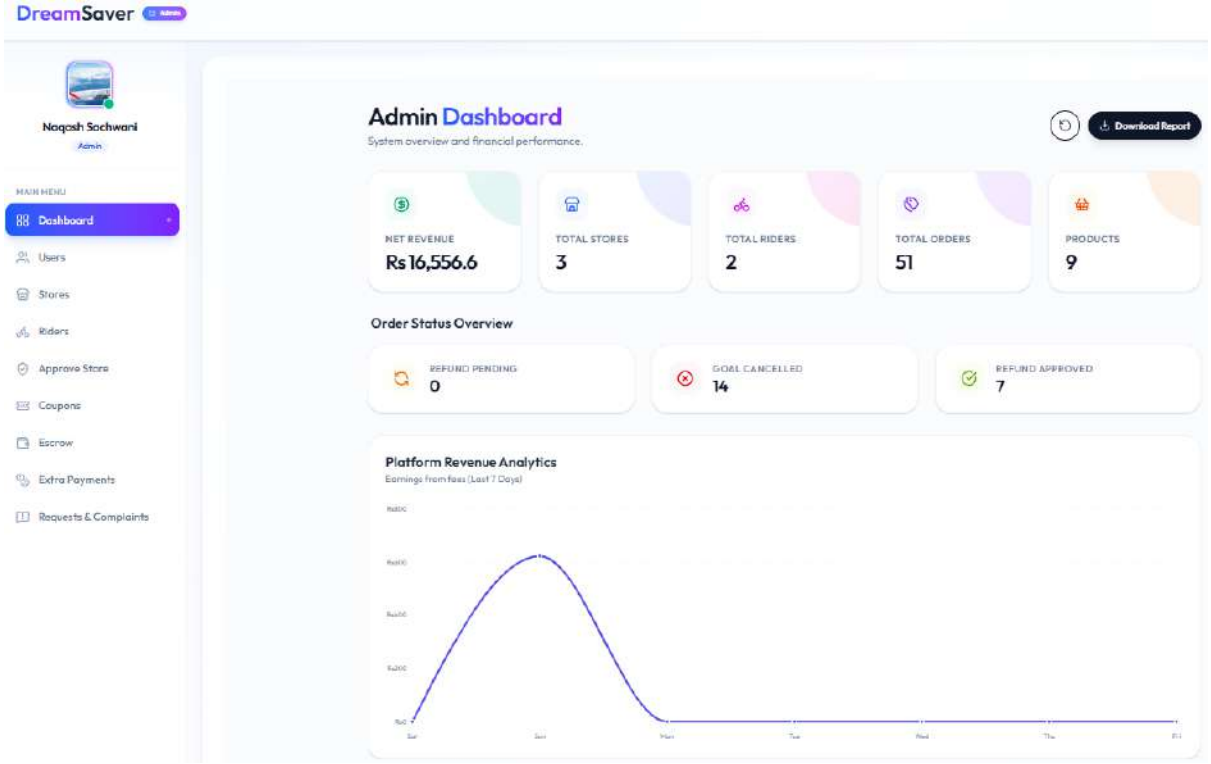


Figure 197: Admin Dashboard – User Manual

5.2.4.2 Admin Store Management

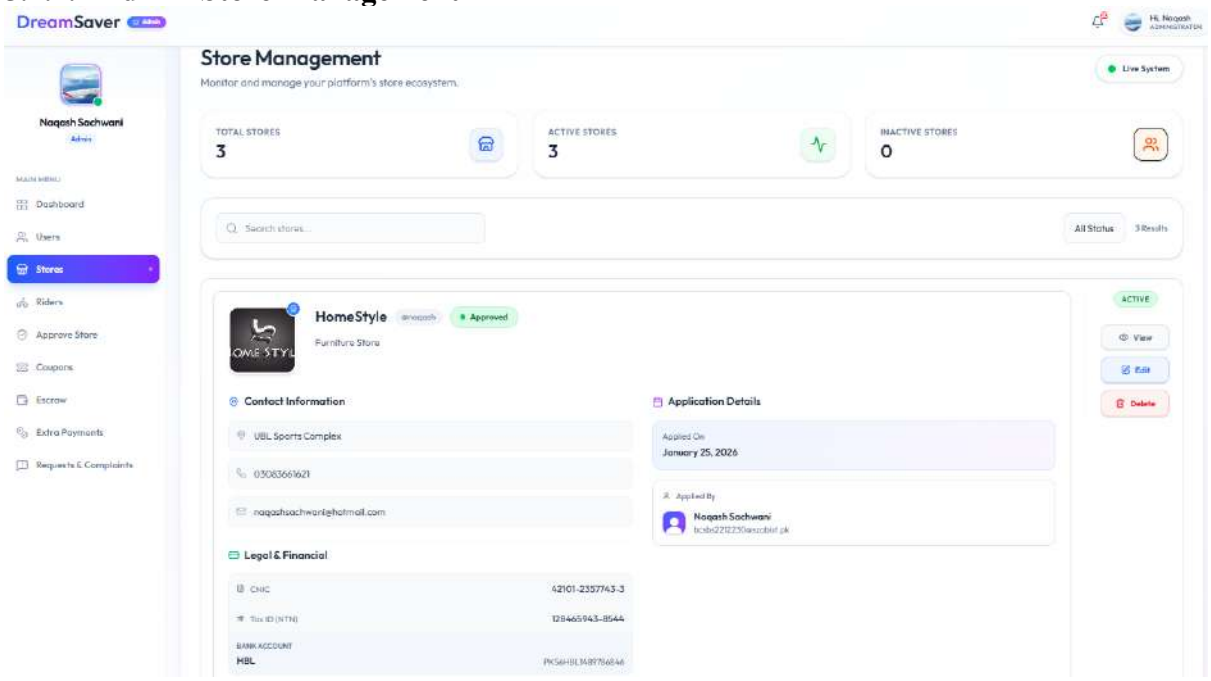


Figure 198: Admin Store Management– User Manual

5.2.4.3 Admin Store Approval

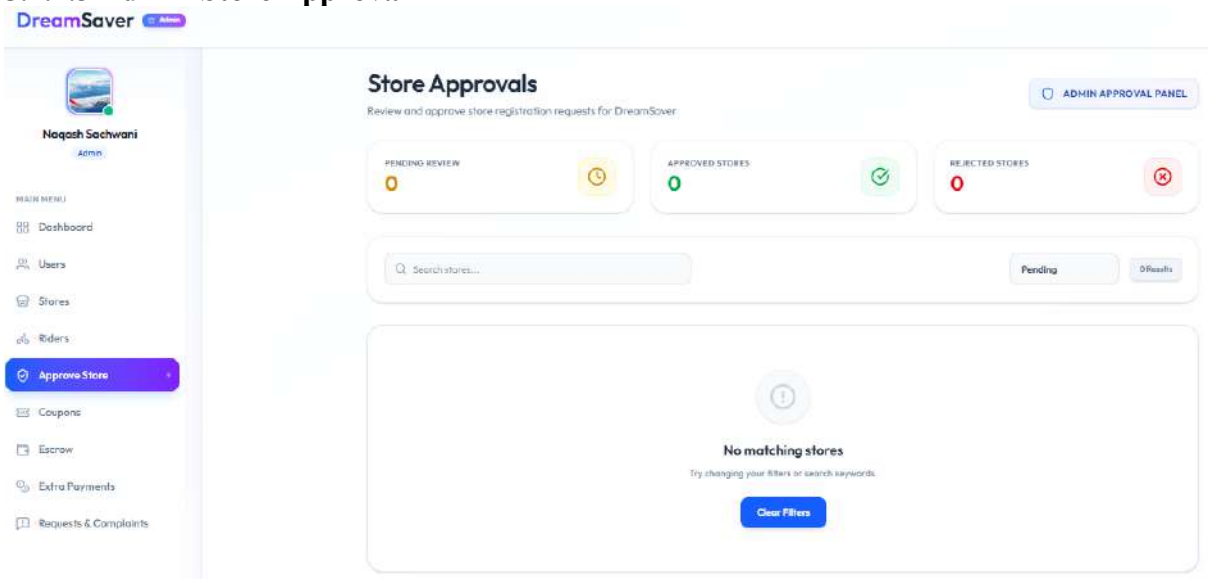


Figure 199: Admin Store Approval – User Manual

5.2.4.4 Admin User Management

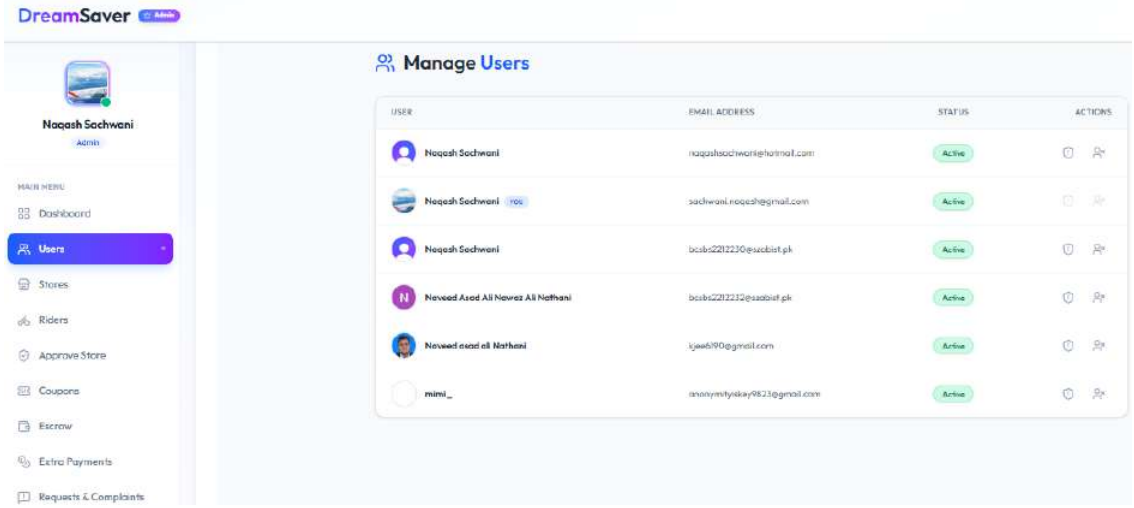


Figure 200: Admin User Management – User Manual

5.2.4.5 Admin Escrow Management

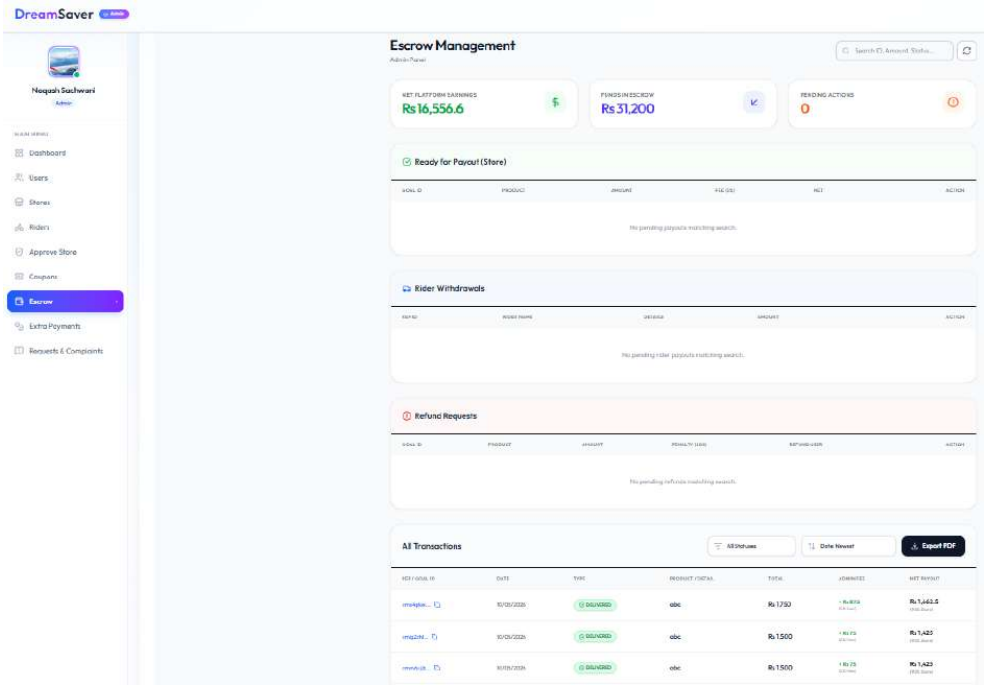


Figure 201: Admin Escrow Management – User Manual

5.2.4.6 Admin Dispute Management

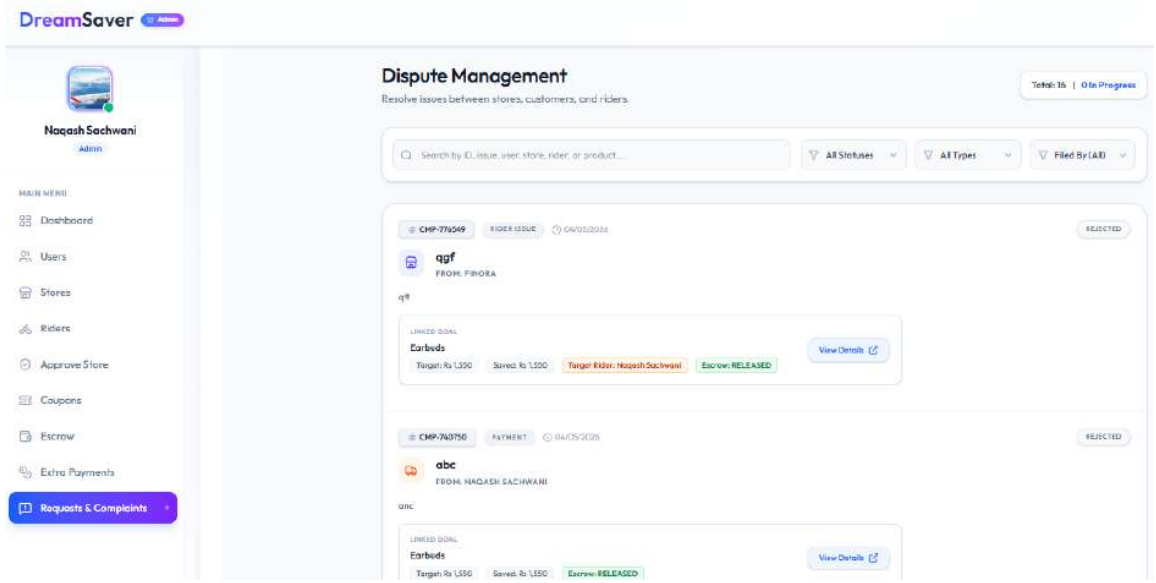


Figure 202: Admin Dispute Management – User Manual

5.2.4.7 Admin Rider Fleet Management

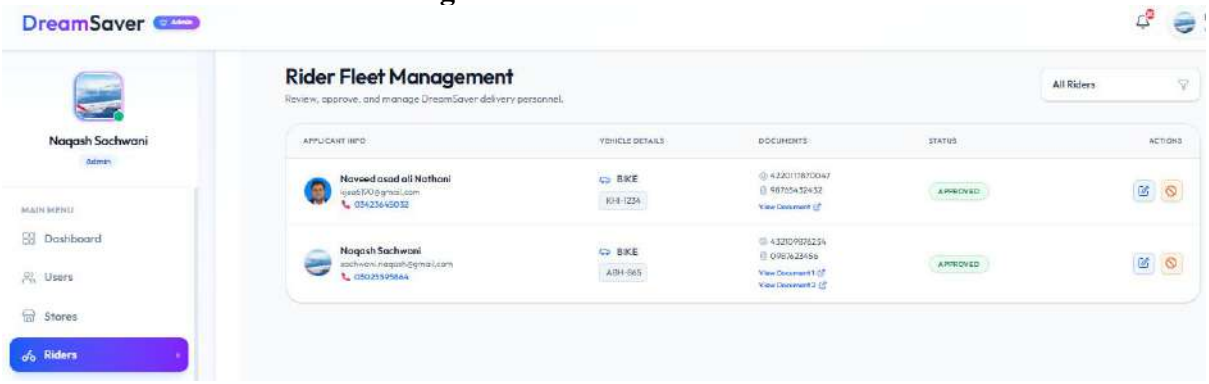


Figure 203: Admin Rider Fleet Management – User Manual

5.2.4.8 Admin Coupon Creation

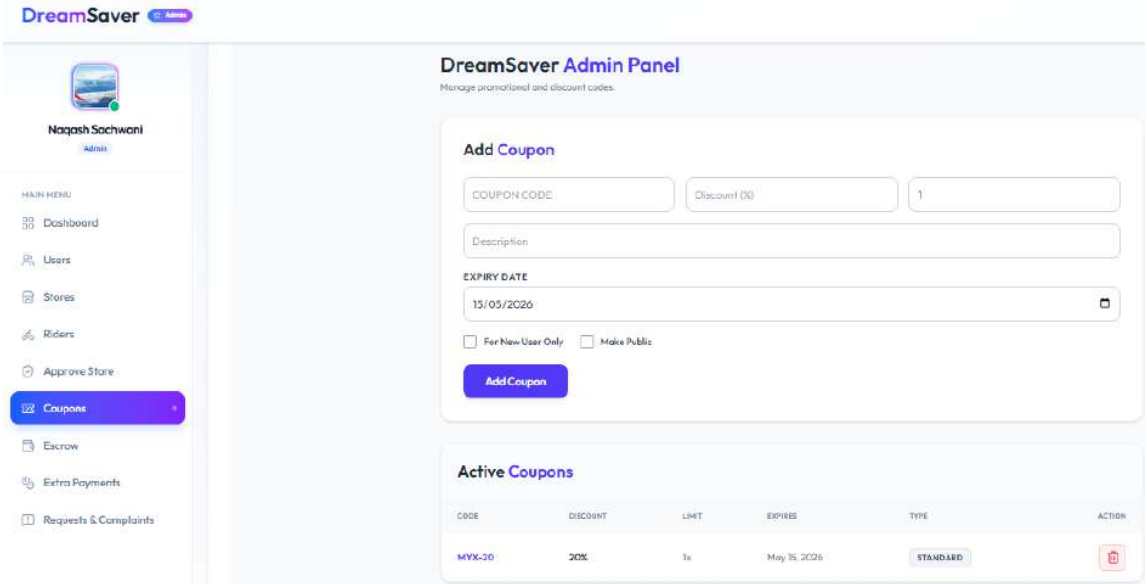


Figure 204: Admin Coupon Creation – User Manual

5.2.4.9 Admin Extra Payments

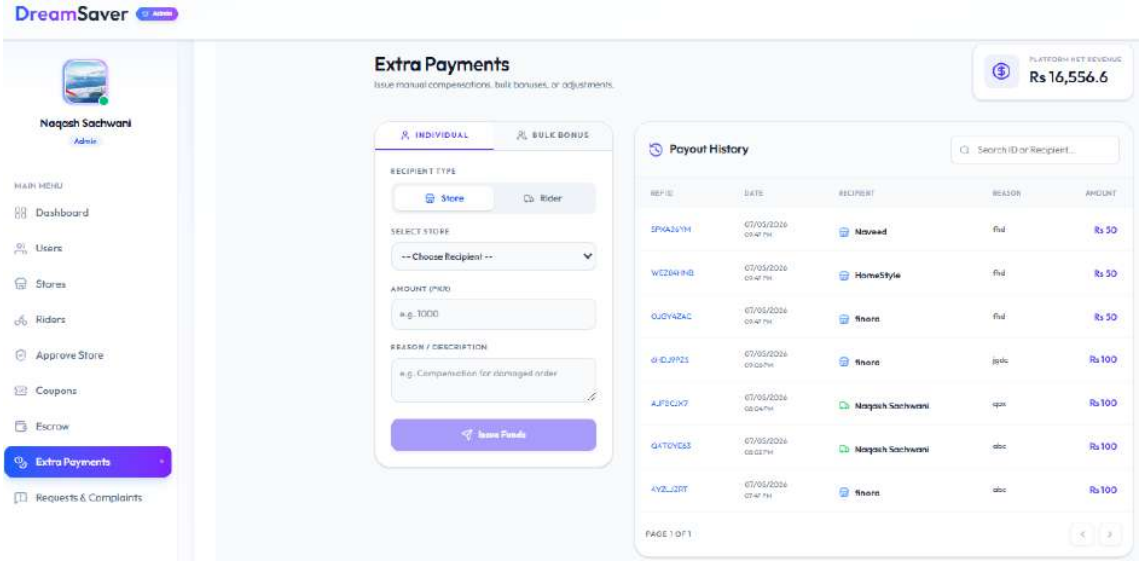


Figure 205: Admin Extra Payments – User Manual

6. How to Use the System

6.1 Basic Workflow

The DreamSaver platform's workflow is based on the collaboration of three parties: Customers, Stores, and Riders, who are supervised by the Admin.

1. Log into the System: Log in to the DreamSaver web portal.
2. Authentication: Register for an account or log in via the secure portal.
3. Role Selection & Approval:
    - Customers can easily check out products and start a savings objective right away.
    - The stores have to go to the "Apply as Seller" portal, provide business information and wait for Admin approval.
    - Riders are required to provide Admin with the details of their vehicle and license and will have to wait for Admin to clear their license.
4. Perform Key Operations:
    - Customers: Choose a product, lock the price, set up a layaway goal, and deposit a scheduled payment until you have paid 100%.
    - Stores: Process completed layaways, package and allocates available Riders through the dashboard.
    - Riders: Take on dispatch jobs, go to the store and fetch the package, use integrated map to navigate to the destination, take a picture as Proof of Delivery and complete the delivery.

6.2 Use Case Scenarios

Example 1: Starting a Savings Goal (Customer)

- Step 1: Customer shops the market and makes a choice on the desired product for him.
- Step 2: When the customer clicks "Set Goal", the current market price of the item is set as the goal.
- Step 3: Customer deposits initially through Stripe or Wallet balance.
- Step 4: Money is kept in the platform's Escrow system. The customer monitors progress in the "My Goals" dashboard until the goal is met.

Example 2: Order Fulfilment & Dispatch (Store)

- Step 1: When a customer has saved 100% of the amount needed to meet their savings target, the store is asked to deliver.
- Step 2: The store packages up the item and after that opens the "Order Management" map interface.
- Step 3: Store chooses an Approved Rider from the drop down menu and clicks on "Send Request.
- Step 4: The store gives the package to the Rider. The Rider will pick up the customer and complete the purchase by taking a photo of the customer's package and adding it to the system.

7. Project-Specific Section (Web Application)

7.1 User Roles

- Admin: Platform Analytics, User Bans, Manual Payments, and Approval Queues (Stores/Riders) - full access.
- Customer: Access to marketplace, digital wallet, layaway progress tracking and delivery tracking.
- Store: Product Inventory Management, Order Fulfilment, Product Pricing controls and Revenue Dashboard.
- Rider: Active delivery queues, live tracking delivery maps, interface to upload delivery photos and earnings wallet.

7.2 Core Integrations & APIs

- Clerk Authentication: Handles secure login, registration and session tokens for all four user roles.
- Stripe Escrow: Manages payment processing in the sandbox in a secure manner while funds are held until layaway is complete or until refunds are initiated.

7.3 Data Input / Output

- Input: Product metadata (images, MRP), CNIC/License upload for verification, Lat/Lng for address and Photo upload for Proof of Delivery.
- Output: Real-time progress bars for financial goals, interactive Leaflet delivery maps and PDF-style transaction receipts.

8. Troubleshooting

Issue	Possible Cause	Solution
"Access Denied" on Store/Rider Dashboard	The application you've submitted remains as "Pending" in the database.	Please wait for an Administrator to review and approve the profile, or contact support.
Map does not show routes.	The server or the network was dropped, or the rate limit of the routing server was exceeded.	Refresh the page to re-download the route or use the straight line visual indicator.
Cannot assign Rider	There are no "Available" or logged in Riders in your city at this time.	Wait later or try to make the order.
The payment method is invalid.	The amount of money stored in the wallet is not enough or the limits for sandbox tests have been reached.	Make sure you are using the user guide test card set of credentials.

Table 89: Troubleshooting

9. FAQs

Q1: What happens if a customer cancels a layaway goal halfway through?

The system will automatically work out any platform fees that may apply and start a refund procedure. The remaining balance will be refunded to the customer's digital wallet depending on the Refund Status policy of the platform.

Q2: How does the Price Lock feature work?

The current price of the product is recorded in the database when a goal is started. The customer will only pay the locked amount and even if the price of the product is increased in the public domain later, there will be no effect on the customer.

Q3: How do Stores and Riders get paid?

Funds are held in Escrow. Once the Rider successfully uploads the photographic Proof of Funds are held in Escrow. If the Rider successfully uploads the photographic Proof of Delivery the system changes the status of the Order to "Delivered" when the order is at the Customer's destination. The smart logic then notifies the admin and the admin will release the product cost to the Store's payout queue and the delivery fee will be released to the Rider's wallet.

10. Limitations

- Network & Hardware Dependency: As for Real-time delivery tracking, an Internet connection is required, and the rider should have a mobile camera that works for delivering.
- Payment Environment: This platform is simulated using the Stripe Sandbox environment at the present time.
- Geographic Service Boundaries: Optimized routing and delivery algorithms for well-mapped urban and suburban road network systems. In remote and unmapped rural areas, delivery times to such locations may be inaccurate or may default to straight-line visual routing.

11. Future Enhancements

- Dedicated Mobile App: Taking the Rider, Store and Customer dashboards into a React Native Mobile App to improve push notifications and access to native GPS hardware.
- Advanced Automation: Using Elixir/Erlang concurrent systems for processing huge amounts of transaction events during the holidays.
- Localized Payment Gateways: Going beyond just Stripe and adding regional mobile wallets and direct bank transfer APIs will make the layaway model more available to the unbanked or underbanked masses.

12. Contact / Support

To request technical support, to ask questions about the platform or to get help with the account, contact the development team:

- Naqash Sachwani (Developer)
- Naveed Asad Ali (Developer)
- Institution: SZABIST University, Karachi Campus
- Project Supervisor: Dr. Syed Samar Yazdani

Appendix A: Glossary

Term	Definition
Layaway	Finance, the provision of which enables customers to book a product and pay for it over a period of time.
Savings Goal	A customer's own strategy for saving money over a stated time horizon for a desired product.
Store	A registered seller who is selling a product by layaway on the DreamSaver platform.
Price Lock	A feature that sets the price for a customer to a certain price for a specific time.
Escrow	A safe holding account in which the money is deposited until the savings target is met.
Refund	The mechanism for performing the return of saved funds to the customer after cancellation, according to the platform policy.
Deposit	The contribution by the user toward his/her savings objective.
Dashboard	Interface to the user including status, goals and transactions.
Rider	Employees who are responsible for collecting and delivering fully laid away items from the warehouse.
Admin	A person who is responsible for the users, dispute resolution, account verification, and payouts of a platform.
Customer	Registered user who logs in to the marketplace and sets up a savings goal for a product.
Wallet	Refunds, promotional bonuses or delivery commission that has been credited into a digital balance within the platform.
Proof of Delivery	A photograph of the package which is taken by the Rider to confirm that a package has been delivered successfully to the Customer.
In Transit	A delivery status that means a package has been retrieved from the Store and is en route to the Customer.
Tracking Number	A tracking code that is given to all orders once they are fully paid that helps track the progress of the order.
Payout	The permission to release cleared money from the platform's escrow account to a Store's or Rider's bank account.
Stripe Gateway	A trustworthy third party credit and debit card payment processor.
Dispute / Ticket	A formal complaint or support request by a user due to an order, payment or platform issue.
Coupon	A discount code that consumers enter with their layaway plans to get a percentage or flat rate discount.

Table 89: Glossary

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	Higher Education Commission Pakistan on 2026-06-27 — 2%
2	Student papers	Shaheed Zulfikar Ali Bhutto Institute of Science & Technology on 2026-06-03 — <1%
3	Student papers	Higher Education Commission Pakistan on 2026-06-27 — <1%
4	Student papers	Sukkur Institute of Business Administration on 2026-06-28 — <1%
5	Student papers	Higher Education Commission Pakistan on 2026-06-05 — <1%
6	Internet	www.coursehero.com — <1%
7	Student papers	Sukkur Institute of Business Administration on 2026-06-27 — <1%
8	Student papers	Islington College,Nepal on 2026-04-22 — <1%
9	Student papers	University of Wales Institute, Cardiff on 2025-09-07 — <1%
10	Student papers	University of London External System on 2017-03-02 — <1%
11	Student papers	Universiti Tunku Abdul Rahman on 2022-04-20 — <1%
12	Student papers	Swinburne University of Technology on 2024-06-30 — <1%
13	Student papers	From HEC Pool on 2026-04-29 — <1%
14	Student papers	University of Hertfordshire on 2023-08-28 — <1%

15 Student papers

University of Wolverhampton on 2021-07-17 <1%

16 Student papers

University of Greenwich on 2022-11-29 <1%

17 Student papers

Higher Education Commission Pakistan on 2026-05-14 <1%

18 Student papers

Asia Pacific University College of Technology and Innovation (UCTI) on 2024-04-30 <1%

19 Student papers

Heriot-Watt University on 2025-11-21 <1%

20 Student papers

Islington College,Nepal on 2026-04-21 <1%

21 Student papers

Miami Dade College on 2023-07-19 <1%

22 Student papers

Swinburne University of Technology on 2023-06-11 <1%

23 Student papers

Islington College,Nepal on 2025-12-30 <1%

24 Student papers

Islington College,Nepal on 2026-04-22 <1%

25 Student papers

LKCFES on 2025-10-13 <1%

26 Student papers

Regis University on 2015-10-18 <1%

27 Internet

open.uct.ac.za <1%

28 Student papers

Higher Education Commission Pakistan on 2025-12-17 <1%

29 Student papers

Islington College,Nepal on 2026-04-22 <1%

30 Student papers

Nanyang Technological University, Singapore on 2015-10-14 <1%

31 Student papers

University of Birmingham on 2024-09-02 <1%

32 Student papers

Islington College,Nepal on 2025-12-29 <1%

33 Student papers

National University of Modern Languages on 2026-04-06 <1%

34 Internet

www.karimcitycollege.ac.in <1%

Abdul Hafeez <hafeez@szabist.edu.pk>
to me, Syed

Report is attached.

***% detected as AI.**

Regards,

Abdul Hafeez Abbasi

Senior Manager IT - SZABIST
Tel: (021) 35824461-3 (Ext. 203)
Campus 100 Clifton, Block 5, Karachi

...

From: reportcheck@szabist.edu.pk <reportcheck@szabist.edu.pk> **On Behalf Of** Naqash Sachwani
Sent: Wednesday, June 24, 2026 10:11 PM
To: reportcheck@szabist.edu.pk
Cc: Dr. Syed Samar Yazdani <dr.syedsamar@szabist.pk> , bcsbs2212232@szabist.pk
Subject: [reportcheck1] Check plagiarism

Naqash Sachwani
2212230
Naveed Asad Ali
2212232
Provide report and plagiarism free certificate.