The Islamic University of Gaza
Faculty of Information Technology
Department of Software development

الجامعـــــــة الإســـلاميـة بغـزة
كلية تكنولوجيا المعلومات
قسم تطوير البرمجيات

Nab: A Real-Time Price Tracking Platform for Essential Goods in Gaza Strip

نبض: منصة لتتبع الأسعار في الوقت الحقيقي للسلع الأساسية في قطاع غزة

By

Osama Hassan Saeed Al-Hendi	**120211752**
Ahmed Hussam Bashir Al-Barbari	**120201450**
Zakaria Raed Yassin Siam	**120210169**
Mohammed Talal Rizq Nasrallah	**120212446**
Mohammed Eyad Ahmad Al-Daly	**120201572**

Supervised by

Dr. Rebhi S. Baraka

A graduation project report submitted in partial fulfilment of the requirements for the degree of Bachelor of Information Technology

December/2025

Abstract

In this project, we develop an e-service platform, called Nabd, that tracks essential commodity prices in the Gaza Strip and offers other services about goods, their locations, and people who sell them. The platform satisfies the people's needs on essential commodity prices and where they can find them. Currently, these prices and commodity locations are not available due to the lack of entrance of goods and the closure of commercial crossings by the occupation.

The platform consists of four integrated modules that together form a unified system for real-time price tracking and community trading in Gaza. It includes an Admin Dashboard that manages system operations, verifies vendors, updates data, and monitors activity to ensure platform integrity; an Exchange Module that facilitates product trades and negotiations, including item-for-item exchanges to overcome liquidity and supply challenges; a Notifications Module that keeps users informed about price updates, product availability, and store verification through the platform, email, or SMS/WhatsApp; and a Home Page that serves as the main interface for browsing products, filtering by city, and viewing recent updates, optimized for smooth use under low-bandwidth conditions.

To develop the platform, we use iterative waterfall methodology because it has proven to be suitable for kind of software. It has a mechanism for error correction providing feedback paths from every phase to its preceding phases. The value of the platform is to meet people's demands by giving the greatest pricing and the best areas to sell, which saves their time and money, as well as saving their lives from danger while walking the great to reach a commodity they need. The platform is simple to use and supports the Arabic language, making it accessible to everyone looking for a specific commodity.

ملخص الدراسة

كجزء من هذا المشروع، أنشأنا منصة خدمات إلكترونية تتابع أسعار السلع في غزة، وتوفر خدمات أخرى تتعلق بالسلع، وأماكن تواجدها، والأشخاص الذين يبيعونها وهي تُلبي حاجة سكان قطاع غزة بمعرفة الأسعار وأماكن توفر السلع، نظرًا لعدم توفرها نتيجة منع دخول البضائع وإغلاق المعابر التجارية من قبل الاحتلال.

منصة نبض هي نظام متكامل لتتبع الأسعار والتداول المجتمعي في الوقت الفعلي بقطاع غزة، تضم أربع وحدات مترابطة. تشمل لوحة التحكم الإدارية لإدارة النظام وضمان موثوقيته، ووحدة التبادل لتسهيل التفاوض والتبادل العيني بين المستخدمين، ووحدة الإشعارات لإيصال تحديثات الأسعار وتوافر المنتجات عبر القنوات المختلفة، إضافةً إلى الصفحة الرئيسية التي تتيح تصفح المنتجات بسهولة حتى في ظروف الاتصال الضعيف، مما يجعل نبض منصة شفافة وفعّالة تعزز الوعي الاقتصادي في المجتمع.

استخدمنا منهجية الشلال التكراري (Iterative Waterfall) لأنها تتضمن آلية لتصحيح الأخطاء، من خلال توفير مسارات تغذية راجعة من كل مرحلة إلى المراحل السابقة. تتيح هذه المسارات إعادة العمل على المرحلة التي وُجدت فيها أخطاء، وتعكس هذه التعديلات على المراحل اللاحقة. تتمثل الفائدة المقترحة في تلبية احتياجات الناس من خلال تقديم أفضل الأسعار وأفضل الأماكن للبيع، مما يوفر وقتهم ومالهم، بل ويسهم في حمايتهم من الخطر أثناء التنقل لمسافات طويلة للوصول إلى السلعة التي يحتاجونها. تتميز المنصة بسهولة الاستخدام ودعمها للغة العربية، مما يجعلها متاحة للجميع ممن يبحثون عن سلعة معينة.

Dedication

As students of the Islamic University of Gaza (IUG), we dedicate this project and application to our families and friends, and everyone who lives in Gaza Strip and cares about it.

Acknowledgment

In the name of Allah, the Most Beneficent, the Most Merciful.

We would like to thank our families for their support, sacrifices, and prayers. We also would like to thank our supervisor Dr. Rebhi S. Baraka for his guiding efforts and support during the project.

Table of Contents

List of Table

List of Figures

List of Abbreviations

DB	Database
CSS	Cascading Style Sheets
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
JS	JavaScript
MVC	Model-View-Controller
PHP	Hypertext Preprocessor
RDBMS	Relational Database Management System
SMS	Short Message Service
UI	User Interface
UML	Unified Modelling Language
UX	User Experience

Chapter 1
Introduction

Chapter 1
Introduction

In light of the challenging conditions in the Gaza Strip, residents are seeking solutions that can save them time, effort, and money by providing real-time information about the availability of goods and trusted sellers. This is particularly important due to high costs, scarcity of goods, and occupation policies that exacerbate food insecurity.

Nabd is an e-service platform designed to meet these needs. It offers a variety of features and functions that allow users to easily access product prices, locations, and vendor information. The platform saves time, effort, and money by allowing users to view up-to-date pricing details. Users can add products to their list of essential goods and receive notifications via third-party services.

The platform collects data on all available products in the Gaza Strip, monitors their prices in real-time, and records details about their locations and trusted vendors. We also aim to expand the platform to include a wide range of commodities, beyond just food.

The remainder this chapter includes the statement of the problem, the project's objectives, the proposed solution, the methodology, the expected outcomes, and the timeline for project.

1.1 Problem Statement

People of the Gaza Strip face significant difficulties in accessing essential goods due to the imposed blockade, closure of commercial crossings, and constant price fluctuations. These factors have led to consumer exploitation and a lack of transparency in pricing and distribution. Moreover, the existing solutions—if available—are often outdated, manually operated, inefficient, and fail to account for the region's unique challenges, such as poor internet connectivity and limited access to digital services.

To address these issues, we need to develop an e-service platform that provides real-time tracking of essential commodity prices, identifies their availability and trusted

sellers. In order to overcome the poor internet connectivity, the platform needs to send notifications via SMS and WhatsApp. It further needs to empower citizens to make informed purchasing decisions, saves them time and money, and protects them from unsafe movement in a context of economic hardship and security risks.

1.2 Objectives

1.2.1 Main Objective

To develop a platform that gathers data on all available products in the Gaza Strip, monitors their prices in real time, and tracks their locations and sellers — with the ultimate goal of enhancing price transparency, reducing exploitation, and facilitating safe, informed access to essential goods.

1.2.2 Sub Objectives

To achieve the general objective and solve the problem, we choose to accomplish the following specific objectives:

1. To examine relevant literature and works, as well as similar applications, and analyse their benefits and drawbacks. Then we can describe the nature and features of our platform.
2. To assess the needs of the platform and identify its functional and non-functional features in order to satisfy user expectations.
3. To specify and create the platform's modules, architecture, components, interfaces, and data in accordance with the requirements.
4. To determine the right tools and technologies to create the platform based on the modules that have been designed.
5. To carry out the necessary platform testing and validation to guarantee that it is accurate and meets the specified requirements.

1.3 Scope and Limitations

Scope:

- Users in the Gaza Strip are the project's aim. For those who sell food commodities and those who are searching for them in Gaza, the platform concentrates on the following services:

- Real-Time Tracking: The platform will display up-to-date information on the prices of food commodities, including details about where these items are located and who the trusted sellers are.

- Location and Vendor Information: The platform will provide users with detailed information about product availability and trusted vendors, helping consumers avoid unreliable sellers.

- Notifications via SMS and WhatsApp: To overcome internet limitations. This empowers citizens to make informed purchasing decisions, saves them time and money.

- User Accounts: The platform will allow users (both vendors and consumers) to create personal accounts, where they can save their preferences, receive alerts, and track transactions.

- Search Functionality: Users will be able to search for specific products by name, price, location, or vendor. Advanced filters will enable them to find the best deals quickly.

Limitations:

- Geographic Restriction: The platform is specifically designed for residents of the Gaza Strip and therefore operates solely within the region. As a result, users outside Gaza will not have access to the services provided.

- Difficulty Identifying Commodities: Due to the ongoing closure of crossings, the platform may face challenges in identifying and tracking all available

commodities in Gaza. This limitation could prevent the platform from providing a comprehensive list of goods in real-time.

- Internet Connectivity Issues: Gaza's limited and unstable internet infrastructure may make it difficult for many residents to use the platform regularly. While the platform will send notifications via SMS and WhatsApp to address this issue, users with unreliable internet access may still face challenges in fully utilizing the platform's features.

- Offline/Online Functionality: While the platform will provide SMS and WhatsApp notifications to users with limited internet access, certain services and real-time updates, such as tracking prices and availability, will only be accessible online. Users without internet access will have difficulty taking full advantage of the platform's features.

- Speed and Responsiveness: The speed and responsiveness of the platform may be affected by the inconsistent internet speed in Gaza. Users might experience delays in receiving real-time updates on prices and product availability.

- Security Concerns: Although the platform will implement security measures to protect user data, the ongoing political instability in Gaza could present challenges in terms of data protection and privacy. Accessing sensitive information securely may be difficult, particularly for users relying on insecure internet connections.

1.4 Importance of the Project

The importance of this project stems from the problem it addresses and the objectives it seeks to achieve. It overcomes the limited access to goods caused by the ongoing blockade and the closure of commercial crossings, the residents of Gaza face severe difficulties in accessing essential goods.

It also overcomes price exploitation where untrusted traders take advantage of price fluctuations caused by shortages, leading to the exploitation of consumers.

The platform provides users with real-time, transparent information about the prices of essential goods, thereby reducing price manipulation and assisting consumers in

making informed choices. It also provides users with easy access to goods tracking the availability and locations of goods, thereby facilitating safe and efficient access to essential commodities.

Overall, the project is crucial in addressing the urgent need for price transparency and access to goods in Gaza Strip. By providing real-time updates on the prices and locations of essential goods, it reduces consumer exploitation, ensures fair pricing, and allows people to find what they need without risking dangerous trips. In a region where access to basic necessities is limited and often manipulated, this platform becomes a vital tool for residents, ensuring both economic fairness and safety.

1.5 Methodology

Our methodology is the Iterative Waterfall Methodology [1] The iterative waterfall methodology has a mechanism for error correction providing feedback paths from every phase to its preceding phases. The feedback paths allow the phase to be reworked in which errors are committed and these changes are reflected in the later phases. We gathered the information we needed for the project during our meeting with our supervisor, and we interviewed Customers and asked if they did deal with Buy Products before. We have followed the following phases which are depicted in Figure 1.1:

Phase 1: Requirement Gathering and Analysis

Besides collecting data, during this phase we will gather all the information and Requirements needed for the project.

Phase 2: Application Design

Prepare the system design and define the overall system architecture based on the Requirement specifications from the first phase.

Phase 3: Implementation

Start coding and developing the website.

Phase 4: Testing and Deployment

Testing every function in the website.

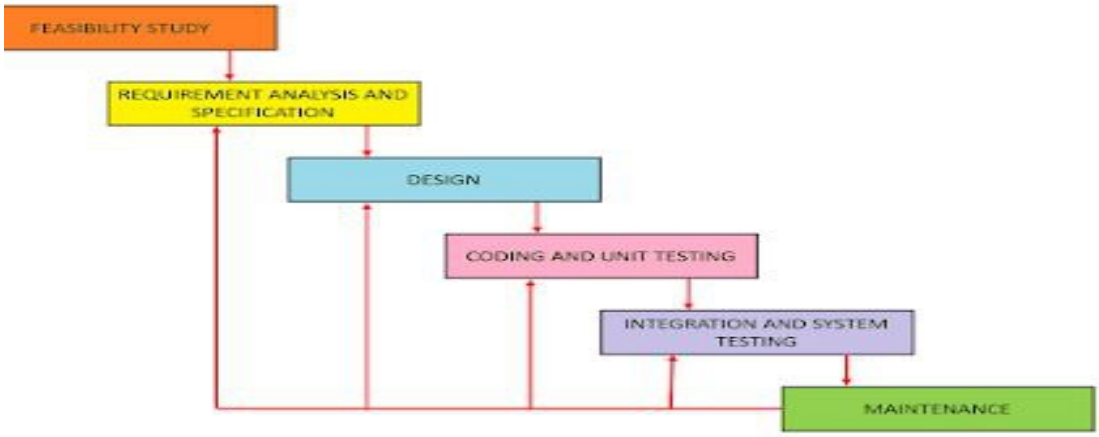


Figure 1.1: Iterative Waterfall Methodology Used in Developing Nabd Platform

1.6 Tools and Equipment

Laravel Framework:

Laravel [2] is an open-source PHP framework, which is robust and easy to understand. It follows a model-view-controller (MVC) design pattern. Laravel reuses the existing components of different frameworks which helps in creating a web application. The web application thus designed is more structured and pragmatic. We used Laravel to Build the website and start implementation.

Vue JS:

Vue JS [3] is an open-source, progressive JavaScript framework, for front-end, facilitates the UI (user interface) development of websites and single-page applications.

PhpStorm:

PhpStorm [4] is a PHP IDE. It provides on-the-fly error prevention, autocompleting and code refactoring, zero configuration debugging and an extended HTML, CSS, and JavaScript editor, it also provides powerful built-in tools for debugging, testing and profiling your applications. Made by Jet Brains.

MySQL:

MySQL [5] is the world's most popular Open-source relational DB management system (RDBMS). With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications.

Excalidraw:

Excalidraw [6] is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.

Twilio:

Twilio [7] gives you the tools to connect digital experiences on any channel, full access to your customer data, and AI to work efficiently.

1.7 Time Table

A time table [8] represents the schedule of activities for the project. It shows the main phases, the tasks under each phase, and the time period (start and end dates or weeks) allocated for each activity. Table 1.1 shows the project schedule and the time allocated for each phase of development. It outlines the main stages of the Nabd project, such as analysis, design, implementation, and testing, along with their corresponding durations. The purpose of this table is to organize the workflow, ensure proper time management, and track the progress of the project from start to completion.

Table 1.1: Time Table of The Project Phases

	Task Name	Task Description	Duration
1	**Requirement Gathering and analysis**	Collecting data	5 days
		Interviewing Customers	10 days
2	**System Design**	Prototyping	5 days
		Database design	7 days
		System user interfaces and ER diagram and use cases	10 days
		Design web pages	10 days
3	**Implementation**	Database creation	15 days
		Front-end development	20 days
		Back-end development	30 days
4	**Testing**	Test login/register unit	2 days
		Test Admin features	5 days
		Test Customer features	5 days
		Integrate units, final test.	2 days
		Total	126 days

Chapter 2
Literature Review

Chapter 2
Literature Review

This chapter provides a review of earlier related efforts and platforms that have parallels to Nabd. The primary goal is to present existing systems, their significant characteristics, and explain how Nabd differs by stressing real-time tracking of crucial products in crisis scenarios. Several e-service and data systems are analysed using academic sources and official websites to determine their strengths and limitations.

2.1 The Food Security Portal

It is a platform [9] that tracks food prices and assesses food security both globally and locally. It collects real-time data on the prices of essential goods, such as grains and oils, and presents this information through charts and indicators. The portal provides regular reports and analyses on how price fluctuations affect food security in various regions. It is used by governments and humanitarian organizations to track markets and guide relief efforts. The platform also offers interactive maps to visualize the impact of price changes in specific areas, helping to make informed decisions on food security strategies, Figure 2.1 shows the main page of The Food Security Portal.

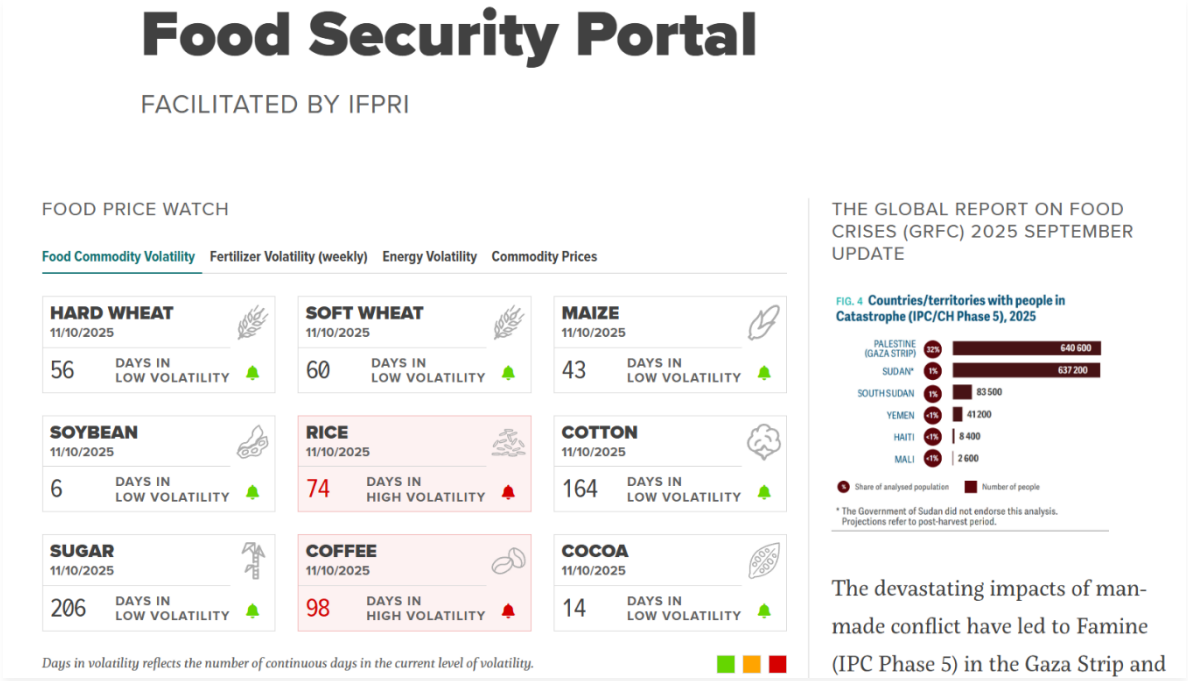


Figure 2.1: Food Security Portal Main Page

2.2 Datasembly

It is a web-based platform [10] that tracks grocery prices across thousands of stores in the U.S., updating data weekly using automated web scraping. It provides interactive price indexes and analytics but does not sell or display actual products. The platform targets retailers, government agencies, and market analysts to support informed decision-making. It is an effective tool for price transparency and monitoring market fluctuations; Figure 2.2 shows the main page of Datasembly.

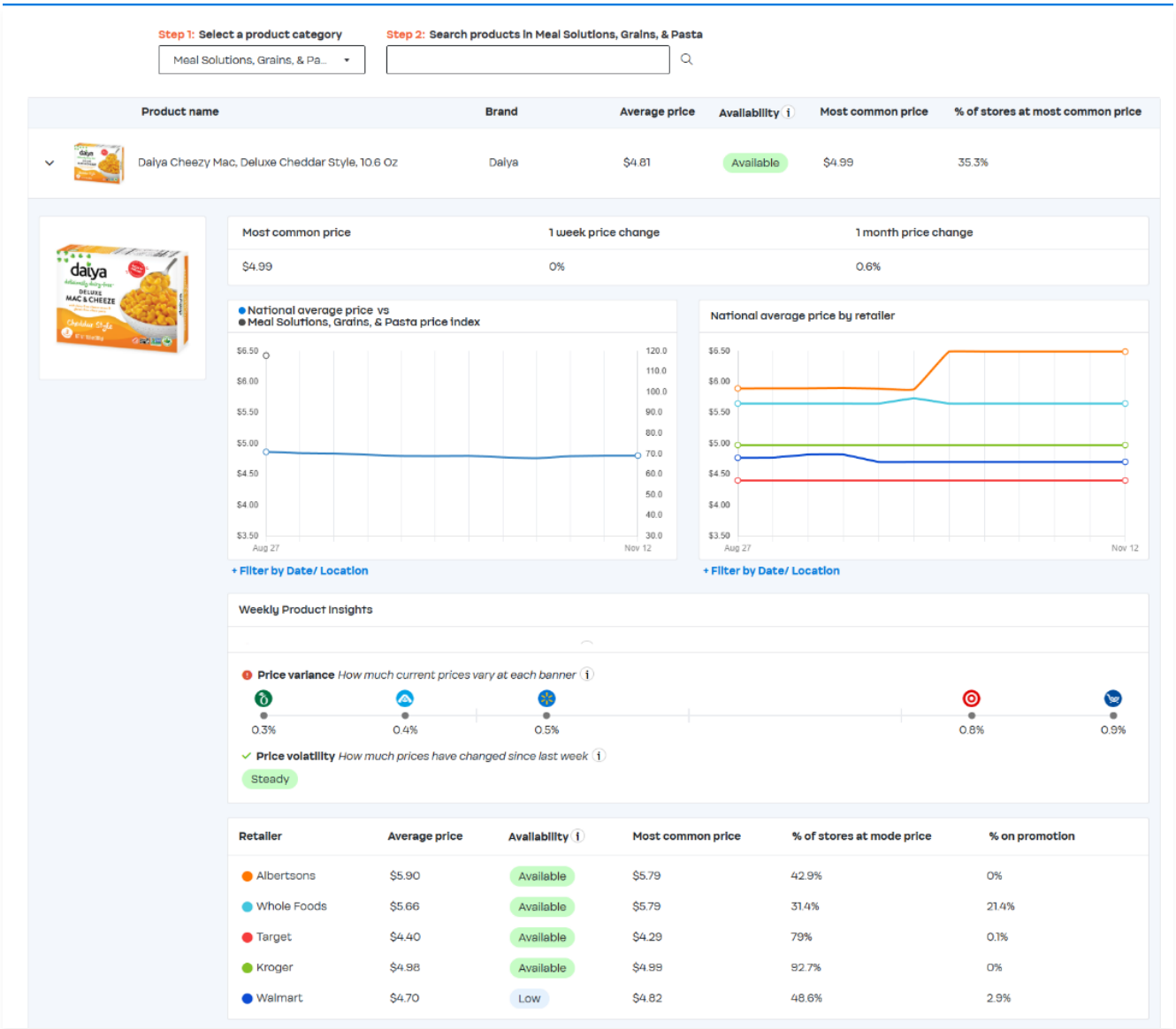


Figure 2.2: Datasembly Main Page

2.3 PriceSpy

It is an online price comparison platform [11] that helps users find the best deals on products across various e-service stores. The site displays and compares product prices from different retailers, while also offering reviews and ratings from other users. It also features price alerts, which notify users via email or app notifications when a product's price drops to a set threshold. The platform allows users to track product price history, providing insights into price trends over time. Overall, PriceSpy aims to provide a simple and effective tool for consumers to compare prices and make informed purchasing decisions, Figure 2.3 shows the main page of PriceSpy.

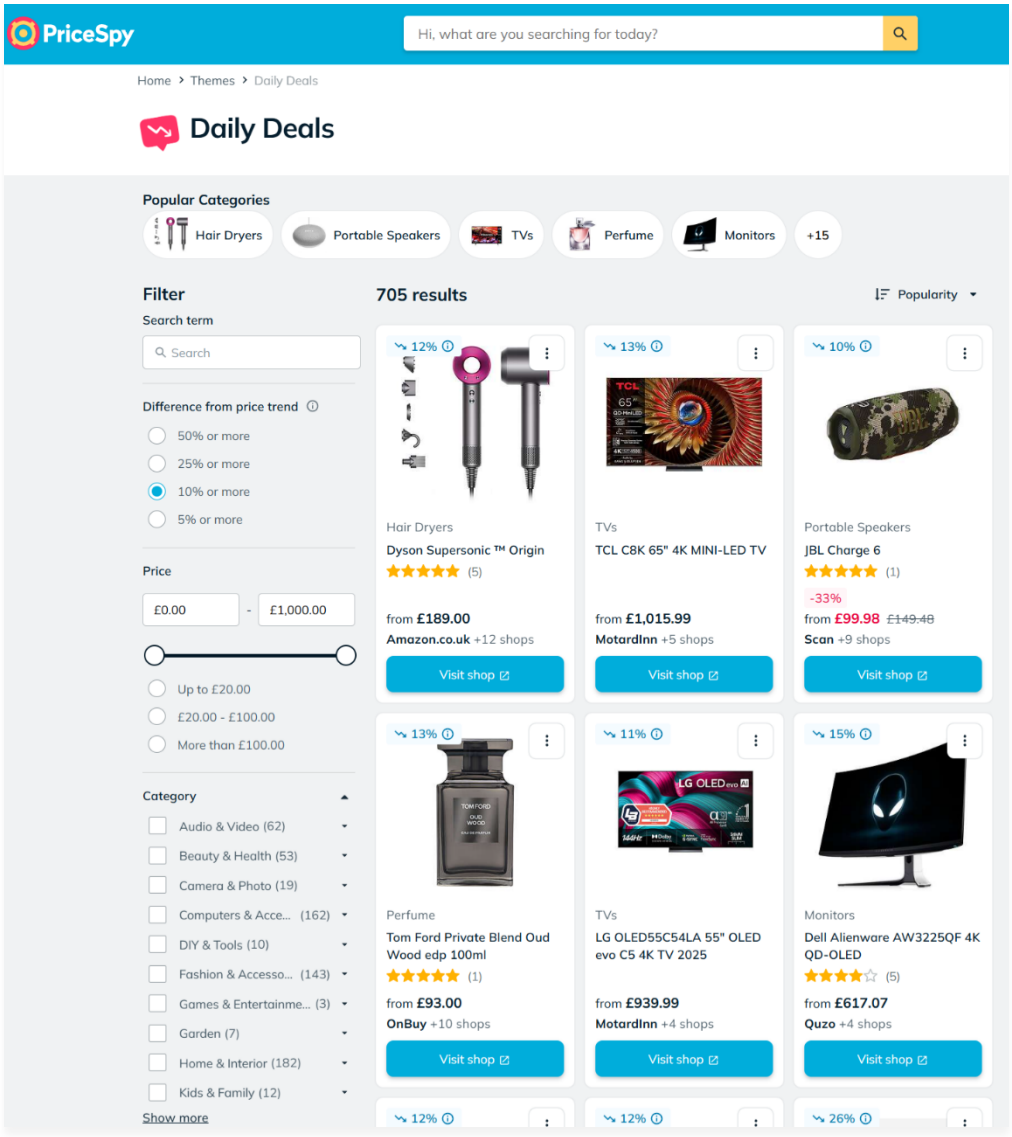


Figure 2.3: PriceSpy Main Page

2.4 Google Shopping

It is an online shopping platform [12] that allows users to search for products, compare prices, and find where they are sold, either online or in local stores. Users can filter product searches based on price, brand, and availability. The platform provides real-time price updates and enables users to set price alerts, notifying them when prices drop. Google Shopping also uses Google Lens, allowing users to scan products in stores to get more information and compare prices online. It helps users make informed purchasing decisions by offering detailed product listings and offering a variety of shopping options, Figure 2.4 shows the main page of Google Shopping.

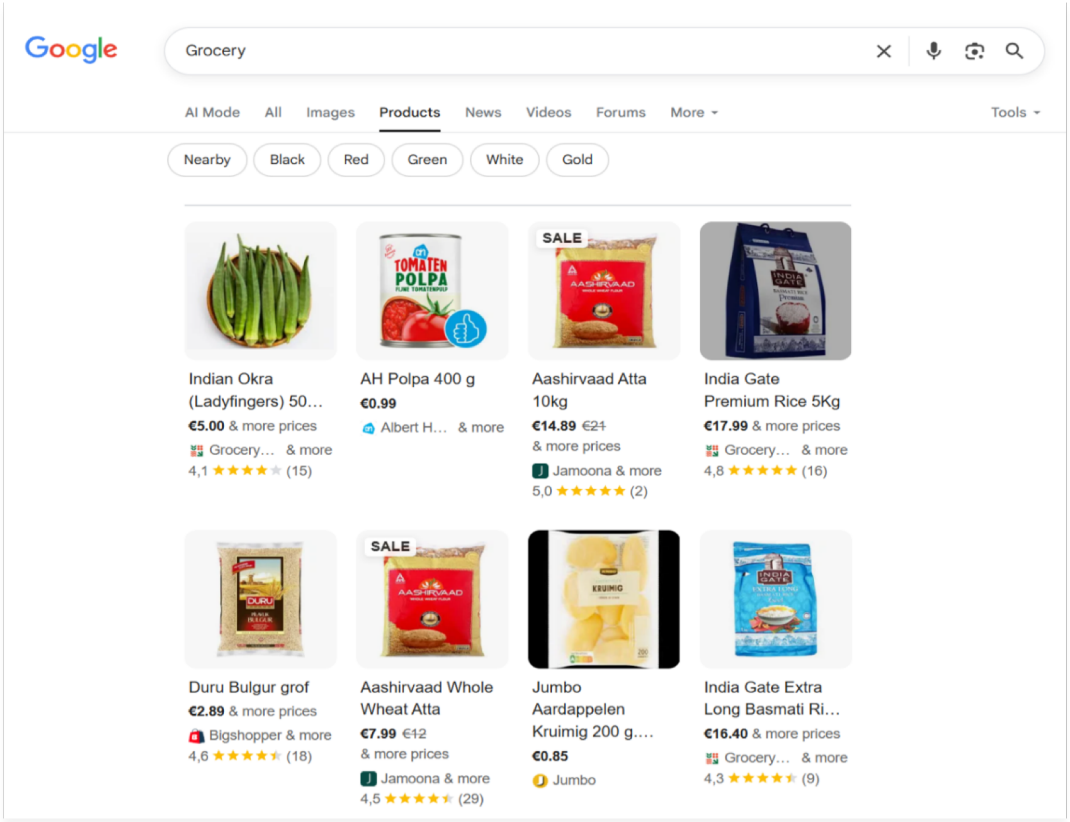


Figure 2.4: Google Shopping Main Page

2.5 Wadi

It is an online store [13] that provides a wide range of products, including electronics, clothing, and household items. It supports various payment options and offers return policies. While Wadi serves a broad consumer base, it lacks real-time updates and

focuses on general retail rather than addressing essential needs in emergency zones Figure 2.5 shows the main page of Wadi.

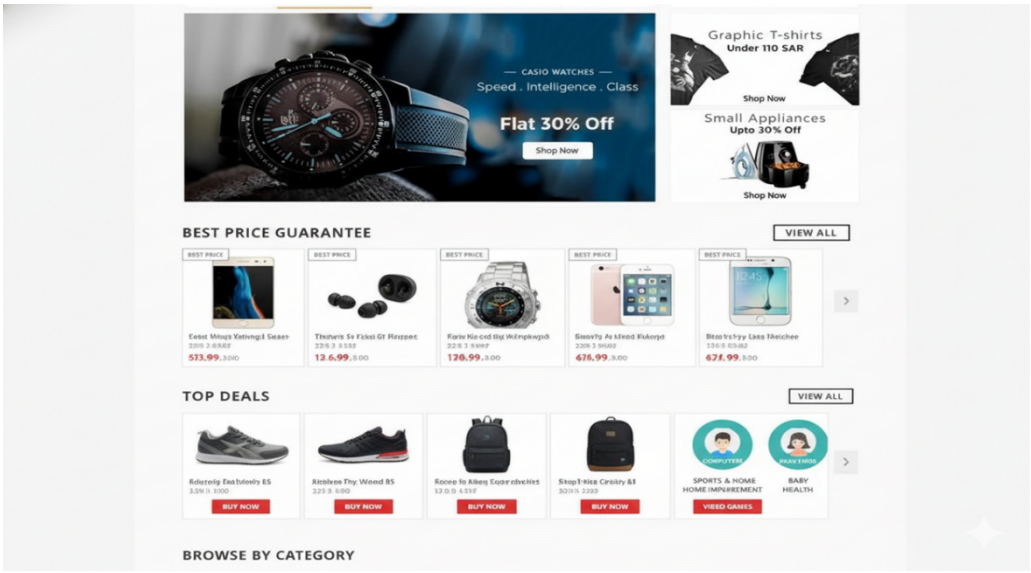


Figure 2.5: Wadi Main Page

2.6 Noon

It is a fast-growing platform in Saudi Arabia and the UAE [14] that features competitive pricing, frequent discounts, and a marketplace for third-party sellers. Its features are designed for convenience and product variety, but the platform does not address urgent supply challenges or cater to restricted regions Figure 2.6 shows the main page of Noon.

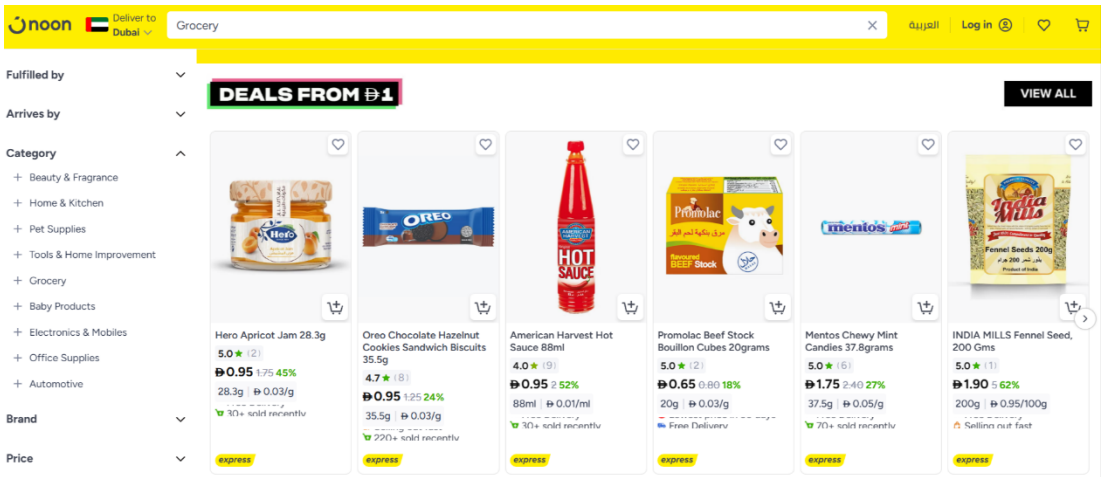


Figure 2.6: Noon Main Page

In contrast, the Nabd Platform is designed specifically for the Gaza Strip, where goods are scarce and prices are highly volatile. It enables users to access current prices, commodity locations, and seller contacts in real time. Unlike traditional e-service apps, Nabd uses SMS and WhatsApp notifications, making it more accessible in areas with weak or intermittent internet service.

2.7 A Comparison of the Most Important Platforms

The comparison highlights that Nabd was specifically developed for the Gaza context, unlike global or commercial platforms that focus on general markets and online shopping. It provides real-time local price tracking, verified stores, Arabic language support, and offline notifications via SMS and WhatsApp, making it more reliable and accessible in crisis environments with weak internet and limited mobility.

Table 2.1 compares the Nabd platform with other existing systems such as the Food Security Portal, Datasembly, PriceSpy, Google Shopping, Wadi, and Noon.
It highlights the main features of each system, including real-time price tracking, local commodity availability, offline access, emergency suitability, and product display.
The results show that Nabd offers a unique combination of features not found together in other platforms, such as real-time and local price tracking, offline access, and strong suitability for emergency contexts.

Table 2.1: A Comparison of Nab with Reviewed Platforms

Feature / Platform	Nab	Food Security	Datasembly	PriceSpy	Google Shopping	Wadi	Noon
Real-time Price Tracking	Yes	Yes	Yes (Weekly)	Yes	Yes	No	No
Local Commodity Availability	Yes	No	No	No	Yes (limited)	No	No
Offline Access	Yes	No	No	No	Yes (Partial)	No	No
Emergency Context Suitability	Yes (Very High)	Yes (High)	Yes (Medium)	Yes (Low)	Yes (Low)	Yes (Low)	Yes (Low)
Display of Products	Yes	No	No	Yes	Yes	Yes	Yes

Chapter 3
Requirements Analysis

Chapter 3

Requirements Analysis

This chapter presents the process of gathering and analysing the requirements for the Nabd platform. It explains how both functional and non-functional requirements were identified for consumers, merchants, and administrators. The aim is to ensure that the system reflects real community needs and operates efficiently under Gaza's Internet limitations. The chapter includes use cases, sequence diagrams, and system behaviour descriptions that form the foundation for the design phase in Chapter 4.

3.1 Functional Requirements:

Functional requirements [15] describe what the system should do and how users interact with it. They define the core operations that make Nabd useful and practical for its audience and are divided into User Requirements and Administrator Requirement, and Merchant Requirements.

3.1.1 User Requirements:

The following requirements outline the main functions that users and vendors can perform on the Nabd platform, such as registration, product search, price tracking, and receiving alerts, reflecting the needs and expectations of end users regarding what the system should do and how they wish to interact with it.

1. The user must be able to register and create a new account.
2. The user is able to enter the system.
3. The user must be able to edit their profile information.
4. The user must be able to change their password.
5. The user should be able to search by product name or category.
6. The user must be able to review the list of available products.
7. The user must be able to see the details of a particular product.
8. The user should be able to search about shops/stores.
9. The user must be able to see the notifications.
10. The user shall be able to see price changes (increase or decrease).
11. The user should be able to show the shop/stores.

12. The user must be able to log out of the system.

3.1.2 Administrator Requirements:

Administrator requirements define how system administrators manage and control the Nabd platform, including managing user accounts, controlling permissions, verifying stores and products, monitoring activities, and ensuring secure access—thereby maintaining data accuracy, reliability, and continuous updates.

1. The system administrator must be able to log on to the system.
2. The system administrator should be able to see the categories and be able to add, delete, or search for these categories.
3. The system administrator should be able to see the products and be able to add, delete, or search for these products.
4. The system administrator should be able to see the shops/stores and be able to add, delete, or search for these shops.
5. The system administrator should be able to see the product price.
6. The system administrator should be able to see the change of price and be able to update this price.
7. The system administrator should be able to see the users and be able to add or delete for these users.
8. The system administrator should be able to see the notifications and be able to add, delete, or update these notifications.
9. The system administrator should be able to see the user's feedback.

3.1.3 Merchant Requirements

The merchant in the Nabd platform is responsible for managing commercial content by listing products, posting promotional offers, and analysing market trends. These requirements enable merchants to add and update product information, create discounts to attract consumers, and access analytical insights that help them understand pricing patterns and demand. Through these functions, merchants can effectively manage their store and contribute to a dynamic and competitive marketplace.

1. The merchant must be able to log on to the system.
2. The merchant must be able to register and create a new store.

3. The merchant must be able to view and update their store information (name, address, location, image).
4. The merchant must be able to add products to their store.
5. The merchant must be able to view the list of their products and see the details of a specific product.
6. The merchant must be able to update product information (price, description, image).
7. The merchant must be able to delete products from their store.
8. The merchant must be able to create discount offers for their products.
9. The merchant must be able to view, update, and delete their discount offers.
10. The merchant must be able to edit their profile information and change their password.
11. The merchant must be able to log out of the system.

3.2 Non-Functional Requirements:

Non-functional requirements [16] describe how the system should operate rather than what it should do. They define the quality attributes that make Nabd efficient, secure, and reliable, even under weak internet connectivity and limited technical resources.

Performance (Real-Time Updates):

- The system must reflect any price change within 5 seconds of update in the database.
- The platform uses optimized API endpoints and caching to ensure near-real-time data delivery despite limited bandwidth in Gaza.

Availability (Offline Support):

- The system must remain operational 24/7.
- When internet connection is lost, the system must display cached data from the most recent session and automatically synchronize once connection is restored.

Reliability (Email/SMS/WhatsApp Notifications):

- Notifications must be sent within 30 seconds after a price alert or system event is triggered.
- The delivery success rate should reach at least 95%, and failed messages must be queued and retried automatically within 5 minutes.
- The notification system must ensure that no duplicate or missing messages occur during transmission.

Usability:

Interface is simple, Arabic-first, and suitable for users with limited technical knowledge.

Maintainability and Security:

The platform follows MVC structure, enabling modular updates and secure role-based access control.

3.3. Use Cases

UML Use Case Diagram [17] illustrates the interaction between consumers, merchants, and administrators within the Nabd platform, highlighting their respective functions such as product management, trading, reporting, and system monitoring. It demonstrates how these roles integrate to create a cohesive and transparent real-time market environment.

Figure 3.1 shows the main use case diagram of the platform and how users interact with it. It outlines the primary activities associated with each role and the relationships between them, providing a clear depiction of the platform's functional structure and user interactions.

A diagram emphasizes the platform's role-based design, where users such as consumers, merchants, and administrators engage with distinct yet interconnected features. It serves as a high-level representation of the system's operational framework,

demonstrating how each user type contributes to and benefits from the platform's integrated services.

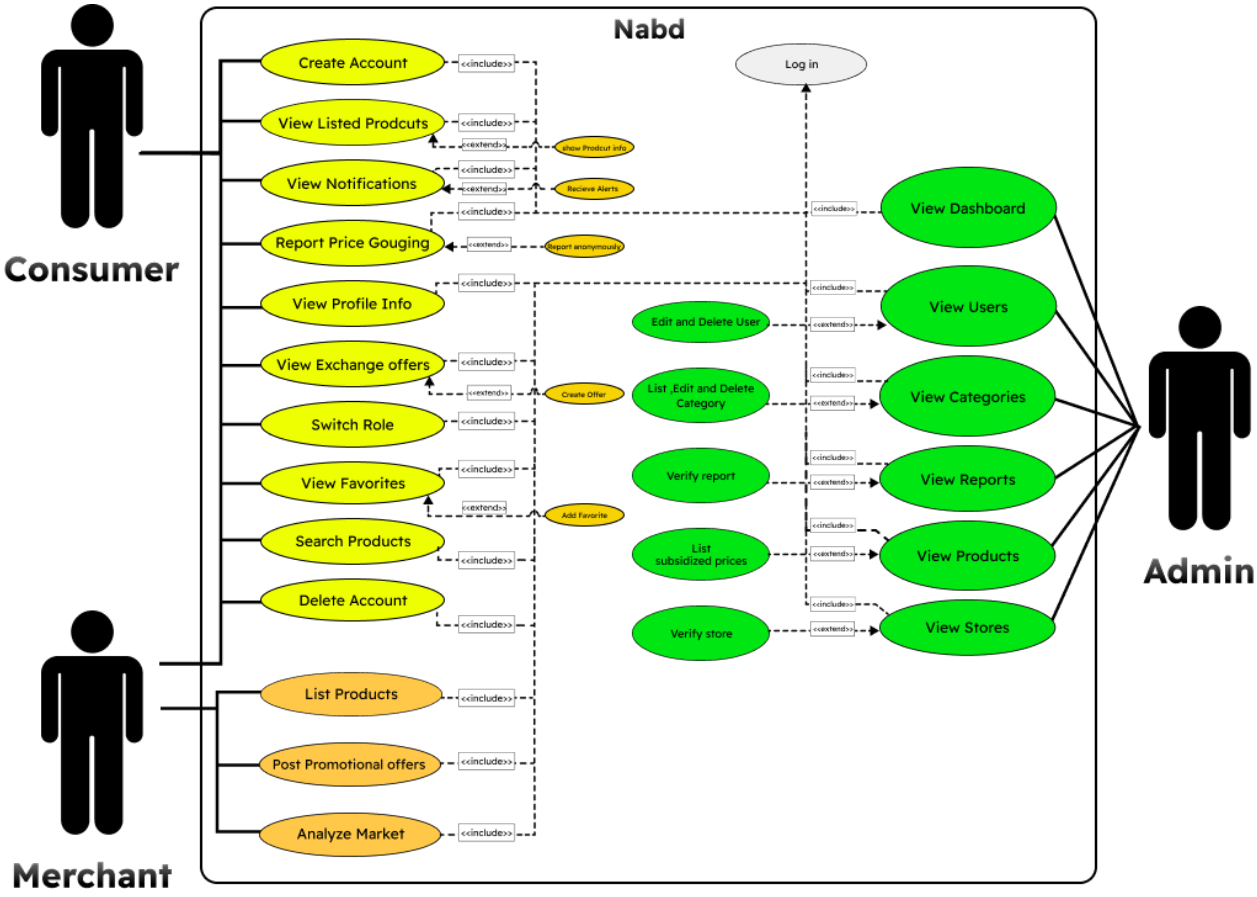


Figure 3.1: Nab Platform Use Cases

3.3.1. Use Case Tables

Use Case Tables [18] are structured tables that describe each use case in detail.
They show what happens before, during, and after each operation, including all conditions and exceptions.

3.3.2 User Use Case Tables:

The following table represents a summary of all use cases in the system for all types of users.
It shows the main functions and interactions that each user can perform within the system.

Table 3.1 presents all the main functions and actions that a normal user can perform within the Nabd Platform, whether the user is a consumer searching for products or a vendor offering them.

The purpose of this table is to describe how users interact with the system and what services are available to them, starting from registration and ending with logging out.

Table 3.1: All User Use Case

ID	**Name**
U01	Registration
U02	Login
U03	Forgot password
U04	Reset password
U05	View Home (public)
U06	Search products
U07	Compare store prices
U08	Create price alert
U09	View active alert
U10	Enable/Disable alert
U11	Configure notification methods
U12	Browse exchange offers
U13	Open offer and chat
U14	Create exchange offer
U15	View profile info
U16	View favourites
U17	Delete Account
U18	Logout

Table 3.2 describes how a new user registers in the system. The user opens the registration page, enters their name, email or phone number, and password, then submits the form. If all data are valid, the system creates the account and redirects the user to the home page.

Table 3.2: Registration Use Case

ID	U01
Name	Registration
Description	Register with name + email or phone + password
Pre-Condition	CSRF cookie obtained and fields valid
Action	- Open Create Account page - Fill name, email/phone, password - Submit
Post-Condition	User signed in and redirected to Home
Exceptions	Validation errors from backend

Table 3.3 explains how users log into the platform using their email or phone number and password. If the data are correct, the user is redirected to the homepage or admin dashboard; otherwise, an error message appears.

Table 3.3: Login Use Case

ID	U02
Name	Login
Description	Login with email or phone + password
Pre-Condition	CSRF cookie obtained and fields valid
Action	- Open Login page - Enter email/phone + password - Submit
Post-Condition	If role=admin > redirect to Admin Dashboard
Exceptions	Validation errors from backend

Table 3.4 shows how users can recover their accounts when they forget their passwords. The system verifies the email or phone number and sends a password reset link or code to the user.

Table 3.4: Forgot Password

ID	U03
Name	Forgot password
Description	User reset their password by verifying email/phone number
Pre-Condition	User registered account.
Action	- Open Reset Password - Submit required fields
Post-Condition	User can set a new password successfully
Exceptions	Validation errors from backend

Table 3.5 describes how the user resets their password after receiving a verification link or code. The user opens the reset page, enters a new password, and submits it. The system updates the password and redirects the user to the login page with a success message.

Table 3.5: Reset Password Use Case

ID	U04
Name	Reset password
Description	Reset password via reset form
Pre-Condition	Valid reset flow and CSRF cookie obtained
Action	- Open forgot password - Submit required fields
Post-Condition	Redirected to Login; Success message
Exceptions	Validation errors from backend

Table 3.6 describes how users or visitors can view the home page, which displays general information and product updates, even without logging in.

Table 3.6: View Home Use Case

ID	U05
Name	View Home (public)
Description	View public landing page content
Pre-Condition	-
Action	Visit Website
Post-Condition	Home display
Exceptions	-

Table 3.7 explains how users search for products by typing the product name or selecting a category. The system shows all related results based on the search keyword.

Table 3.7: Search Product Use Case

ID	U06
Name	Search products
Description	Search for products by name/ category
Pre-Condition	Auth required for downstream views
Action	Use Search bar input
Post-Condition	Suggestion shown; can open results
Exceptions	-

Table 3.8 explains how users compare the prices of the same product across different stores to choose the best deal.

Table 3.8: Compare Store Price Use Case

ID	U07
Name	Compare store prices
Description	Compare product prices across stores
Pre-Condition	Result loaded
Action	Inspect items in list/map
Post-Condition	User understands pricing differences
Exceptions	-

Table 3.9 describes how a logged-in user can create a new exchange offer by entering product details and submitting the offer to appear on the platform.

Table 3.9: Create Exchange Offer Use Case

ID	U14
Name	Create exchange offer
Description	Create a new exchange offer
Pre-Condition	Authenticated
Action	- Press create new offer - Enter details - Submit
Post-Condition	Offer created
Exceptions	Backend validation error

3.3.3 Merchant Use Case Tables:

This section describes all the main interactions and actions that a merchant can perform in the Nab platform. Merchants are responsible for managing their stores, products,

and promotional offers. The tables in this section list each merchant operation, along with its steps, conditions, and possible exceptions. These use cases ensure that merchants can fully manage their commercial activities, such as adding products, updating their store, and creating discounts.

Table 3.10 lists the complete set of use cases available for merchants. It provides an overview of all operations they can perform, including store creation, product management, and offer/discount handling.
It acts as a summary reference showing the capabilities provided to merchants within the platform.

Table 3.10 All Merchant Use Case

ID	Name
M01	Create/Register Store
M02	Update Store Information
M03	View Own Products
M04	Add Product to Store
M05	Update Product
M06	Delete Product from Store
M07	Create Offer/Discount for Product
M08	Update Offer/Discount
M09	Delete Offer/Discount

Table 3.11 explains how a merchant can register and create a new store. It includes the required steps such as entering store details (name, address, location, image) and submitting the form.
The store enters a pending approval state until reviewed.
Possible exceptions include duplicate stores or validation errors.

Table 3.11 Create/Register Store Use Case

ID	M01
Name	Create/Register Store
Description	Create and register a new store for the merchant
Pre-Condition	Authenticated as User
Action	- Navigate to store registration - Enter store details (name, address, location, image)

Post-Condition	Store created and pending approval
Exceptions	Store already exists, validation error, unauthorized access

Table 3.12 describes how merchants can update their store information, including changing the name, address, location, or store image.
The process requires the merchant to navigate to the store settings and submit updated details.
Exceptions include unauthorized access or missing store data.

Table 3.12 Update Store Information

ID	M02
Name	Update Store Information
Description	Update existing store information and settings
Pre-Condition	Authenticated as Merchant, Store exists
Action	- Navigate to store settings - Modify store details (name, address, location, image) - Submit changes
Post-Condition	Store information updated
Exceptions	Store not found, validation error, unauthorized access

Table 3.13 explains the process of creating a new promotional offer for one of the merchant's products.
It involves selecting a product, entering discount details (percentage/price), and setting a start and end date.
After submission, the offer becomes active.
Exceptions include invalid discount dates or missing product information.

Table 3.13 Create Offer for Product

ID	M07
Name	Create Offer for Product
Description	Create a new discount offer for a product in the store
Pre-Condition	Authenticated as Merchant, Product exists in store

Action	- Navigate to my products section - Select product to add offer - Press create offer - Enter offer details (discount price/percentage, start date, end date) - Submit
Post-Condition	Offer created and active
Exceptions	Product not found, invalid dates, validation error, unauthorized access

Table 3.14 describes how merchants can update an existing discount or offer.
They can modify discount values, change dates, or update the offer status.
Typical exceptions include invalid inputs or missing offer records.

Table 3.14 Update Offer for Product

ID	M08
Name	Update Offer
Description	Update existing offer or discount details
Pre-Condition	Authenticated as Merchant, Offer exists
Action	- Navigate to my products section - Select product to update offer - Modify offer details (discount, dates, status) - Submit changes
Post-Condition	Offer updated
Exceptions	Offer not found, invalid dates, validation error, unauthorized access

Table 3.15 explains how to delete an offer from a product.
The merchant selects the product, chooses the offer to remove, and confirms deletion.
Errors may occur if the offer doesn't exist or if the merchant lacks permission.

Table 3.15 Delete Offer for Product

ID	M09
Name	Delete Offer
Description	Remove an offer or discount from a product

Pre-Condition	Authenticated as Merchant, Offer exists
Action	- Navigate to my products section - Select product to delete offer - Confirm deletion
Post-Condition	Offer deleted
Exceptions	Offer not found, unauthorized access, deletion error

3.3.4. Admin Case Tables

The following table represents a summary of all use cases related to the Admin in the Nabd system.

It shows the main functions and interactions that the administrator can perform within the platform, such as managing users, products, categories, and monitoring system activities.

These functions allow the admin to control the overall system operation, maintain data accuracy, and ensure smooth performance across all modules.

Admin Use Case Tables:

All Admin Use Case Tables describe all the functions and activities that the Administrator (Admin) can perform in the system.

It focuses on the admin role only, showing how the admin interacts with the system and what operations are allowed, Table 3.10 shows All Admin use case.

Table 3.16 lists all the main actions that the administrator can perform in the Nabd system. These include logging in, managing products, categories, users, stores, and viewing reports or notifications. It defines the admin's role in controlling and maintaining the platform's data and operations.

Table 3.16: All Admin Use Case

ID	Name
A01	Admin login
A02	View dashboard overview
A03	View products
A04	Add product
A05	Edit product
A06	Delete product
A07	View categories

A08	Add category
A09	Edit category
A10	Delete category
A11	View users
A12	Edit user details
A13	Change user role and status
A14	Verify Stores
A15	Verify Reports
A16	Logout

Table 3.17 explains how the administrator logs into the dashboard using valid credentials. After successful login, the system redirects to the admin control panel. If the entered data are invalid, an error message appears.

Table 3.17: Admin Login Use Case

ID	A01
Name	Login
Description	Admin logs into dashboard
Pre-Condition	CSRF cookie obtained and fields valid
Action	- Enter credentials - Submit
Post-Condition	Redirect to Admin Dashboard
Exceptions	Validation error

Table 3.18 describes how the administrator views the main dashboard. The dashboard displays system statistics such as the number of products, categories, users, and recent activities or notifications.

Table 3.18: View Dashboard Overview Use Case

ID	A02
Name	View dashboard overview
Description	- View counts of products, categories and users - View recent users - View Latest notification
Pre-Condition	Authenticated as admin

Action	Open Dashboard view
Post-Condition	Overview data display
Exceptions	Backend error

Table 3.19 shows how the admin can view all existing products in the system. The admin can also search, edit, or delete products from this section to keep product data up to date.

Table: 3.19: View Products Use Case

ID	A03
Name	View product
Description	List existing product
Pre-Condition	Admin authentication
Action	Products view
Post-Condition	Products display
Exceptions	Backend error

Table 3.20 explains how the administrator can view, add, or manage product categories. This function helps organize the products into logical groups for easier browsing and management.

Table 3.20: View Categories Use Case

ID	A07
Name	View categories
Description	List existing categories
Pre-Condition	Admin authenticated
Action	Open categories view
Post-Condition	Categories display
Exceptions	Backend error

3.4. Sequence Diagrams

Sequence Diagram [19] is a type of UML behavioural diagram that illustrates how objects in a system interact with each other over time to complete a specific process or use case.

3.4.1. User Customer Sequence Diagram

This section shows how the system interacts with the user during key operations such as login and registration. It illustrates the sequence of actions between the user interface, the backend controllers, and the database to complete each process successfully.

Figure 3.2 illustrates the sequence of actions that occur during the login process. It shows how the user enters their email or phone number and password in the interface, then the system validates the information through the backend controller. If the credentials are correct, access is granted, and the user is redirected to their homepage or dashboard; otherwise, an error message is displayed.

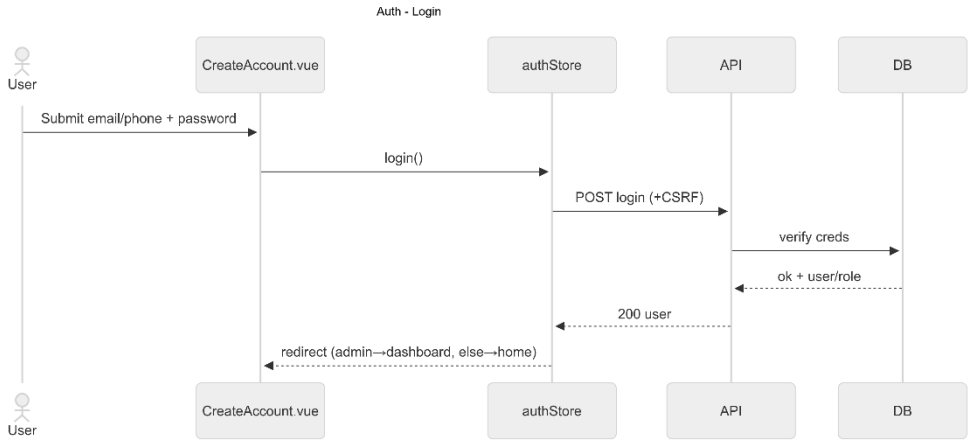


Figure (3.2): Login Sequence Diagram.

Figure 3.3 illustrates the sequence of steps that occur during the user registration process. It shows how the user fills in their personal details such as name, email or phone number, and password in the interface. The data are then sent to the backend

controller for validation and saved in the database. If the registration is successful, the system confirms the process and redirects the user to the home page**.**

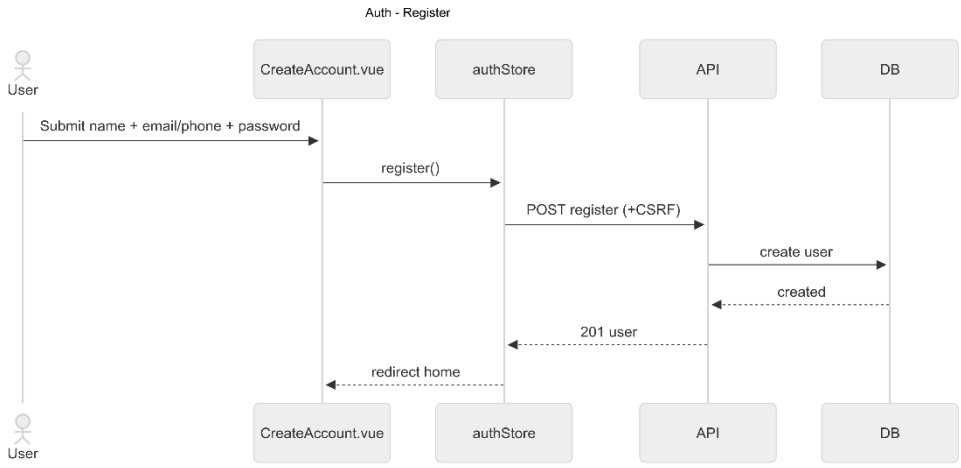


Figure (3.3): Register Sequence Diagram.

Figure 3.4 illustrates the sequence of interactions that occur when the administrator manages product categories. It shows how the admin sends a request through the dashboard to view, add, edit, or delete a category. The system processes the request through the backend controller, updates the database accordingly, and then displays the result or confirmation on the admin interface.

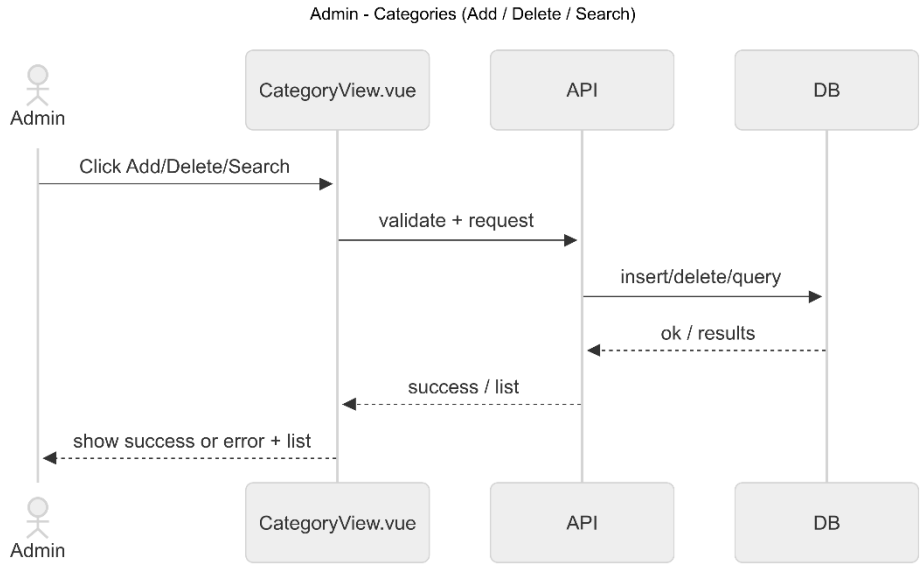


Figure (3.4): Admin Categories Sequence Diagram.

Figure 3.5 illustrates the sequence of actions that occur when the administrator views the main dashboard. It shows how the admin sends a request to load system data such

as the number of users, products, categories, and notifications. The backend controller retrieves this information from the database and returns it to the interface, where it is displayed as statistics and summaries for easy monitoring.

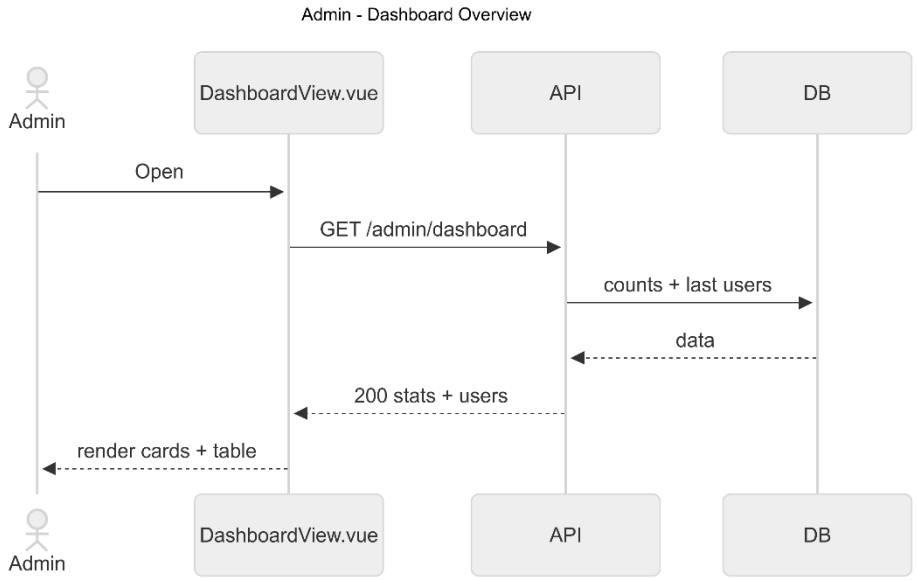


Figure (3.5): Admin Dashboard Overview Sequence Diagram

Figure 3.6 shows how a merchant registers a new store. The merchant enters the store details in the interface, then the backend controller validates the information and saves it in the database. After successful registration, the system confirms the creation of the store and returns a success message to the merchant.

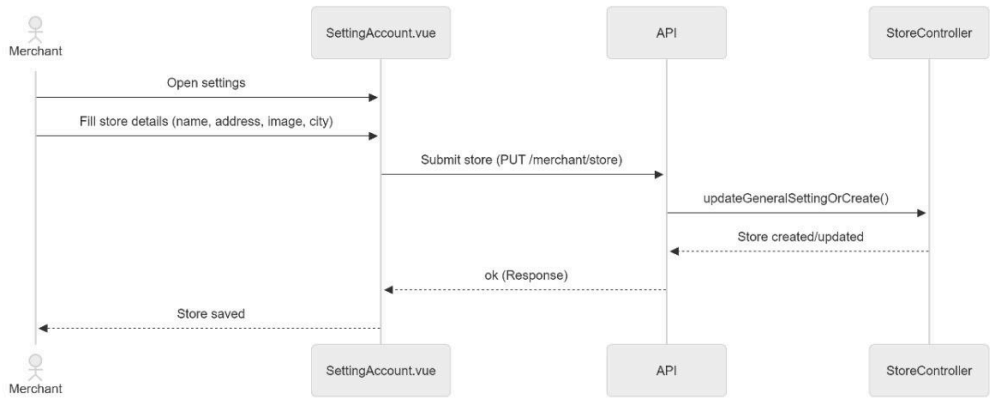


Figure (3.6): Merchant Register Store Sequence Diagram

Figure 3.7 explains the steps a merchant follows to update a product. The merchant selects a product and sends the new data (such as price or description) to the system. The backend verifies the input, updates the product record in the database, and then sends a confirmation back to the merchant interface.

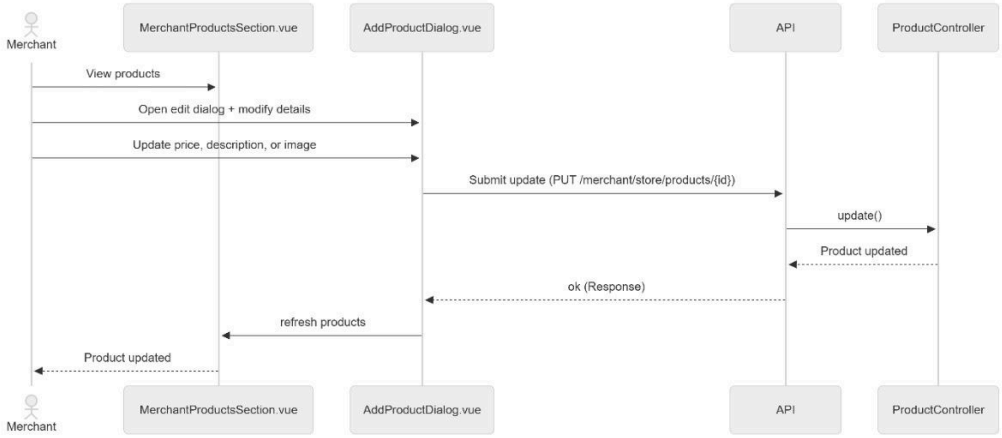


Figure (3.7): Merchant Update Product Sequence Diagram

Figure 3.8 illustrates how a merchant creates a discount offer for a product. The merchant enters discount details, the controller checks and stores the offer in the database, and the system updates the product's discount status. Finally, a success response is returned to the merchant indicating the offer has been created.

Figure (3.8): Merchant Create Discount Offer for Product Sequence Diagram

Chapter 4
Platform Design

Chapter 4
Platform Design

This chapter describes the design of the Nabd platform, explaining how its architecture and components were structured to meet the system requirements.
It outlines the use of the Model-View-Controller (MVC) pattern which ensures scalability, maintainability, and a clear separation between the backend and frontend.
The chapter divides into four sections. Section 4.1 outlines the overall architecture of the Nabd platform, section 4.2 presents the class and module design, section 4.3 presents the design of the data model and the database schema, section 4.4 presents the user interface design.

4.1 Nabd Platform Architecture

The Nabd platform follows the Model–View–Controller (MVC) architecture as shown in Figure 4.1. This separation of concerns ensures scalability, modularity, and maintainability of the system.

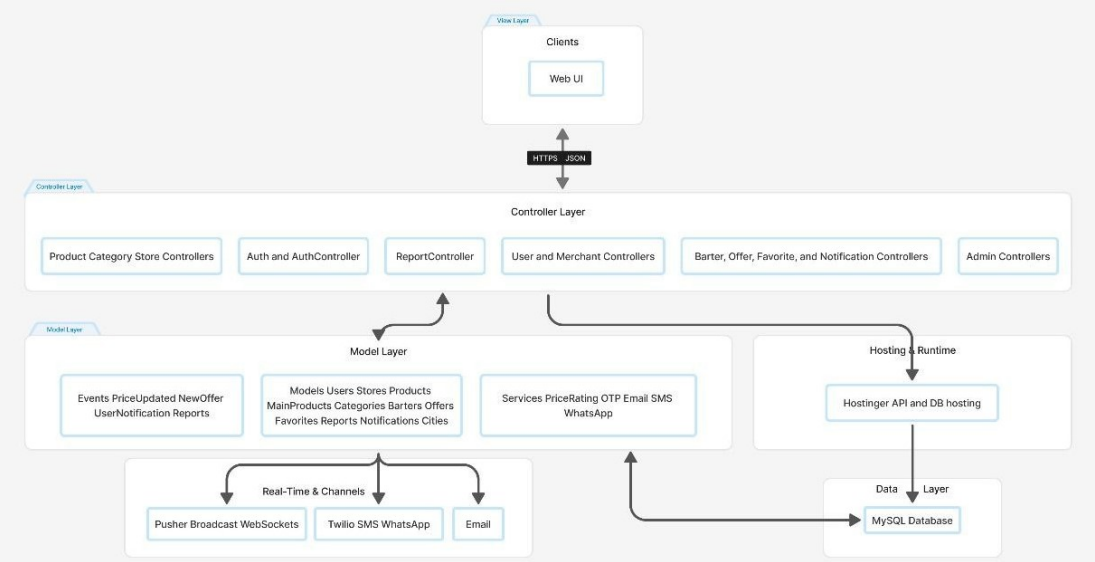


Figure 4.1 Overall Architecture of Nabd Platform

4.1.1 Model Layer

It is responsible for managing key entities such as:
Users: store personal data, preferences, and roles (user, merchant, admin).
Stores: include details about shop name, location, owner, and verification status.

Products: represent commodities, including price, category, and store associations.
Categories: group related products to improve navigation.
Notifications: store alerts related to price changes or user-specific updates.
Orders and Order Details: track purchase and transaction data for analysis.

4.1.2 Controller:

UserController: Handles registration, authentication (JWT tokens), profile updates, and alert preferences.
AdminController: Manages categories, stores, users, and monitors reports or system statistics.
ProductController: Processes CRUD operations for products and updates their prices.
NotificationController: Controls how and when notifications are sent (email, SMS, WhatsApp).
AuthController: Manages login, logout, and password reset functionalities.

4.1.3 View:

Main interfaces include:

- Login and Registration Pages: Allow users to authenticate and create accounts.
- Product Search and Map Pages: Display commodities, prices, and seller locations dynamically.
- Admin Dashboard: Displays analytics, reports, and management tools.
- Notifications Page: Shows alerts for price changes, active offers, and exchange requests.

4.2 Functions and Class Design

This section presents the controller classes used in the Nabd platform, including their responsibilities, main functions, parameters, return types, and key attributes.

4.2.1 AuthController

The AuthController manages all user authentication operations, ensuring secure access to the platform.

register(Request $request)

Creates a new user account after validating required fields such as name, email or phone, and password. Supports registration with either email or phone number. Generates a session and triggers a JoinNewUserEvent upon successful registration.

login(Request $request)

Authenticates the user using email or phone number and password. If the credentials are valid, the function creates a session and returns the authenticated user with their associated store and city information.

logout(Request $request)

Invalidates the currently active authentication session, effectively logging the user out of the system.

deleteAccount(Request $request)

Permanently deletes the authenticated user's account after verifying the current password. Invalidates the session and removes all user data from the system.

4.2.2 UserController

The UserController manages user profile data and personal settings.

update(Request $request)

Updates the authenticated user's personal information including name, email, phone, password, role, theme, currency, notification preferences, city, and location sharing settings. Validates current password when changing password and triggers ChangeUserRoleEvent if the user's role is modified.

4.2.3 CustomerController

The CustomerController handles customer-specific operations and profile management.

updateProfile(Request $request)

Updates the customer's personal information (name, email, phone, role) and optionally creates a store if the user changes their role to merchant. Validates input with regex patterns for name, phone, and store details.

updatePreferences(Request $request)

Modifies the user's notification and interface preferences including language settings (ar/en), currency (ILS/USD), theme (light/dark), and notification methods (sms/email/whatsapp/push).

getUserReliabilityScore(Request $request)

Calculates and returns a reliability score (0-100) for a user based on phone verification, accurate reports count, completed trades, and account activity/age. Used to establish trust ratings in the barter system.

4.2.4 StoreController (Merchant)

The StoreController handles all operations related to stores on the platform for merchants.

index()

Retrieves and returns a list of all stores belonging to the authenticated merchant user.

store(Request $request)

Creates a new store by receiving input such as store name, address, image, latitude, and longitude. Sets the store status to 'pending' for merchants or 'active' for admins. Prevents merchants from creating multiple stores.

update(Request $request, $id)

Updates the store's information, allowing merchants to modify details like name, address, image, and coordinates. Merchants cannot modify the store status directly.

destroy($id)

Removes a specific store from the system after verifying that it belongs to the authenticated merchant.

updateGeneralSettingOrCreate(Request $request)

Updates existing store settings or creates a new store if one doesn't exist. Handles image uploads and associates the store with a city.

4.2.5 CategoryController

The CategoryController manages the structure of product classification.

index()

Returns a list of all product categories available in the system.

getCategories()

Returns a simplified list of categories with only id and name fields for dropdown selections.

store(Request $request)

Creates a new category based on the submitted name and description, ensuring uniqueness of category names.

update(Request $request, $id)

Updates the name or description of an existing category while maintaining name uniqueness.

destroy($id)

Deletes the specified category, typically restricted to admin-level users.

4.2.6 ProductController

The ProductController is responsible for product management, price updates, and product search for merchants.

index()

Retrieves a paginated list of products belonging to the authenticated merchant's store, including store information and active offers.

getProducts()

Returns a list of all main products with their category information, used for product selection when creating store products.

store(Request $request)

Adds a new product to the merchant's store using the provided product details (product_id, description, price, image). Prevents duplicate products in the same store and triggers UserNotification event.

show($id)

Retrieves a specific product from the merchant's store with full details including store information, user data, reports, and active offers. Calculates and includes store rating.

viewProduct(Product $product)

Retrieves a product for public viewing with comprehensive details including store rating and price rating calculated using PriceRatingService.

update(Request $request, $id)

Edits an existing product's data such as product_id, description, price, or image. When price is updated, triggers PriceUpdated and UserNotification events, and sends email notifications to users who have favourited the product.

destroy($id)

Removes the product from the merchant's store, ensuring it no longer appears in search results.

lastProduct()

Returns a paginated list of the most recently added products across all stores, including store information and active offers.

productHasOffer()

Returns a paginated list of products that currently have active offers, including store information and offer details.

priceRating(Product $product, PriceRatingService $priceRatingService)

Calculates and returns a price rating for a product by comparing its price against similar products in the same category using the PriceRatingService.

4.2.7 NotificationController

The NotificationController handles the delivery and tracking of price alert notifications.

index()

Returns all notifications that belong to the authenticated user, including product information for each notification.

store(Request $request)

Creates a new price alert notification for a user. Accepts product_id, type (lt/gt for less than/greater than), and target_price. Checks existing products against the alert condition and immediately notifies the user if matches are found.

update(Request $request, $id)

Updates the status of a notification (active/inactive) for the authenticated user.

destroy($id)

Deletes the specified notification, removing the price alert for the authenticated user.

changeMethodStauts(Request $request)

Updates the user's notification method preferences (email, sms, whatsapp) by enabling or disabling specific notification channels.

activeNotifications()

Returns paginated active notifications for the authenticated user with unread count, filtered to show only user_notification type notifications.

markAsRead()

Marks all notifications for the authenticated user as read.

4.2.8 BarterController

The BarterController manages the barter/exchange functionality between users.

index()

Retrieves a paginated list of active barters, showing barters where the user is the creator or has been accepted. Includes user information, store ratings, and response status.

publicIndex()

Returns a public paginated list of all barters for browsing.

show($id)

Retrieves a specific barter with full details.

store(Request $request)

Creates a new barter/exchange offer by receiving input such as offer_item, request_item, description, location, image, quantity, contact_method, availability, and exchange_preferences. Handles image uploads and sets status to 'active.'

update(Request $request, $id)

Updates an existing barter's information, allowing the creator to modify all barter details including image replacement.

destroy($id)

Removes a specific barter from the system after verifying ownership.

respond(Request $request, Barter $barter)

Allows a user to respond to a barter offer by creating a BarterResponse with 'pending' status. Prevents users from responding to their own barters or submitting duplicate responses.

acceptResponse(Barter $barter, BarterResponse $response)

Accepts a response to a barter, updating the barter's accepted_by field and marking the response as 'accepted' while rejecting all other responses.

markAsCompleted(Barter $barter)

Marks a barter as completed, changing its status from 'active' to 'completed.'

4.2.9 OfferController

The OfferController manages special offers and discounts for merchant products.

store(Request $request)

Creates a new offer for a product with discount_price or discount_percent, start_date, end_date, and active status. Verifies that the product belongs to the authenticated merchant and triggers NewOfferEvent.

update(Request $request, $id)

Updates an existing offer's details including discount values, dates, and active status. Verifies merchant ownership before allowing updates.

destroy($id)

Removes an offer from the system after verifying that the associated product belongs to the authenticated merchant's store.

4.2.10 FavoriteController

The FavoriteController manages user favourites for price tracking.

index()

Returns a paginated list of all products favourited by the authenticated user, including product details, active offers, and store information.

store($productId)

Adds a product to the user's favourites list for price tracking, creating the favourite if it doesn't already exist.

destroy($productId)

Removes a product from the user's favourites list, stopping price tracking for that product.

4.2.11 SearchController

The SearchController handles advanced search functionality for stores and products.

searchStores(Request $request, MainProduct $product)

Searches for stores that have a specific product, with filtering options and pagination. Applies price rating calculations and store reliability ratings. Returns stores with their products, city information, and calculated ratings.

searchStoresByCategory(Request $request, Category $category)

Searches for stores that have products within a specific category. Applies sophisticated price rating algorithms based on market statistics, handles edge cases like small samples and high variance, and returns stores sorted by price rating score.

4.2.12 ReportController

The ReportController manages user reports about incorrect prices or suspicious products.

index()

Returns a paginated list of all reports with associated product, store, and user information for admin review.

store(Product $product)

Creates or updates a report for a product, incrementing the report count and associating the authenticated user with the report. Triggers NewReportEvent when a new user reports the product.

update(Request $request, Product $product)

Updates a report's status from 'pending' to 'reviewed' by an admin. Increments accurate_reports_count for all users who reported the product when the status changes.

destroy(Product $product)

Deletes a report for a specific product, removing it from the system.

4.3 Database Design

This section explains the database structure of the Nabd platform. It was designed using MySQL to ensure efficient data storage, scalability, and integrity. Each table represents a real entity in the system, and clear relationships connect users, stores, products, and notifications to support all platform functions.

4.3.1 ER Diagram

The ER diagram [20] shows the relationships between the main entities of the database, such as a User belongs to a city, a Store belongs to a User and a City, a Product belongs

to a Store and a Category, an Order belongs to a User and contains multiple Products, and a Notifications belong to a User. Figure 4.2 illustrates the Entity-Relationship (ER) Diagram of the Nabd platform. It shows how the main entities in the database, such as Users, Stores, Products, Categories, Orders, and Notifications—are connected to each other.

4.3.2 Database Tables

This section describes the main database tables used in the Nabd platform. Each table represents a real entity in the system, such as users, products, stores, and categories. The tables are designed in MySQL to ensure data accuracy, scalability, and strong relationships between entities.

Table 4.1 outlines the structure of the users table in the Nabd platform. This table serves as the central repository for user information, capturing essential fields such as name, email, encrypted password, phone number, and address. Uniqueness is guaranteed for the email field, while an auto-incrementing primary key ensures efficient data referencing. Role-based entries distinguish between administrators, merchants, and customers, supporting robust access control and system authorization. Additional columns include account status, localization preferences (language, currency, theme), notification settings, and city association, allowing for a personalized user experience. The schema supports account lifecycle management through timestamps for creation, updates, and soft deletion, while optional fields facilitate features such as location sharing and notification channels. Altogether,

this structure underpins secure authentication, precise authorization, and a flexible user profile system within the Nabd platform.

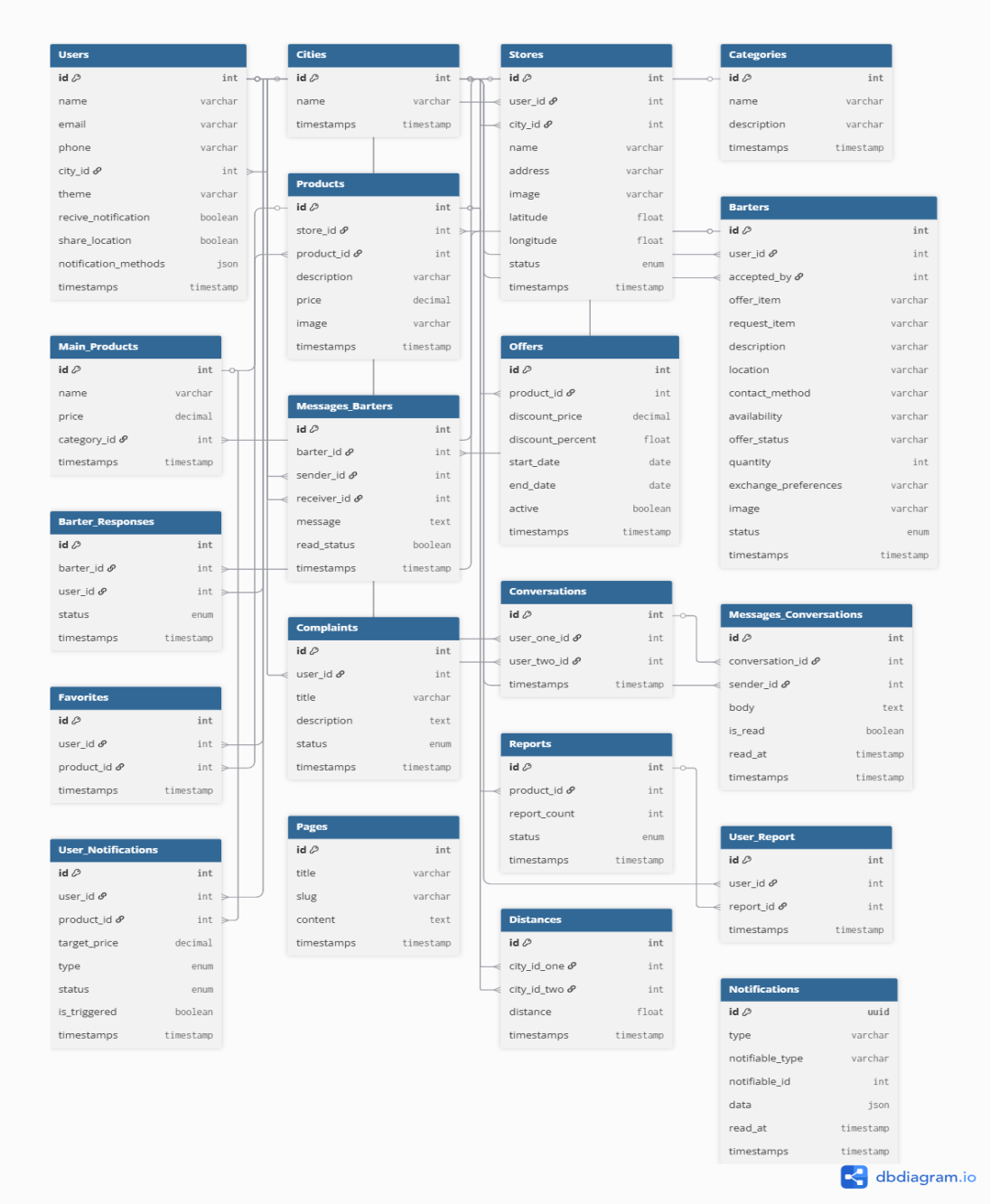


Figure 4.2: ER Diagram

Table 4.1: User Database Table

Column	Type	Attributes	Null	Default	Extra	Link to	Comment
Id	BIGINT UNSIGNED	Primary Key, Auto Increment	No	Null	Auto Increment	-	Primary key for users
Name	VARCHAR(255)	-	No	Null	-	-	User full name
Email	VARCHAR(255)	UNIQUE	Yes	Null	-	-	User email address
Password	VARCHAR(255)	-	No	Null	-	-	Encrypted user password
Phone	VARCHAR(20)	-	Yes	Null	-	-	User phone number
Address	VARCHAR(255)	-	Yes	Null	-	-	User home address
Role	Enum('user', 'admin')	-	No	'user'	-	-	Defines user type
Status	ENUM('active','inactive','pending')	-	No	'pending'	-	-	Account status
Language	VARCHAR(255)		No	'ar'			User's preferred language
Currency	VARCHAR(255)		No	'ILS'			Default currency
Theme	VARCHAR(255)		No	'light'			UI Theme
Notification_methods	JSON		Yes	NULL			Notification channel preferences
Recive_notification	BOOLEAN		No	True			User receives notifications?
Share_location	BOOLEAN		No	False			User shares location?
City_id	BIGINT UNSIGNED	Foreign Key to cities.id	Yes	NULL			Associated city
Remember_token	VARCHAR(100)	Foreign Key to cities.id	Yes	NULL			Session persistence
Deleted_at	TIMESTAMP	Soft Deletes	Yes	NULL			Soft delete timestamp
Created_at	TIMESTAMP	-	Yes	CURRENT_TIMESTAMP	-	-	Date of creation
Updated_at	TIMESTAMP	-	Yes	Null	-	-	Last update date

Table 4.2 defines the structure of the products database in the Nabd platform. It stores all essential information related to each product, including descriptions, prices, available quantities, and associated images. Each product entry is linked to a specific main product (providing the product's name and category) and a store through foreign

key relationships (product_id and store_id). The table also maintains timestamps for creation and last updates, supporting accurate tracking and management of products. With these relationships and data points, the structure supports core platform functionality, including displaying product details, searching, stock management, and associating products with stores and categories.

Table 4.2: Product Database Table

Column	Type	Attributes	Null	Default	Extra	Link To	Comment
Id	BIGINT UNSIGNED	PRIMARY KEY, AUTO INCREMENT	No	NULL	Auto Increment	-	Primary key for products
Store_id	BIGINT UNSIGNED	Foreign Key to stores(id)	No	NULL	-	stores(id)	Linked store
Product_id	BIGINT UNSIGNED	Foreign Key to main_products(id)	No	NULL	-	main_products(id)	Linked main product (name, category)
Description	TEXT	-	Yes	NULL	-	-	Product description
Price	DECIMAL(10,2)	-	No	0	-	-	Product price
Image	VARCHAR(255)	-	Yes	NULL	-	categories(id)	Product image path
Quantity	INT UNSIGNED	-	Yes	NULL (0 if set)	-	stores(id)	Available quantity
Created_at	TIMESTAMP	-	Yes	CURRENT_TIMESTAMP	-	-	Date of creation
Updated_at	TIMESTAMP	-	Yes	NULL	-	-	Last update date

Table 4.3 defines the structure for all registered stores in the Nabd platform. It stores detailed information about each store, including the store name, owner, address, city, geographical coordinates (latitude and longitude), logo, and status. Each store is linked to a user through the user_id field, identifying the store's manager or owner, and optionally to a city through the city_id field for precise regional organization.

The table also includes a status field to indicate whether a store is pending, active, or inactive, alongside timestamps for both creation and updates. This design supports efficient management and organization of stores, enabling key platform features such as store discovery, product display, category filtering, regional availability, and location-based operations.

Table 4.3: Stores Database Table

Column	Type	Attributes	Null	Default	Extra	Link to	Comment
Id	BIGINT UNSIGNED	Primary Key, Auto Increment	No	NULL	Auto Increment	-	Primary key for stores
User_id	BIGINT UNSIGNED	Foreign Key	No	NULL	-	users(id)	Linked user (store owner)
City_id	BIGINT UNSIGNED	Foreign Key	Yes	NULL	-	cities(id)	Store city/location
Name	VARCHAR(255)	-	Yes	NULL	-	-	Store name
Address	VARCHAR(255)	-	Yes	NULL	-	-	Store address
Image	VARCHAR(255)	-	Yes	NULL	-	-	Store logo or cover image
Status	ENUM('pending','active','inactive')	-	No	'pending'	-	-	Store availability status
Created_at	Timestamp	-	Yes	CURRENT_TIMESTAMP	-	-	Date of creation
Updated_at	Timestamp	-	Yes	NULL	-	-	Last update date

Table 4.4 defines the structure of the product categories stored in the Nabd platform. It contains information about each category, including its name, description, image, and status. The purpose of this table is to organize products under specific groups to make it easier for users to browse and for the system to manage product classification. Each category has a unique ID used as the primary key, and the table includes timestamps to track when a category was created or last updated. This design ensures efficient organization and retrieval of categorized data across the system.

Table 4.4: Categories

Column	Type	Attributes	Null	Default	Extra	Comment
Id	BIGINT(20) UNSIGNED	Primary Key	No	NULL	Auto Increment	Primary key for categories
Name	VARCHAR(255)	-	No	NULL	-	Category name
Description	TEXT	-	Yes	NULL	-	Category name
Created_at	TIMESTAMP	-	Yes	CURRENT_TIMESTAMP	-	Date of creation
Updated_at	TIMESTAMP	-	Yes	NULL	-	Last update date

4.3 Interface Design

Interface design of the Nabd Platform focuses on simplicity and responsiveness to provide users with a smooth and efficient experience. The design follows a lightweight and clear structure represented through a simple wireframe that defines the placement of essential elements such as input fields and buttons, ensuring ease of interaction and minimal data consumption. This approach aims to address the issue of weak internet connectivity by creating a fast-loading interface with well-organized and limited components that support the core functionalities, including registration, login, and notification management in line with the platform's overall objectives.

Login Wireframe [21] is a visual prototype or simple sketch that shows the layout and structure of the login page before the actual design or coding stage, Figure 4.3 shows the Login wireframe.

Figure 4.3: Login Wireframe

View Product Wireframe [22] is a visual layout that shows how a single product or item is displayed on the screen, including its main details, image, and actions available to the user, Figure 4.4 shows view product wireframe.

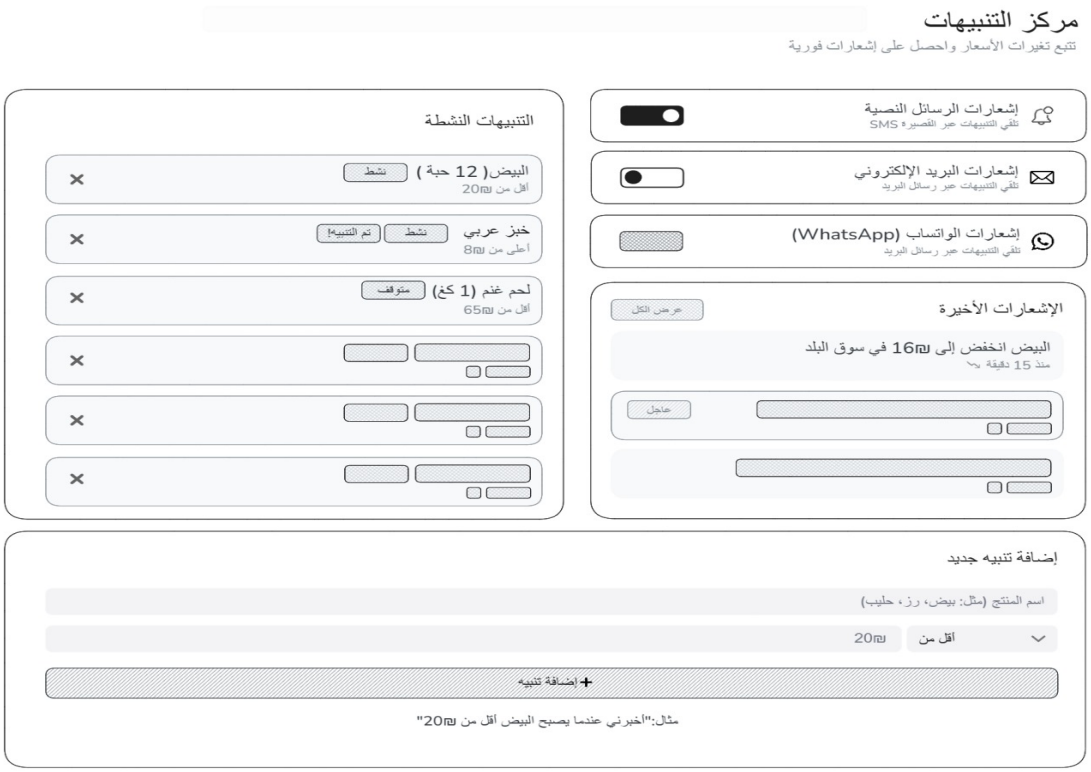


Figure 4.4: View Product Wireframe

Notification Wireframe [23] is a visual prototype that represents the layout and structure of the notification interface, showing how notifications appear to the user within the system, Figure 4.5 shows view notification wireframe.

Figure 4.5 View Notification Wireframe

Chapter 5
Platform Implementation

Chapter 5
Platform Implementation

The implementation of the Nabd platform was carried out in accordance with the design and architecture outlined in Chapter 4. This chapter summarizes the main development process, explaining how the system's components were connected and how its core functions — such as user authentication, account registration, and data operations (retrieving, updating, and deleting records), were implemented. The implementation ensures consistency between the design and the actual system behaviour, achieving the intended functionality and performance of the platform.

5.1 Overall Structure

The implementation of the Nabd platform follows a modular, layered architecture that connects the frontend, backend, and database through well-structured APIs.
The system consists of four main components:

5.1.1 Frontend Layer (Vue.js):

Handles user interaction, dynamic UI rendering, and sending HTTP requests to the backend.

5.1.2 Backend Layer (Laravel API):

Implements business logic, authentication, validation, notifications, events, listeners, and manages all API endpoints.

5.1.3 Database Layer (MySQL):

Stores users, products, stores, categories, notifications, OTP, and logs.
Uses relational tables with FK constraints to ensure data integrity.

5.1.4 Communication Layer (RESTful APIs):

Connects Vue.js components with Laravel controllers, allowing CRUD operations and real-time updates. This layered structure ensures clear separation of concerns, easier

maintainability, secure request handling, and consistent platform behaviour across all modules.

5.2 Functionality

This section explains the main functions implemented in the Nab platform during development. It shows how different parts of the system—such as authentication, product updates, notifications, and exchange offers—work together through the backend controllers and API endpoints.

Figure 5.1 illustrates how the platform manages user login and registration. When a user submits their information through the interface, the controller receives the request, validates the data, and either creates a new account or verifies an existing one. If authentication is successful, the system generates a secure JWT token that allows the user to access protected pages. This process ensures that only authorized users can enter the platform and that all login and registration operations are handled safely.

AuthController.php

```php
public function register(Request $request)
{
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'password' => 'required|string|min:6',
        'email' => 'required_without_all:phone|email|unique:users,email',
        'phone' => 'required_without_all:email|string|size:13|unique:users,phone',
    ]);
    $validator->after(function ($validator) use ($request) {
        if (empty($request->email) && empty($request->phone)) {
            $validator->errors()->add('email', 'يجب إدخال البريد الإلكتروني أو رقم الهاتف.');
            $validator->errors()->add('phone', 'يجب إدخال البريد الإلكتروني أو رقم الهاتف.');
        }
    });
    $validated = $validator->validate();
    $validated['notification_methods'] = [
        'sms' => false,
        'email' => false,
        'whatsapp' => false,
    ];
    $user = User::create($validated);
    session()->regenerate();
    session()->regenerateToken();
    $request->session()->invalidate();
    $request->session()->regenerateToken();
    Auth::login($user);
    event(new JoinNewUserEvent($user));
    return response()->json([
        'message' => ApiMessage::USER_CREATED->value,
        'user' => $user->load(['store.city', 'city']),
    ], 201);
}
```

Figure (5.1): User Authentication & Registration Controller

Figure 5.2 shows how the system processes updates to a product's price and sends immediate alerts to users. When a merchant changes the product price, the controller updates the record in the database and triggers an event. This event activates the notification system, which sends real-time alerts to all users who follow or subscribe to that product. This mechanism ensures that the platform always reflects the latest market prices and keeps users informed instantly.

ProductController.php

```php
public function update(Request $request, $id)
{
    $product = Product::findOrFail($id);
    $validated = $request->validate([
        'product_id' => 'sometimes|required|exists:main_products,id',
        'description' => 'nullable|string',
        'price' => 'sometimes|required|numeric|min:0',
        'image' => 'sometimes|file|image|max:2048',
    ]);
     $oldPrice = $product->price;
     $product->update($validated);
        if (isset($validated['price']) && $validated['price'] != $oldPrice) {
        event(new PriceUpdated($product->id, $product->price));
        event(new UserNotification($product));
        $users = User::whereHas('favorites', function ($q) use ($product) {
            $q->where('product_id', $product->id);
        })->get();
        foreach ($users as $user) {
            $user->notify(new \App\Notifications\ProductPriceUpdated($product));
        }
    }
    return response()->json([
        'message' => ApiMessage::PRODUCT_UPDATED->value,
        'product' => $product
    ]);
}
```

Figure (5.2): Product Update with Real-Time Price Notifications

Figure 5.3 explains how the system handles the creation of an exchange (barter) offer. A user selects the product they want to exchange and submits the offer through the interface. The controller validates the offer details and then saves the new offer in the database. After the data is stored successfully, the system sends a confirmation response back to the user. This functionality allows users to trade goods directly without needing money, supporting community-based exchange during difficult economic situations.

BarterController.php

```php
public function store(Request $request)
{
    $validated = $request->validate([
        'offer_item' => 'required|string|max:255',
        'request_item' => 'required|string|max:255',
        'description' => 'nullable|string',
        'location' => 'nullable|string',
        'image' => 'nullable|image',
        'quantity' => 'nullable|string|max:255',
        'contact_method' => 'nullable|string|max:255',
        'availability' => 'nullable|string|max:255',
        'exchange_preferences' => 'nullable|string|max:255',
        'offer_status' => 'nullable|string|max:255',
    ]);
    if ($request->hasFile('image')) {
        $path = $request->file('image')->store('barters', 'public');
        $validated['image'] = Storage::url($path);
        $fileName = basename($path);
        $source = storage_path('app/public/barters/' . $fileName);
        $destinationDir = base_path('storage/barters');
        if (!is_dir($destinationDir)) {
            mkdir($destinationDir, 0775, true);}
        $destination = $destinationDir . '/' . $fileName;
        if (file_exists($source)) {
            copy($source, $destination);}}
    $barter = Barter::create(array_merge($validated, [
        'user_id' => Auth::id(),
        'status' => 'active',
    ]));
    return response()->json([
        'message' => ApiMessage::BARTER_CREATED->value,
        'barter'  => $barter,
    ]);}
```

Figure (5.3): Barter Exchange Offer Creation

Figure 5.4 shows how the User Model in Laravel is linked to other entities in the database using Eloquent relationships. It highlights that each user can have a store, can create products, receive notifications, and perform orders. These relationships structure the data in a way that makes retrieving and managing user-related information efficient and well organized. The model acts as a central connector between different parts of the platform.

User.php

```php
class User extends Authenticatable
{
    use HasApiTokens, Notifiable, HasRoles, HasFactory;
    protected $fillable = [
        'name', 'email', 'password', 'phone', 'address',
        'role', 'status', 'theme', 'currency',
        'recive_notification', 'notification_methods',
        'city_id', 'share_location', 'accurate_reports_count'];
    protected static function booted(){
        static::updated(function (User $user) {
            if ($user->isDirty('role')) {
                event(new ChangeUserRoleEvent($user));
            }});}
    protected function password(): Attribute{
        return Attribute::make(
            set: fn ($value) => Hash::make($value),);}
    public function store(): HasOne{return $this->hasOne(Store::class);}
    public function barters(){return $this->hasMany(Barter::class);}
    public function favorites(){return $this->hasMany(Favorite::class);}
public function isTrusty(): Attribute{
        return Attribute::make(function () {
            if ($this->role == 'admin')
                return true;
            if ($this->email && $this->phone && $this->share_location && $this->city_id) {
                if ($this->role == 'customer')
                    return true;
                if ($this->role == 'merchant')
                    if ($this->store->name && $this->store->address && $this->store->city_id &&
$this->store->image)
                        return true;
                return false;}});}}
```

Figure (5.4): User Model with Eloquent Relationships

Figure 5.5 demonstrates how authentication data is handled on the frontend, particularly using Vue.js. The Authentication Store saves the JWT token, user information, and session state. It ensures that once a user logs in, their session remains active and secure across all pages of the platform. The store also manages the logout process and attaches the authentication token to API requests, allowing the system to verify the user's identity whenever an action is performed.

auth.js

```javascript
const register = async (credentials) => {
  await getCsrfToken();
  try {
    loading.value = true;
    const response = await axiosClient.post("/register", credentials);
    if (response.status === 201) {
      isAuth.value = true;
      user.value = response.data.user;
      backErrors.value = null;
      router.push({ name: "home" });

      emitter.emit("showNotificationAlert", [
        "success",
        "تم تسجيل الدخول بنجاح!",
      ]);
      isCheckedAuth.value = true;
    }
  } catch (error) {
    backErrors.value = error?.response?.data?.errors;
  } finally {loading.value = false;}};
```

Figure (5.5): Authentication Store

5.3 Database and Data Model

This section describes the structure of the Nab platform's database and explains how the data model was designed to support all system functions. The database was built using MySQL and organized using relational tables that represent real entities such as users, products, stores, and categories. Each table contains specific attributes and is linked to other tables through foreign keys to ensure data integrity. The purpose of this section is to show how the platform stores, manages, and retrieves information efficiently, enabling features such as product browsing, store management, price updates, and notifications. By defining clear relationships between tables, the system ensures consistent and accurate data handling across all modules.

5.3.1 User Database

Figure 5.6 shows the structure of the User Database Table, which stores all information related to users registered on the platform. The table includes basic personal information such as name, phone number, and email, as well as authentication data like passwords and tokens. It also stores user preferences, such as notification methods, and their location details through city ID. Additional fields include timestamps for account creation and updates. This table forms the foundation of the system because it identifies who the users are, how they interact with the platform, and what permissions or roles they have.

id	city_id	name	email	password	phone	address	role	status
1	3	سامي عودة	admin@gmail.com	$2y$12$CnwYvPlTxMsii0QuzWBfu0Pi5JvLOZnP...	+972565656565	NULL	admin	pending
3	1	عمر سليم	omar@mail.com	$2y$12$LOL.kVVUQ4CtIwSgOQAbyeeuNAsu4p...	NULL	NULL	cus...	pending
4	8	زكريا حلبي	zakaria@halabi.ps	$2y$12$RRTKqX0GNP2wdX4tGDxYo.ctXvf1Nnt...	+970591000000	NULL	cus...	pending
6	4	خالد نوفل	khaled@pharma.ps	$2y$12$ZVulUnWOcBksI2vppLqNWOA3nJHQQ...	+970598888888	NULL	mer...	active
7	9	محمد عبدالله	m@g.com	$2y$12$W16e0K40DBO/hBxOUnkOLuuRponTe...	NULL	NULL	mer...	active
9	2	مسعود بدر	mas@g.com	$2y$12$IlEvDHytQ1o7RiXVjZeBBe.VIWWFFSSu...	NULL	NULL	cus...	pending
10	8	ابراهيم حسونة	h@mail.com	$2y$12$vhMHiqccSqdNVKbEWPLaO.pMdE4zb/H...	+970592111111	NULL	mer...	active
11	4	أحمد سلامة	ah@mail.com	$2y$12$oUdi2.K7t.p.49PSaAXB.uAODLR6kTmD...	+970596000000	NULL	mer...	active
15	4	محمد إياد الدالي	eyad@mail.com	$2y$12$pSb26HxDCOxSGGjYnTL8teyccPeo56f...	+970593000001	NULL	mer...	active
22	1	حسام هاني	os@g.com	$2y$12$eBNsg6JxV6TbsYPhCOlIPuowJs7OFHx...	+970595000049	NULL	cus...	pending
23	NULL	فادي شراب	dih@gmail.com	$2y$12$Y8eUwsQwljLZFRPs0iuk6.o9oOWRgWl...	NULL	NULL	cus...	pending

Figure (5.6): User Database

5.3.2 Product Database

Figure 5.7 presents the Product Database Table, which stores details for every product listed in the system. Each product entry includes a reference to the store that sells it and the main product category it belongs to. The table contains attributes such as price, description, image, and available quantity. It also includes timestamps for creation and updates to support tracking changes over time. This table is essential for displaying products to users, filtering items by category, updating prices, and managing stock levels within each store.

id	store_id	description	price	image	created_at	updated_at	produ
2	2	أكياس رمل للبيع بالجملة - متينة ومسامية للماء	350.00	products/HLe2qh3U7MtRhXHnByanthy2EHvEpG...	2025-10-23 12:32:22	2025-10-23 12:32:22	7
3	1	Xiaomi Note 20 Pro	650.00	products/4sQjgqR2Km3MIGTKXPNBE2tuKzTqpA...	2025-10-23 12:32:22	2025-10-23 12:32:22	4
4	1	شاحن Type C	75.00	products/1z4mZyUkZJ1SWfEDRmkerHrSHoEPJ6...	2025-10-23 12:32:22	2025-10-23 12:32:22	8
8	4	معطر golden tex ورق	30.00	products/Yy7anmuHdgfwZZGIyItFwfj8onSaEGA...	2025-10-23 12:32:22	2025-10-23 12:32:22	15
9	4	قطرات systane	22.00	products/9rRObeBG8VV3VgXpgYcdQslHRSjS4sj...	2025-10-23 12:32:22	2025-10-23 12:32:22	16
10	4	عدد 50"goodcare" كمامات	35.00	product products/9rRObeBG8VV3VgXpgYcdQslHRSjS4sjUlpZZ02pg.webp	[illegible]	[illegible]5-10-23 12:32:22	17
11	6	فول أمريكانا عدد 1	30.00	products/875AlEvRY5e5byE9O0COD0Ffcy8VD5...	2025-10-23 12:32:22	2025-10-23 12:32:22	2
12	6	حليب المراعي كامل الدسم	55.00	products/MbHBVN2j6pDAuI08QJIaiKmGuvCOed...	2025-10-23 12:32:22	2025-10-23 12:32:22	11
14	6	سكر ناعم أبيض صافي	5.00	products/0B7w3FJpPQKaLkikFPnfqKYJ3sZ2QWo...	2025-10-23 12:32:22	2025-10-23 12:32:22	20
15	7	أرز بسمتي أبيض 1 كيلو صافي	9.00	products/TAzTLHKNS675nVp7p2SznOrVBkjIwQz...	2025-10-23 12:32:22	2025-10-23 12:32:22	10
16	8	أرز حبة قصيرة 2 كيلو	36.00	products/8yUUiF51iugfk1qfyrS77x0Vkv8QvoJLs...	2025-10-23 12:32:22	2025-10-23 12:32:22	10
17	8	زيت دوار الشمس 5 لتر	77.00	products/W4OlLB3lcaoYv4TxmYIIaAWdZnvsIpN...	2025-10-23 12:32:23	2025-10-23 12:32:23	26
18	8	جبنة بيضاء تركية	13.00	products/Zb4FOHAJPeqcdvtdmhLmXdFpH3Rq5...	2025-10-23 12:32:23	2025-10-29 11:14:37	23
19	8	قشطة بوك عدد 1	22.00	products/wLp99TYDeo333QzvMKWouWopl45kH...	2025-10-23 12:32:23	2025-10-23 12:32:23	25
20	8	عصير تفاح أورجينال 1 لتر	15.00	products/K0ZrBXsBJK12LLsV9GOxpYZDq5dxDi7...	2025-10-23 12:32:23	2025-10-23 12:32:23	27
21	8	[illegible] Nestle [illegible]	10.00	products/G4qKTW6MRLsctODAn32dei7Fi6VCCn	2025-10-23 12:32:23	2025-10-29 11:45:19	22

Figure (5.7): Product Database

5.3.3 Stores Database

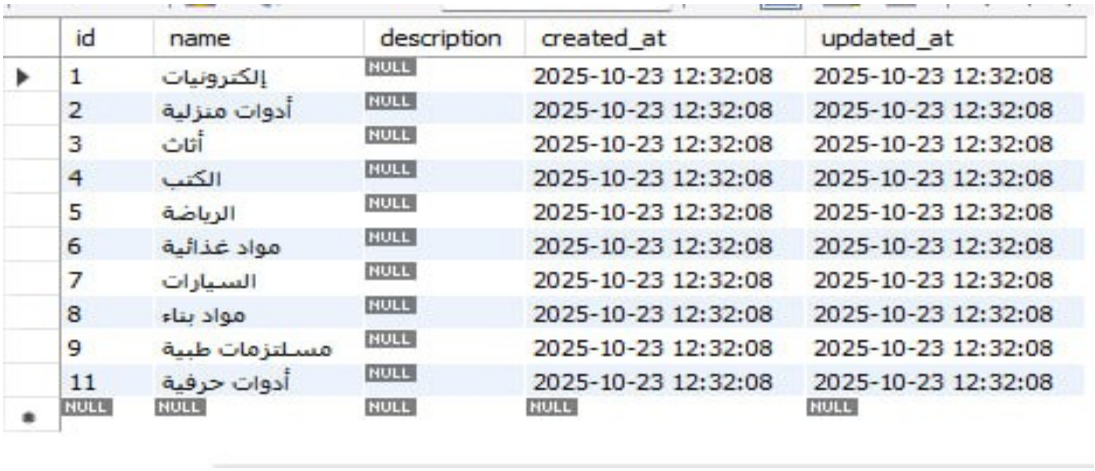
Figure 5.8 describes the Stores Database Table, which contains information about all stores registered in the platform. The table includes fields such as store name, address, image, owner ID (linked to the user table), and location through the city ID. It also stores the store's status (pending, active, or inactive), allowing administrators to manage and verify stores before they appear to the public. By organizing store data clearly, this table enables users to browse shops, view their products, and access store-specific information.

id	user_id	city_id	name	address	image	latitude	longitude	status	created_at
1	2	5	Gaza Tech	غزة - شارع الوحدة	stores/uSldQEkCt9nFJss4azKaOG7aZ6yqhYdbQ...	NULL	NULL	active	2025-10-23 12:32:19
2	4	8	حلبي للمقاولات	شارع البركة	stores/IGP5NUG4xQhSLXQGNlT95xQAd0b0H8s...	NULL	NULL	active	2025-10-23 12:32:22
4	6	4	صيدلية الحلو	شارع النصر	stores/XgOmubT2NWxVeUppzE7l8fVU1SWjlt4f...	NULL	NULL	active	2025-10-23 12:32:22
5	7	9	أبو حمد للمواد الغذائية	شارع النص	stores/2RalQLtnHxiOQTsdrFgGkzzYPaIRKGLJ6k...	NULL	NULL	active	2025-10-23 12:32:22
6	8	9	أبو حميد للمواد الغذائية	شارع البحر	stores/nDhJlGV4swu9gKyJ4QFcRNAHu9m92hqr...	NULL	NULL	active	2025-10-23 12:32:22
7	10	4	بقالة حسونة	شارع عمر المختار	stores/r1l0ELOV6EPzxvT54F0V958Vho6lPPdYD...	NULL	NULL	active	2025-10-23 12:32:22
8	11	4	العمدة مول	غزة - تل الهوا	stores/RFEol34KzOHRpFzcGPHO5ynw5JIUzTG0...	NULL	NULL	active	2025-10-23 12:32:22
10	3	1	NULL	NULL	NULL	NULL	NULL	pending	2025-10-23 12:32:22
12	15	4	محل إياد للمنظفات	غزة - شارع الوحدة	stores/Z8lzVdIYQB9LDHghTkD80VEoxbflfyq7hn...	NULL	NULL	active	2025-10-23 12:32:22
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure (5.8): Stores Database

5.3.4 Categories Database

Figure 5.9 shows the Categories Database Table, which stores product categories used across the platform. Each category includes a name, description, image, and timestamps. Categories help group products into logical sections, making it easier for users to search and browse items. The table also simplifies backend management by allowing administrators to add, edit, or remove categories easily, ensuring a well-organized product structure within the system.



id	name	description	created_at	updated_at
1	إلكترونيات	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
2	أدوات منزلية	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
3	أثاث	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
4	الكتب	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
5	الرياضة	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
6	مواد غذائية	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
7	السيارات	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
8	مواد بناء	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
9	مستلزمات طبية	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
11	أدوات حرفية	NULL	2025-10-23 12:32:08	2025-10-23 12:32:08
NULL	NULL	NULL	NULL	NULL

Figure (5.9): Categories Database

5.4 User Interface

This section explains the overall system architecture of the Nab platform. It describes how the application is built using a layered structure that separates the frontend, backend, and database. The frontend communicates with the backend through RESTful APIs, while the backend processes requests, applies business logic, and interacts with the database. The architecture ensures scalability, maintainability, and secure data flow across the system. It also highlights how different components—such as authentication, product management, notifications, and data storage—work together to provide a smooth user experience and efficient system performance.

5.4.1 Home Page

Figure 5.10 shows the main Home Page of the Nab platform. The interface displays essential product categories, latest offers, and quick access to stores. It is designed to provide a smooth browsing experience where users can easily search for items, explore

discounts, and navigate through different sections of the platform. The home page acts as the central entry point for users to access all features.

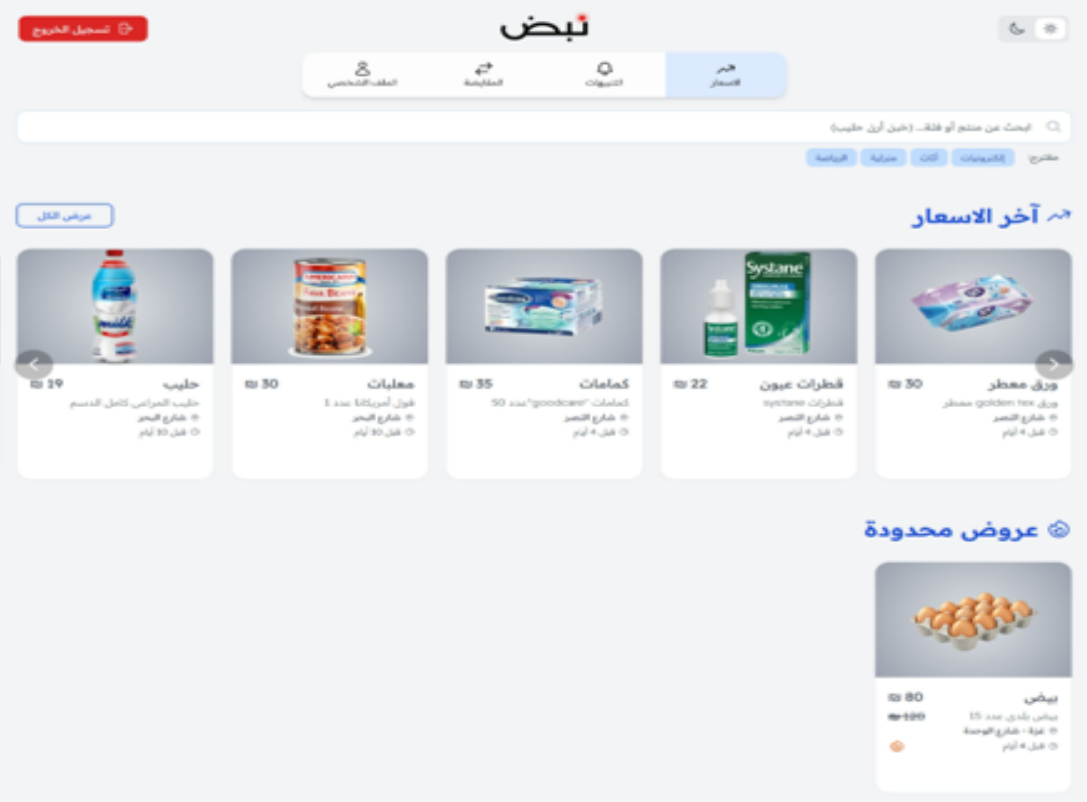


Figure (5.10): Home Page

5.4.2 Admin Dashboard

Figure 5.11 presents the Admin Dashboard, which is the control panel used by administrators to manage the entire system. Through this dashboard, the admin can

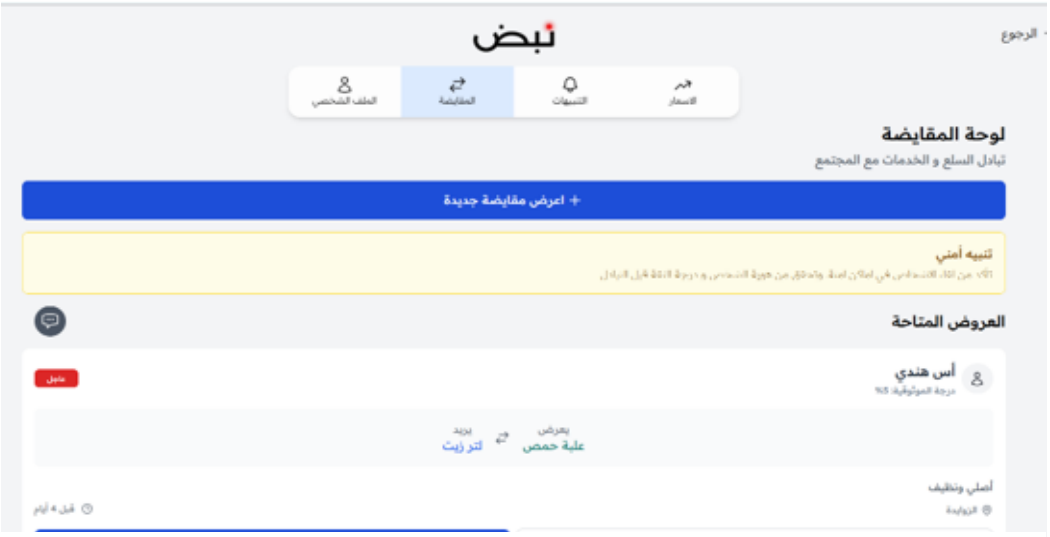


Figure (5.11): Admin Dashboard

monitor users, stores, products, and categories. It includes tools for reviewing store requests, managing product approvals, and overseeing system activities. The dashboard offers clear statistics and organized menus, enabling efficient management of the platform.

5.4.3 Exchange Screen

Figure 5.12 illustrates the Exchange Screen, where users can view and manage product exchange (barter) offers. The screen displays available exchange requests, details of products offered, and the user's own offers. It allows users to accept, reject, or create exchange proposals. This interface supports the system's goal of encouraging product swapping as an alternative to purchasing.

Figure (5.12): Exchange Screen

5.5.4 Notification Screen

Figure 5.13 shows the Notification Screen, which displays all alerts sent to the user. Notifications may include price changes, store updates, discount offers, and responses to exchange requests. The screen organizes notifications in a clean layout, ensuring users can quickly see what has changed and respond to important updates.

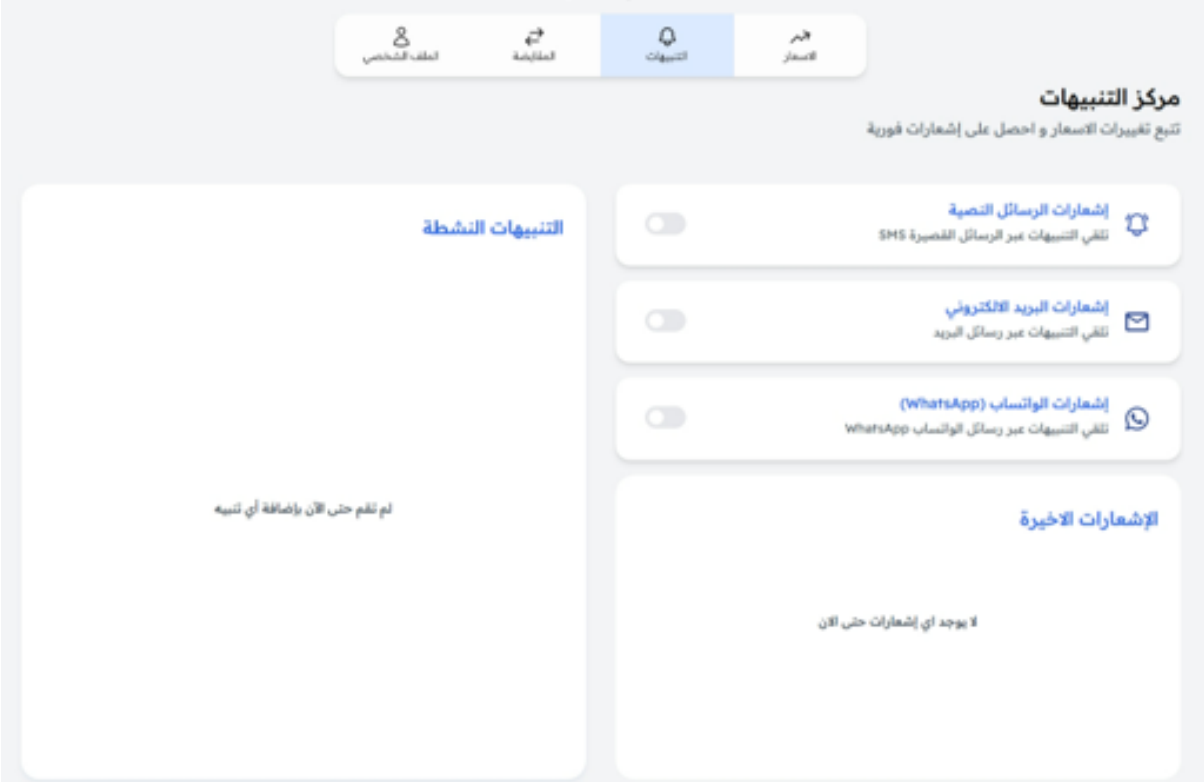


Figure (5.13): Notification Screen

Chapter 6
Testing

Chapter 6
Testing

Testing phase aims to ensure that the Nab Platform functions correctly, reliably, and efficiently under different conditions. Testing verifies that all functional and non-functional requirements defined in earlier chapters have been met and that the platform performs as expected when interacting with users, stores, and the database.

6.1 Functional Testing

Functional testing [24] is a type of software testing that focuses on verifying that each feature or function of a system works according to the specified requirements.
In other words, it checks what the system does, not how it does it.

6.1.1. User Login Test

A User Login Test [25] is a type of functional testing that verifies the login feature of a system or website works correctly and securely.
It ensures that users can access their accounts only with valid credentials and are blocked if they enter incorrect ones.

Objective: Verify that registered users can successfully log in with valid credentials and that invalid credentials are rejected.

Expected Result:

1. Successful authentication with JWT token generation → Redirected to dashboard.
2. Invalid credentials → error message “Invalid email or password.”

Result:

Passed — authentication and error handling work correctly

Figure 6.1 illustrates the result of testing the user login process with invalid credentials. It shows how the system detects incorrect email or password inputs and prevents unauthorized access. An error message appears to inform the user that the login attempt failed, confirming that the validation and security functions work correctly.

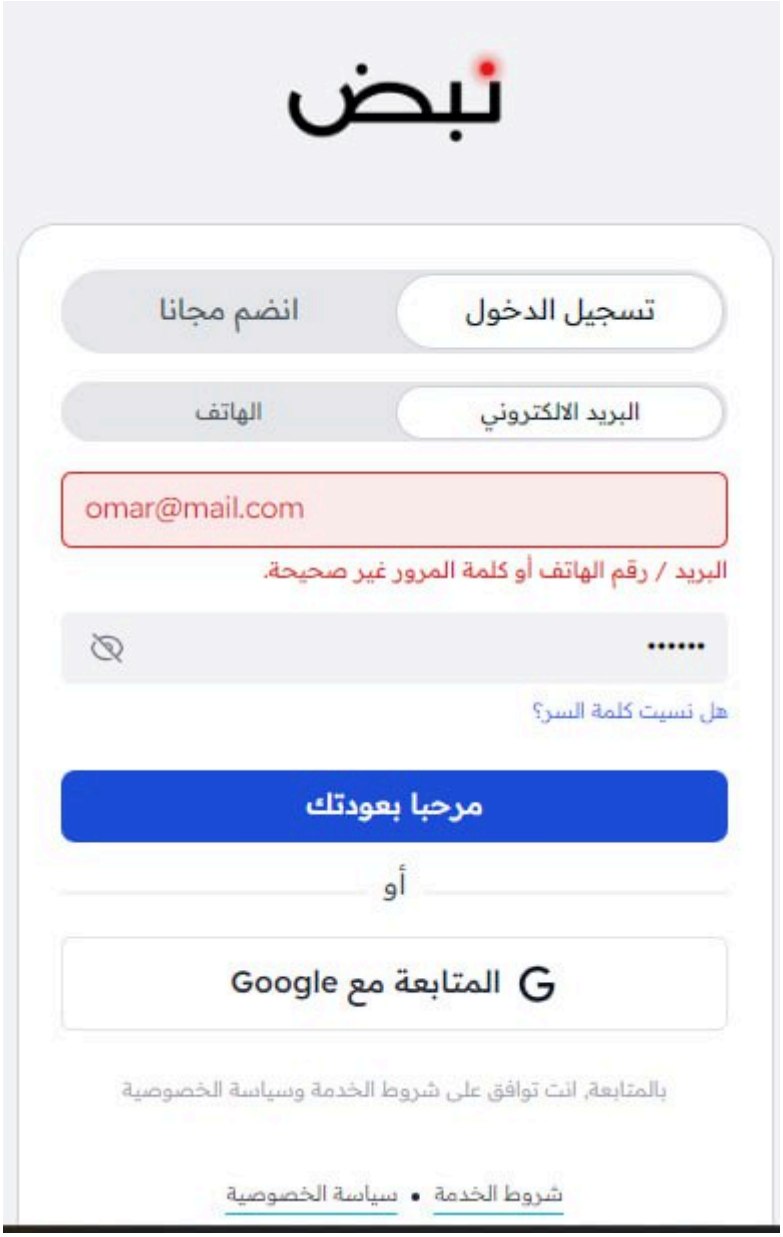


Figure 6.1: User Login Test Invalid

6.1.2 Admin Login Test

An Admin Login Test [26] is a functional test that focuses on verifying that the administrator login feature of a system works correctly and securely.
It ensures that only authorized admins can access the admin dashboard, and that all security and validation rules are properly applied.

Objective: Ensure that administrators can access the backend securely, while unauthorized users are blocked.

Expected Result:

1. Admin authenticated successfully → Access granted only for admin roles.
2. Non-admin → 403 Forbidden.

Result: Passed — secure authorization implemented.

Figure 6.2 illustrates the result of testing the administrator login process with valid credentials. It shows that when the correct email and password are entered, the system successfully authenticates the admin and grants access to the dashboard. This confirms that the admin authentication and authorization mechanisms are functioning properly and securely.

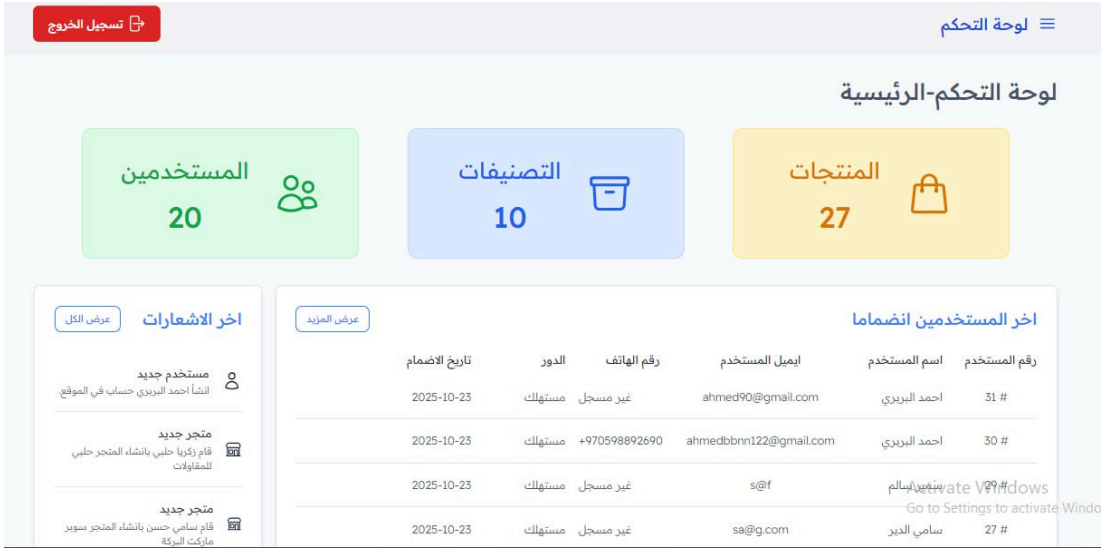


Figure 6.2: Admin Login Test Successfully

6.2. Unit Testing

Unit testing [27] focuses on testing individual components or functions of the system in isolation, for example, a single function, method, or class.
It's usually done by developers during the coding phase.

Purpose: Verify that each function works correctly in isolation.
Unit tests were written using PHPUnit for core modules (Controllers, Models).

Sample Code:

Test

```
public function store (Request $request) {
        $validated = $request->validate ([
            'name' => ['required', 'string', 'max:255'] ,
            'price' => ['required', 'int', 'min:5'] ,
            'category_id' => [ 'required', 'int', 'exists:categories,id'],
        ] );
        return MainProduct::create( $validated );
    }
```

Result: All units passed successfully**.**

Figure 6.3 illustrates the result of testing the "Add Product" function in the admin panel. It shows that when the administrator enters valid product information, such as name, price, category, and store, the system successfully saves the data in the database and confirms the operation with a success message. This verifies that the product management module works correctly and allows admins to add new items without errors.

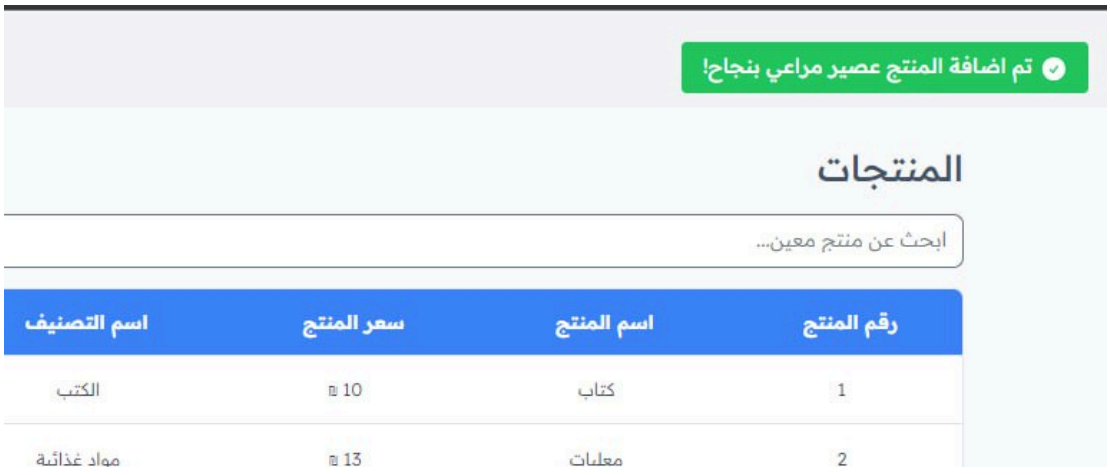


Figure 6.3 Add Product (Admin) Successfully

6.3. Integration Testing

Integration testing [28] checks how different modules or units work together after being combined.
It's done after unit testing to ensure that components communicate correctly.

Objective: Confirm that all modules work together and that data flows correctly between frontend, backend, and database.

Table 6.1 summarizes the main integration test scenarios used to verify how different system modules work together. Each scenario checks the interaction between connected components such as authentication, product management, and notifications. The results confirm that the integrated modules function correctly and deliver the expected system behavior.

Table 6.1: Integration Test Scenarios

Test Modules	Integrated With	Expected Result
Login + Dashboard	Authentication + Product Display	Dashboard loads with user data
Admin Add Product → User Search	ProductController + Search API	Product appears in search results
Create Alert → Notification	Alert Module + NotificationController	Notification sent successfully

Result: All integrations performed successfully.

Figure 6.4 illustrates the result of testing the product search function on the Nab platform. It shows that when a user enters a valid keyword in the search bar, the system retrieves matching products from the database and displays them correctly on the interface. This confirms that the search module functions properly, providing users with accurate and fast results.

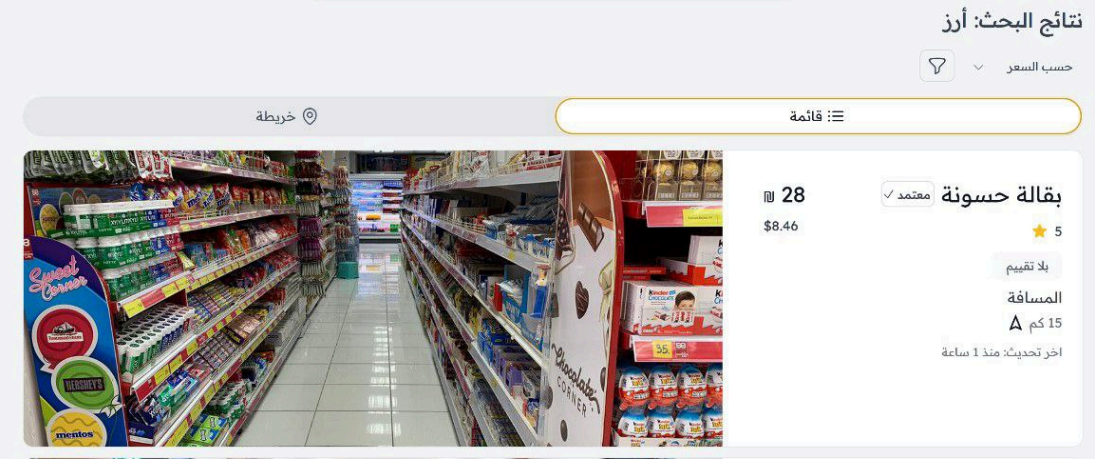


Figure 6.4 Search Results Successfully

6.4. Acceptance Testing

Acceptance testing [29] was conducted to ensure that the Nabd platform meets all business requirements and performs correctly from the perspective of real end users. This phase represents the final verification step before deployment, confirming that the system's functions, usability, and performance align with the expectations defined during the requirement analysis stage. The testing was executed with actual users representing different roles within the system, including customers, merchants, and administrators.

6.4.1 Acceptance Criteria

To measure the correctness and completeness of the platform, clear acceptance criteria were established:

1. Users must be able to register, log in, and reset their passwords successfully.
2. The search module should retrieve accurate product results with appropriate filtering options.
3. Platforms must update and display price changes in real time.
4. Notifications must reach the intended users through the selected delivery channels (in-platform, SMS, WhatsApp).
5. Administrators must be able to manage products, categories, stores, and user accounts without system failures.
6. The interface should load effectively under low-bandwidth conditions common in Gaza.

7. All platform operations must demonstrate correct validation, error handling, and secure access control.

6.4.2 Test Plan

Acceptance tests [30] were carried out in a controlled environment designed to resemble real operating conditions.

The plan included:

1. Participants: 10 real users (5 general users, 3 merchants, and 2 administrators).
2. Environment: Mobile devices and laptops using standard internet and low-bandwidth mobile networks.
3. Testing Tools: Browser interface, mobile interface
4. Scope: All major use cases presented in Chapter 3, covering user operations, admin operations, notifications, browsing, and search.

6.4.3 Execution of Test Scenarios

1. User Registration and Login:

Participants attempted to create accounts and log in using email or phone credentials.

Result: Successful authentication using JWT tokens, with accurate handling of invalid inputs.

2. Product Search and Filtering:

Users searched for basic goods such as “flour” and “sugar” using keywords and category filters.

Result: The system displayed accurate matches with fast response and correct filtering.

3. Store and Category Browsing:

Users viewed store profiles, category lists, and product groupings.

Result: All pages loaded properly, and cached data improved performance under weak internet.

4. Price Update and Alerts:

Admins updated product prices while users subscribed to price change alerts.

Result: Notifications were delivered within acceptable time frames on supported channels.

5. Admin Panel Operations:

Administrators performed CRUD operations for stores, products, categories, and user accounts.

Result: All operations functioned correctly, and updated records became visible immediately.

6.4.4 Issues Observed and Resolutions

During acceptance testing, several minor issues were identified:

1. Slow initial loading on weak connections.

Resolution: API response caching was optimized to reduce load times.

2. Duplicate notifications for some users.

Resolution: The notification queue handler was adjusted to prevent repeated dispatch.

3. Delayed store verification updates.

Resolution: Model caching was synchronized to reflect changes more reliably.

All identified issues were resolved and retested to ensure stability.

Final Result:

The Nabd platform has been fully accepted by end users and is considered ready for real-world operation.

Chapter 7

Conclusion and Recommendations

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

The development of Nabd which is a real-time price tracking platform for essential goods in the Gaza Strip has successfully achieved its primary objective providing the people of Gaza with an accessible, reliable, and user-friendly digital platform that delivers up-to-date information on product prices, availability, and trusted sellers. The platform addresses a vital humanitarian and economic need in Gaza, where market instability, limited imports, and the closure of commercial crossings have made daily purchasing decisions both difficult and risky. By centralizing accurate market data and offering real-time notifications through multiple communication channels (such as SMS and WhatsApp), Nabd empowers users to make informed choices, reduces the risk of exploitation, and supports transparency and fairness in local trade. Technically, the system's architecture based on Laravel (MVC) and Vue.js has proven to be efficient and modular, allowing for smooth interaction between the backend and frontend. The use of MySQL ensured reliable data storage and retrieval, while testing through Postman and PHPUnit confirmed the system's stability and performance. Through this project, the team not only built a functional platform but also deepened their understanding of software engineering practices — including requirement analysis, modular design, implementation, testing, and deployment — while tackling a real-world problem that directly affects their community. In conclusion, the Nabd platform demonstrates how technology can serve as a bridge between necessity and accessibility, turning information into a life-saving and time-saving tool for thousands of residents in Gaza.

7.2. Recommendations

While the current version of Nab fulfills its primary goals, there are several opportunities for improvement and expansion in the future:

1. **Mobile Application:** Develop an Android and iOS version to make the platform more accessible and user-friendly.
2. **English Language Support:**
   The platform currently operates entirely in Arabic, and future updates will

include full English language support to make it accessible to a wider audience, including international users and humanitarian organizations.

3. **Expansion Beyond Gaza:**
   Extend the platform to cover other Palestinian regions facing similar economic challenges, adapting data collection and vendor integration accordingly.
4. **AI-Based Price Prediction:**
   Implement machine learning algorithms to analyse market data and forecast future price trends.
5. **Offline Mode:**
   Enable users to access previously stored data when the internet connection is weak or unavailable.
6. **Collaboration with Local Authorities and NGOs:**
   Establish partnerships with local municipalities, consumer protection bodies, and humanitarian organizations to verify data accuracy and expand the platform's social impact.

References

[1] GeeksforGeeks. (2025, September 29). Software engineering: Iterative waterfall model.

https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/

[2] TutorialsPoint. (2025). Laravel tutorial: Laravel overview.

https://www.tutorialspoint.com/laravel/index.htm

[3] Vue.js. (2025). The progressive JavaScript framework.

https://vuejs.org/

[4] Brains, J. (2015, November 2). Component Source: About PhpStorm.

https://www.componentsource.com/product/phpstorm/about

[5] MySQL. (2025). About MySQL.

https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html

[6] Excalidraw. (2023). What is Excalidraw?

https://excalidraw.com/

[7] Twilio. (2025). Connect digital experiences across channels.
https://www.twilio.com/

[8] Association for Project Management. (2024). What is scheduling in project management? APM.

https://apm.org.uk/resources/what-is-project-management/what-is-scheduling-in-project-management/

[9] International Food Policy Research Institute. (2025). The Food Security Portal: A platform that tracks food prices.

https://www.foodsecurityportal.org/

[10] Datasembly. (2025). Web-based platform that tracks grocery prices.

https://www.datasembly.com/

[11] PriceSpy. (2025). Online price comparison platform.

https://pricespy.co.uk/

[12] Google Shopping. (2025). Online shopping platform that allows users to search for products.

https://www.google.com/shopping

[13] Wadi. (2025). Web application to buy products online.

https://www.wadi-store.com/

[14] Noon. (2025). Online shopping platform that offers competitive pricing.

https://www.noon.com/uae-en/

[15] Wright, G. (2025, August 12). What are functional requirements? TechTarget.

https://www.techtarget.com/whatis/definition/functional-requirements

[16] GeeksforGeeks. (2025, July 15). Non-functional requirements in software engineering. Retrieved December 7, 2025, from

https://www.geeksforgeeks.org/non-functional-requirements-in-software-engineering/

[17] GeeksforGeeks. (2025, July 23). Use Case Diagram – Unified Modeling Language (UML). Retrieved December 7, 2025, from

https://www.geeksforgeeks.org/use-case-diagram/

[18] Visual Paradigm. (n.d.). What is Use Case Specification? Retrieved December 7, 2025, from

https://www.visual-paradigm.com/guide/use-case/what-is-use-case-specification/

[19] Wisbey, O. (2023, March 31). What is a sequence diagram? TechTarget. Retrieved December 7, 2025, from

https://www.techtarget.com/searchsoftwarequality/definition/sequence-diagram

[20] IBM. (2025). What is an entity relationship diagram? IBM Think. Retrieved December 7, 2025, from

https://www.ibm.com/think/topics/entity-relationship-diagram

[21] Wireframe (design). (n.d.). What is Wireframe? What If Design Glossary. Retrieved December 7, 2025, from

https://www.whatifdesign.co/design-glossary/wireframe

[22] Feng, S., Yuan, M., Chen, J., Xing, Z., & Chen, C. (2023). Designing with Language: Wireframing UI Design Intent with Generative Large Language Models

https://arxiv.org/abs/

[23] Wireframe (design). (n.d.). What is a wireframe? ProductPlan Glossary. Retrieved December 7, 2025, from

https://www.productplan.com/glossary/wireframe/

[24] GeeksforGeeks. (2025, July 11). Differences between functional and non-functional testing. Retrieved December 7, 2025, from

https://www.geeksforgeeks.org/software-engineering/differences-between-functional-and-non-functional-testing/

[25] Katalon Team. (2025). Functional Testing in Software Testing: Types, Tools & Free Template. Retrieved December 7, 2025, from

https://katalon.com/resources-center/blog/functional-testing

[26] Katalon Team. (2025). Functional Testing in Software Testing: Types, Tools & Free Template. Retrieved December 7, 2025, from

https://katalon.com/resources-center/blog/functional-testing

[27] Software Testing Help. (2025). Unit testing in software testing: A complete guide. Retrieved December 7, 2025, from

https://www.softwaretestinghelp.com/unit-testing/

[28] Guru99. (2025). Integration testing in software testing. Retrieved December 7, 2025, from

https://www.guru99.com/integration-testing.html

[29] GeeksforGeeks. (2025). Acceptance testing in software engineering. Retrieved December 7, 2025, from

https://www.geeksforgeeks.org/software-engineering/acceptance-testing-software-testing/

[30] TechTarget. (2024, October 22). What is acceptance testing? Retrieved December 7, 2025, from

https://www.techtarget.com/searchsoftwarequality/definition/acceptance-test