Block-Chain Based Product Authenticity Verification System

Danish Rafique
(01-135221-011)

Luqman Matloob
(01-135221-023)

Supervisor: Dr. Faisal Imran

Bachelor of Science in Computer Science

Department of Computer Science
Bahria University, Islamabad
December 2025

C e r t i f i c a t e

We accept the work contained in the report titled "**Blockchain-Based Product Authenticity Verification System**", written by Mr. **Danish Rafique** and Mr. **Luqman Matloob** as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by:

Supervisor: Dr. Faisal Imran

Internal Examiner:

External Examiner:

Project Coordinator: Dr. Kabeer Ahmed Bhatti (Fyp Coordinator IT Department)

Head of the Department: Dr. Khurram Ehsan (Head Of Department Computer Science)

December, 2025

Abstract

The rise in counterfeit goods in recent years has caused significant issues for sellers, consumers, and law enforcement. This project presents a blockchain-based system to ensure that products are authentic and that the path from stores to buyers is transparent. However, traditional centralized product registries are vulnerable to tampering and lack verifiable end-to-end traceability, making provenance checks unreliable and multi-stakeholder coordination difficult. Sellers can use this system to create stores and register products under them, with each product assigned a unique product ID. Users can search for products using product ID, before making purchase request. Sellers, buyers, and administrator are among the various user categories in the system, each has distinct responsibilities. Smart contracts reduce the possibility of data alteration by keeping product information secure and publicly accessible. To illustrate how the system functions and how users interact with it, various diagrams were created, including activity diagrams, use case models, and sequence diagrams. This system aims to prevent counterfeit goods, boost consumer trust, and provide a secure, collaborative method of verifying product authenticity by utilizing the transparent and reliable aspects of blockchain technology.

Acknowledgements

We extend our sincere gratitude to Allah Almighty for His countless blessings, guidance, and strength throughout the completion of this project. This work would not have been possible without His mercy and support.

We extend our sincere gratitude to our supervisors and faculty for their guidance, insightful feedback, and encouragement throughout this project. Their support played a vital role in shaping the direction and quality of our work.

We also thank our peers and lab colleagues for their constructive discussions and technical support, as well as the open-source community whose tools and libraries made this work possible. In addition, we are grateful to the testers and early users who provided valuable feedback that improved the system's usability and robustness.

Finally, we are deeply thankful to our families and friends for their patience, motivation, and unwavering support throughout this journey.

Danish Rafique & Luqman Matloob
Bahria University
Islamabad, Pakistan
December 2025

"The great thing about blockchain is that it allows us to trust without trusting anyone."

Vitalik Buterin, Co-founder of Ethereum

Contents

List of Tables

List of Figures

Acronyms & Abbreviations

AES	Advanced Encryption Standard
AES-256	Advanced Encryption Standard 256-bit
CBC	Cipher Block Chaining
AES-256-CBC	Advanced Encryption Standard 256-bit Cipher Block Chaining
JWT	JSON Web Token
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
JSON	JavaScript Object Notation
UI	User Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
URL	Uniform Resource Locator
DB	Database
ABI	Application Binary Interface
ERC	Ethereum Request for Comments

Chapter 1
Introduction

Brands and customers are getting into trouble given the rise in the availability of fake products in used and new markets. Brand impersonation is another problem that luxury brands can frequently face as counterfeits are sold in their names. Meanwhile, on their part, buyers, particularly in the second-hand markets, struggle to verify whether a good is genuine and in the right possession.

- **In New Products:** In the case of the blockchain, the blockchain system system allows sellers to register their products under the store, which are then verified and solidified by the admin.This prevents the sale of counterfeit products under this brand name and it also enables consumers to ascertain that the product is produced by the brand endorsed.
- **In Used Products:** Buyers can determine whether a product is genuine and see the entire history of change of ownership using the product ID. This serves to establish confidence in the second-hand market and prevents customers from being duped into purchasing fakes or stolen items. .

This system enhances transparency, traceability and trust in the entire product life cycle by ensuring security to both brands and customers.

1.1 Problem Statement

Although online and second-hand markets become increasingly common, there is still no universal and safe method to ensure that the product is not a fake and does not belong to its owner. Black marketeers may misuse brand names, and companies may be at the receiving end when their brands are used without their authorization and buyers are deceived to purchase improper products.

Among the essential issues are:

- Advertisers claiming to be dealing with genuine products although they are counterfeits, using brand names that are common.
- Prospective customers who want to buy used goods have no means of verifying whether an item is authentic or stolen.
- Consumers are used to paper receipts and word of mouth to obtain confirmation that a particular product is genuine and that fakes and losses can exist.

The proposed project is meant to address these problems by offering a technology solution that could ensure verification of product authenticity and ownership, and ensure whether the product is a new one or has undergone the resale process.

1.2 Project Motivation

Fake products in both the local and international markets have been on the rise, and this has made it essential for sellers and buyers to identify whether the products are real. Counterfeiters will tend to copy high-end brands and expensive products to their disadvantage, deteriorating the image of the brand and resulting in a loss of money. Moreover, customers, particularly those who try the second-hand market, cannot distinguish between a fake item and a product that actually belongs to the seller. Most of the currently deployed verification techniques rely on central databases, physical security features, or paper-based records, which can be compromised or misplaced [5]. It is evident that what is required is more more secure, transparent, and immutable, fostering trust for both buyers and sellers. This necessity resulted in the creation of a blockchain-based verification platform on the basis of which unique Product IDs (PIDs) are generated to verify the authenticity of a product and ensure its ownership history is being traced, all the way until its production and resale.

1.3 Project Objectives

- To develop a blockchain run platform where brands will be able to register actual product that will enable them save the brand identity and reputation.
- To create undeletable and non-duplicable Product IDs (PIDs) that serve as digital identifiers for every product registered within the platform.
- To provide consumers, especially those in the secondary markets, with a safe and clear method of verifying whether a product is an original and to view its previous change of ownership.
- To build stronger trust among manufacturers, sellers, and customers in both new and used product markets.

1.4 Project Scope

The system will be designed as a blockchain-based system to check the authenticity and ownership of products. It is targeted on two situations, namely:

- First-hand use cases: Stores can register their genuine products on the blockchain to stop counterfeit goods from being sold under their brand.
- Second-hand use cases: When a product changes hands, the current owner can transfer ownership, and buyers can check if the product is real and belongs to the person selling it using a PID.

The project is focused only on digital verification.

1.5 Methodology

The development of the Blockchain-Based Product Authenticity Verification System follows a structured and modular approach:

- Requirement Analysis: We began by investigating the issues that both the brands and the customers are facing issues with counterfeit products. We identified that both new and second-hand markets had particular use cases.

- System Design: The design of the system encompasses the front-end interface, the back-end API, and smart contracts. Solidity language is used to implement blockchain activities.
- Smart Contract development: Employing Solidity allowed us to develop smart contracts to regulate the process of product registration, transfer of ownership, and verification activities. Such contracts make the data secure, unalterable throughout the network.
- Frontend Development: Our frontend development required an intuitive interface that sellers and buyers can interact with without any hitches, thus we utilized React and Tailwind CSS.
- Backend and API Integration: The Node.js and the Express.js were utilized in creating the APIs to work with the non-sensitive product data stored in MongoDB but to be connected with the product information that could be ascertained in the blockchain.
- Testing and Validation: We tested the API routes with the help of Postman tool. We tried the system under different scenarios and this meant that the system would work well with new product process and resale product process.
- This procedure makes the system structured, safe and can be improved later with such attributes as connection to a targetable blockchain or providing a larger range of verifications.

1.6 Tools and Technologies

The following tools and technologies are used in the development of this Blockchain-Based Product Authenticity Verification System:

Frontend

- HTML
- CSS
- Tailwind CSS (for styling)
- React (for building the user interface)
- TypeScript (for type-safe JavaScript)

Backend

- Node.js (JavaScript runtime environment)
- Express.js (web framework for Node.js)

Database

- MongoDB (NoSQL database)
- MongoDB Atlas (cloud-hosted database service)

Blockchain & Smart Contracts

- Solidity (programming language for smart contracts)

Testing and API Tools

- Postman (for API testing and development)

Chapter 2
Literature Review

Blockchain technology has become well-known as a dependable way to store records that can't be changed and aren't controlled by a single organization [1]. It particularly comes in handy in cases where transparency and trust are required, like the tracking of products in supply chains, establishment of ownership of a product and ownership of an asset. Although numerous systems of blockchain have been developed with particular applications in mind, such as ensuring the safety of food or a luxury product, relatively few of them can be fully developed to address the needs of both a manufacturer and a consumer of used goods by providing both validation of the products received by the manufacturer and the owner.

2.0.1 Existing Blockchain-Based Verification Platforms

Several platforms have shown how blockchain can be used for tracking and verifying products:

- **IBM Food Trust**: This platform helps increase transparency in the food supply chain by letting producers, suppliers, and retailers track products from where they are made all the way to the store shelf. It isn't designed for individual consumers or non-food items, but it still shows how useful traceability can be [2].
- **Everledger**: This platform focuses on the diamond industry and helps people check where a diamond comes from and who owns it. It keeps a secure and unchangeable record of a diamond's history, which is especially helpful when diamonds are sold again [3].
- **VeChain**: VeChain uses blockchain along with Internet of Things (IoT) devices to monitor every step of a product's journey, from being made to being delivered. It works well for large companies managing logistics, but it doesn't offer much support for regular people to check ownership or verify resale [4].

Beyond these widely cited platforms, additional efforts highlight complementary approaches:

- **Permissioned blockchains for provenance**: Hyperledger Fabric demonstrates consortium governance, private channels, and modular consensus suited to enterprise traceability [6].
- **Ecosystem surveys and classifications**: Systematic reviews catalog design patterns, domains, and open issues for blockchain-based applications, including provenance and logistics [12].
- **Credential frameworks**: Standards for decentralized identifiers and verifiable credentials provide portable, cryptographically verifiable claims that can support

authenticity and ownership proofs across systems [14, 15].

2.0.2 Background and Theoretical Foundations

At its core, blockchain provides append-only, tamper-evident logs maintained by a distributed set of nodes. Smart contracts implement deterministic state machines for asset registration, transfer, and verification, enabling rules-driven provenance and ownership logic without a single point of control [7]. For product authenticity, two properties are critical: (1) *immutability and auditability*, where creation and transfer events form an auditable trail; and (2) *public verifiability*, enabling independent checks across stakeholders. In supply-chain contexts, the literature links these properties to improved transparency and trust, while noting practical deployment considerations [9, 10].

2.0.3 Non-Blockchain Approaches and Their Limitations

Traditional provenance systems rely on centralized databases, enterprise resource planning (ERP), or certificate authorities. These enable fast internal queries but face limitations: (i) tamper risk and opaque governance, (ii) limited portability of proofs outside the operator's platform, and (iii) organizational data silos that complicate cross-party verification. Adoption studies emphasize that, without interoperable and verifiable records, transparency claims are hard to substantiate across supply networks [13].

2.0.4 Privacy, Security, and Data Management in Provenance Systems

A recurring challenge is how to represent ownership and product metadata without over-disclosing private information:

- **On-chain vs. off-chain partitioning**: Storing identifiers, hashes, or commitments on-chain provides integrity and ordering, while detailed records remain off-chain to preserve privacy and reduce cost [12].
- **Encryption of ownership data**: Encrypting sensitive payloads (e.g., owner identifiers) and recording only ciphertext on-chain enables public auditability of event existence and timing without revealing personal data [8].
- **Access control at the application layer**: Role-based policies govern when and how details are revealed (e.g., during a verification request), limiting exposure while maintaining verifiability [9, 10].

2.0.5 Decentralized Identity and Ownership Proofs

Emerging standards propose portable, cryptographically verifiable claims that can accompany products throughout their lifecycle. In provenance contexts, these enable selective disclosure (revealing only what is necessary), portability across marketplaces, and layered attestations by manufacturers, stores, and owners [14, 15]. Such capabilities complement on-chain event records by making user-held proofs verifiable without depending on a single platform's database.

2.0.6 System Design Patterns Relevant to This Work

Literature and industry practice suggest a set of design choices for ownership and authenticity verification systems that align with this project's architecture:

- **Unique product identifiers**: Assigning an immutable identifier at registration ensures stable lookups and disambiguation across time and vendors [9, 10].
- **Event-centric smart contracts**: Recording creation and transfer events on-chain provides a canonical timeline for verification [7].
- **Backend-managed signing for UX**: A server-side wallet can orchestrate contract interactions while still emitting publicly verifiable events on-chain, reducing friction for end users [13].
- **Encrypted on-chain payloads**: Ownership details are encrypted before submission, balancing verifiability with privacy [8].
- **Role-based flows**: Distinct permissions for sellers, buyers, and admins streamline governance and reduce misuse [9, 10].

2.0.7 Alignment with the Proposed System

The proposed system adopts these patterns in a way that directly addresses gaps identified in prior solutions:

- **Consumer-level ownership verification**: Unlike many enterprise-focused platforms, end users can search a product by its unique ID and request verification prior to purchase, reducing counterfeit risk at the point of sale [13, 12].
- **Support for second-hand markets**: Encrypted ownership data and publicly accessible verification endpoints enable trustable resale without exposing sensitive details [8].
- **Clear role separation**: Sellers register products; buyers initiate verification and transactions; administrators oversee system integrity—consistent with governance structures observed in the literature [9, 10].
- **Hybrid storage for privacy and performance**: Sensitive data is encrypted when recorded on-chain, with application-layer controls for selective disclosure during verification workflows [8, 12].

2.1 Comparative Analysis

Table 2.1: Comparison between Existing Solutions and the Proposed System

Feature	**IBM Food Trust**	**Everledger**	**VeChain**	**Proposed System**
Primary Focus	Supply chain traceability for food	Diamond provenance and anti-counterfeit	Product lifecycle tracking for enterprises	Ownership & authenticity of both first-hand and second-hand goods
Consumer Ownership Verification	Not available	Yes (for diamonds)	Minimal	Full consumer-level ownership verification
Support for Second-hand Market	No	Limited to luxury resale	Not supported	Full second-hand product verification
Manufacturer-to-Buyer Connection	No direct connection	Not available	Enterprise-focused	Connects manufacturers to both initial and future owners
Proof of Ownership	Supply chain records only	Encrypted ledger entries	Lacks direct ownership proof	Uses encrypted Product ID (PID) for verifiable ownership

2.1.1 Trade-offs and Practical Considerations

While existing platforms demonstrate strong provenance capabilities, they often optimize for enterprise logistics rather than consumer verification at resale. In contrast, the proposed system emphasizes: (i) public verifiability of current ownership state for pre-purchase checks, (ii) privacy-preserving disclosures using encryption and role-based access, and (iii) operational simplicity via backend-managed interactions with auditable on-chain events [13, 8, 12].

2.1.2 Gaps in the Literature and Research Opportunities

The literature leaves several open areas relevant to consumer-centric authenticity verification: usability at scale and key handling without centralization; lifecycle completeness across repairs, recalls, and multi-party transfers; interoperability via standardized identifiers and claims; and aligning incentives across actors. The proposed approach contributes toward these gaps by combining encrypted on-chain records, public verification endpoints, and role-aware workflows aimed at both primary and secondary markets [9, 10, 14, 15].

Chapter 3
Requirement Specifications

3.1 Existing System

Currently, the means of verifying product ownership mostly relies on a central database, paper receipts, or reseller third-party websites. Such approaches have a number of problems:

- No clear proof of ownership.
- Documents can be easily faked.
- It's hard to track ownership changes for second-hand products securely.
- Fake products can be sold under well-known brand names.

3.1.1 Limitations or Drawbacks of the Existing System

- **No permanent records of ownership:** Data in traditional systems can be changed.
- **No way to track the product's life cycle:** Existing systems don't allow tracking from the manufacturer to the current owner.
- **Not good for second-hand transfers:** Buyers often rely on word-of-mouth or screenshots to prove a product is real.
- **No way to check if a product is genuine:** Users can't confirm if the product was made by the brand.
- **Easy to counterfeit:** Brands face problems when their names are used to sell fake products.

3.2 Proposed System

Our blockchain-based system removes the weaknesses of traditional systems by using decentralized smart contracts to:

- Let manufacturers register real products with a secure Product ID (PID).
- Make product transfers from one owner to another traceable and unchangeable.
- Give consumers a way to check if a product is genuine at any point in its life.

3.2.1 How the Proposed System Solves Existing Limitations

- **Immutable Records:** All transactions are stored on the blockchain and can't be changed.
- **Secure Transfers:** Every ownership change is checked

- **PID Verification:** Each product gets a unique and unchangeable PID.
- **Brand Protection:** System should implement a way to strictly protect brands from counterfeit products before letting any store to register, system must verify them.

3.3 System Requirements

3.3.1 Functional Requirements

Table 3.1: Functional Requirements of the System

Category	**Requirement**
Landing Page	The application must provide a landing page accessible to all visitors.
User Roles	The system must support three roles: **ADMIN, STORE, USER**.
User Roles	Admin accounts will be pre-seeded; they cannot sign up but can log in using seeded credentials.
User Roles	Admin must have a dedicated pages to manage users, stores, and products.
User Roles	Users and Stores can create accounts and will have separate pages based on their roles.
Admin Features	Admin can review, approve, or reject store creation requests.
Admin Features	Admin can manage stores, users, and products.
Store Features	A store can only register products after approval by an Admin.
Store Features	When a store registers a product, it becomes the product's initial owner, and ownership details are uploaded to the blockchain.
Product Management	Each registered product must be assigned a unique product code.
Product Management	Users can search products using the unique product code.
Product Management	Critical ownership data must be stored on the blockchain, while non-critical data is stored in a conventional database (hybrid architecture).
Ownership Data Security	Ownership information must be encrypted before being stored on the blockchain.
Ownership Data Security	The system must handle encryption and decryption internally, abstracting complexity from end users.
Ownership Privacy & Transfer	Users cannot view ownership details by default. They must request the owner to reveal information.

Ownership Privacy & Transfer	Owners can approve or deny reveal requests. If approved, the user can view ownership details.
Ownership Privacy & Transfer	Users can submit purchase requests for products. Owners can accept or reject these requests.
Ownership Privacy & Transfer	Once a purchase request is accepted, ownership is automatically transferred to the buyer.
Authentication & Authorization	The system must enforce role-based authentication and authorization to restrict access according to user roles.

3.3.2 Non-Functional Requirements

Table 3.2: Non-Functional Requirements of the System

Category	**Requirement**
Security	JWT-based authentication for protected routes.
Security	Role-based authorization for admin/store/user scopes.
Security	Secure password hashing using bcrypt with salt.
Cross-Browser Compatibility	The application must function correctly across major browsers (e.g., Chrome, Firefox, Edge, Safari).
Mobile Responsiveness	The user interface must be responsive across mobile, tablet, and desktop devices.
UI Theme Support	The application must provide support for both **Dark Mode** and **Light Mode**.

3.4 Use Cases

Table 3.3: Use Case: Visit Landing Page

Use Case ID	UC1
Use Case Name	Visit Landing Page
Actor(s)	Public (Visitor)
Trigger	Visitor opens the application URL.
Pre-Conditions	None.
Post-Conditions	Landing page is displayed with sign up / login and product search by code.
Primary Data	Inputs: optional productCode. Outputs: public content.
Priority	High
Basic Flow	Visitor navigates to the site. System renders landing page with public info and search.
Alternative Flow	N/A.

In Figure 3.1, the Visitor interacts with the Product Ownership System by accessing the application and being directed to the landing page. This use case diagram demonstrates that a visitor can reach the system's entry point without authentication, where general information about the platform and publicly available features such as a basic product search or overview content can be accessed.

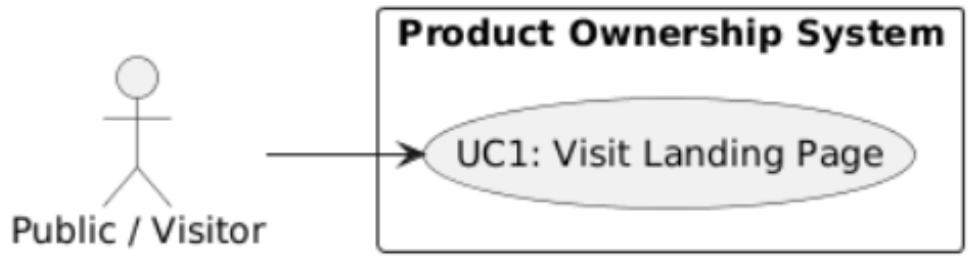

Figure 3.1: Use Case Diagram: UC1 — Visit Landing Page

Table 3.4: Use Case: Sign Up (User/Store)

Use Case ID	UC2
Use Case Name	Sign Up (User/Store)
Actor(s)	Visitor (User/Store)
Trigger	Visitor chooses to create an account.
Pre-Conditions	Visitor not authenticated.
Post-Conditions	Account created (role USER or STORE). STORE remains unverified until admin approval.
Primary Data	Inputs: email, password, identity fields. Outputs: user/store account.
Priority	High
Basic Flow	1. Visitor enters email, password, identity fields. 2. System validates; hashes password (bcrypt+salt) and creates the account.
Alternative Flow	Duplicate email or weak password: show error and allow retry.

In Figure 3.2, the **Visitor**, **User**, and **Store** actors interact with the **Product Ownership System** to perform the sign-up process. This use case diagram illustrates that an unauthenticated visitor can initiate registration and, depending on the selected role, create either a regular user account or a store account, thereby gaining access to the system with the appropriate permissions and functionality for that role.

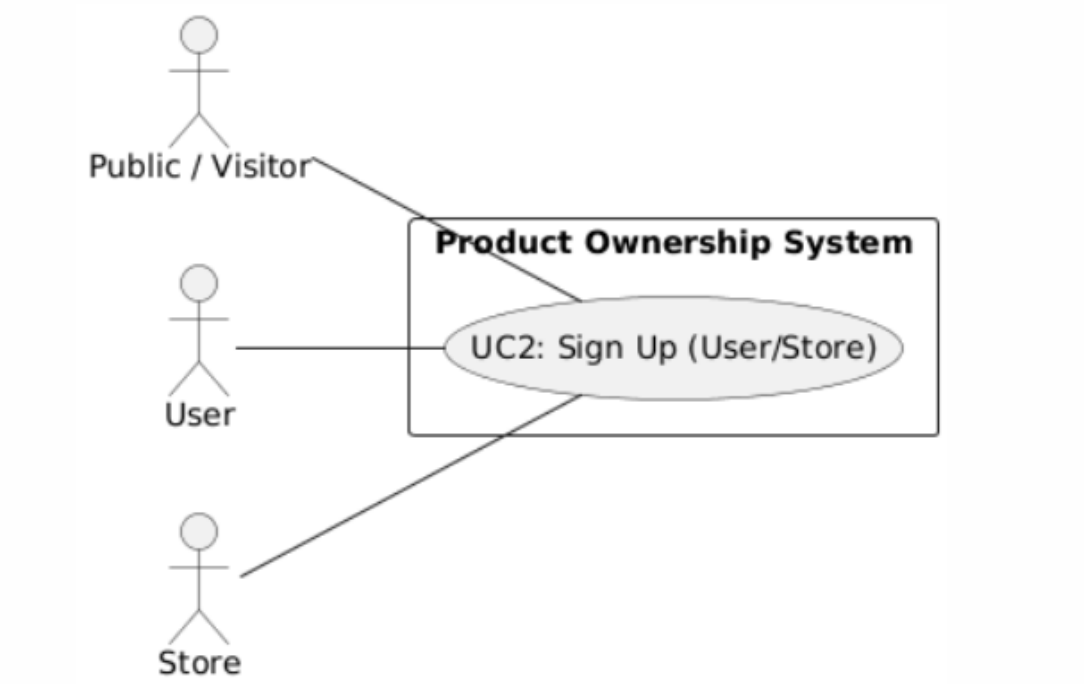

Figure 3.2: Use Case Diagram: UC2 — Sign Up (User/Store)

Table 3.5: Use Case: Login (Role-based)

Use Case ID	UC3
Use Case Name	Login (Role-based)
Actor(s)	Admin, User, Store
Trigger	Actor submits credentials.
Pre-Conditions	Account exists (Admin is pre-seeded).
Post-Conditions	JWT issued, session established; redirected to role dashboard.
Primary Data	Inputs: email, password. Outputs: JWT token, session.
Priority	High
Basic Flow	1. Actor enters credentials. 2. System verifies credentials and issues JWT. 3. System redirects to the appropriate role dashboard.
Alternative Flow	Invalid credentials: show error; allow retry.

In Figure 3.3, the **Admin**, **User**, and **Store** actors interact with the **Product Ownership System** to perform role-based login. This use case diagram illustrates that each actor authenticates using their credentials and, upon successful login, is granted access to the system with permissions and interface options appropriate to their specific role.

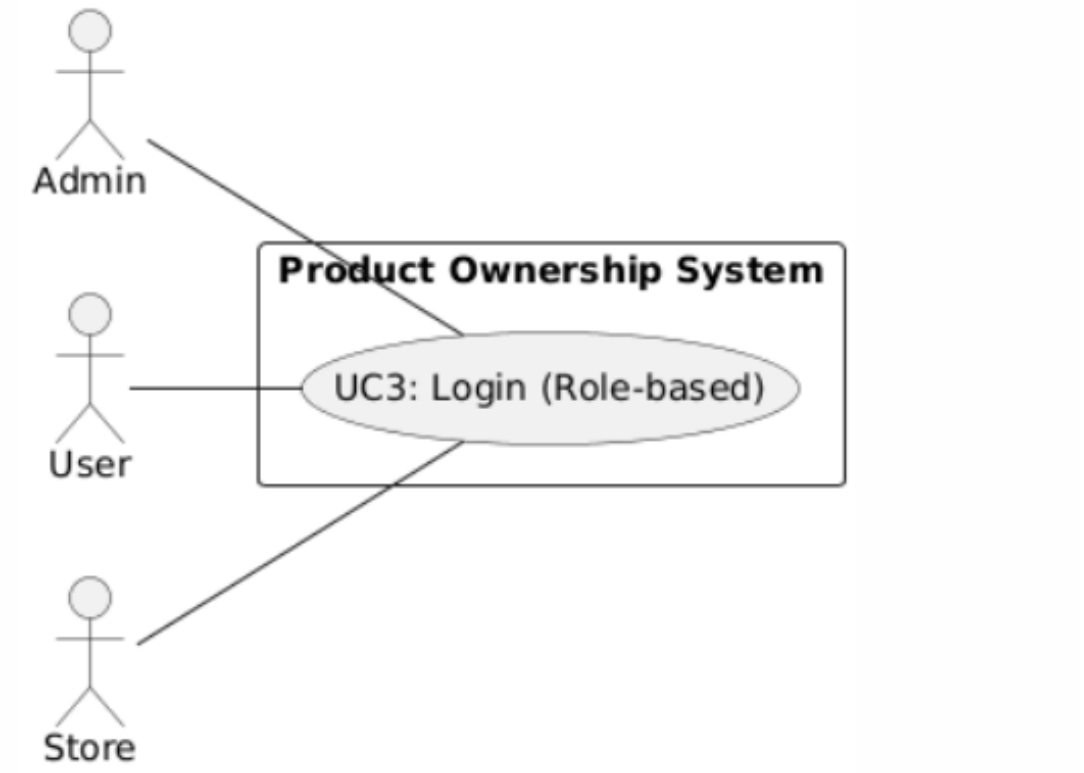

Figure 3.3: Use Case Diagram: UC3 — Login (Role-based)

Table 3.6: Use Case: View Role Dashboard

Use Case ID	UC4
Use Case Name	View Role Dashboard
Actor(s)	Admin, User, Store
Trigger	Authenticated actor opens dashboard.
Pre-Conditions	Valid session (JWT).
Post-Conditions	Role-specific capabilities are visible.
Primary Data	Inputs: actor role. Outputs: stats, lists, actions.
Priority	Medium
Basic Flow	1. Actor opens dashboard. 2. System renders Admin (manage users/stores/products), Store (manage products/requests), or User (owned/registered products, requests) views.
Alternative Flow	Insufficient permissions: deny access.

In Figure 3.4, the **Admin**, **User**, and **Store** actors interact with the **Product Ownership System** to view their respective role-specific dashboards. This use case diagram illustrates that, after logging in, each actor is presented with a tailored dashboard that exposes the features, controls, and information relevant to their assigned role.

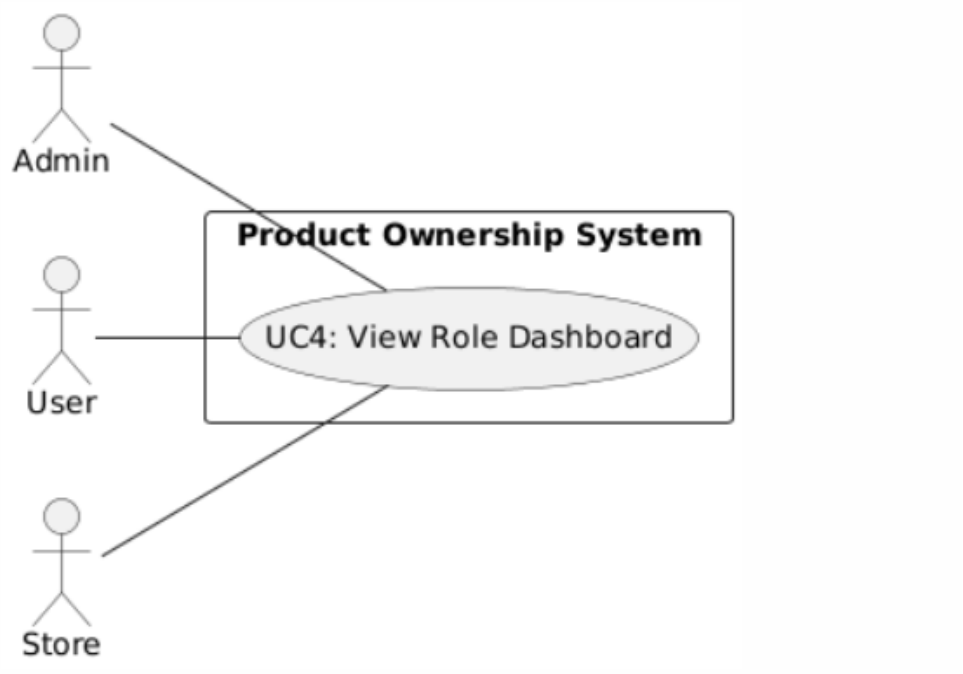

Figure 3.4: Use Case Diagram: UC4 — View Role Dashboard

Table 3.7: Use Case: Verify/Reject Store

Use Case ID	UC5
Use Case Name	Verify/Reject Store
Actor(s)	Admin
Trigger	Admin reviews pending store records.
Pre-Conditions	Pending store(s) exist.
Post-Conditions	Store marked as verified or rejected, timestamps recorded.
Primary Data	Inputs: decision. Outputs: updated store status.
Priority	High
Basic Flow	1. Admin opens pending stores. 2. Admin approves or rejects the store.
Alternative Flow	Insufficient information: request more details.

In Figure 3.5, the **Admin** interacts with the **Product Ownership System** to verify or reject store accounts. This use case diagram illustrates that the admin reviews pending store registration requests and decides whether to approve them for activation or reject them, thereby controlling which stores are allowed to operate within the system.

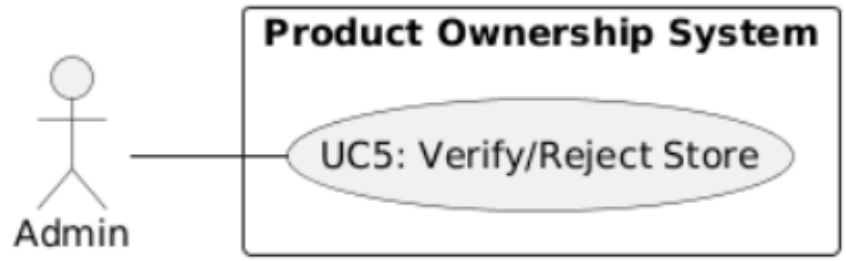

Figure 3.5: Use Case Diagram: UC5 — Verify/Reject Store

Table 3.8: Use Case: Register Product

Use Case ID	UC6
Use Case Name	Register Product
Actor(s)	Store (Verified)
Trigger	Store submits product details.
Pre-Conditions	Store is verified,actor is store member/owner.
Post-Conditions	Product created with unique `productCode`; initial encrypted ownership anchored on-chain.
Primary Data	Inputs: product details. Outputs: product record, productCode, on-chain reference.
Priority	High
Basic Flow	1. Actor enters product details. 2. System generates unique `productCode`. 3. System creates initial ownership (store as owner), encrypts ownership data, writes to blockchain; non-critical data saved to DB.
Alternative Flow	Blockchain write fails: rollback DB and report error.

In Figure 3.6, the **Store** actor interacts with the **Product Ownership System** to register a product. This use case diagram illustrates that only a **verified store** can perform product registration, during which the system records the product details and encrypts the ownership information before writing it to the blockchain.

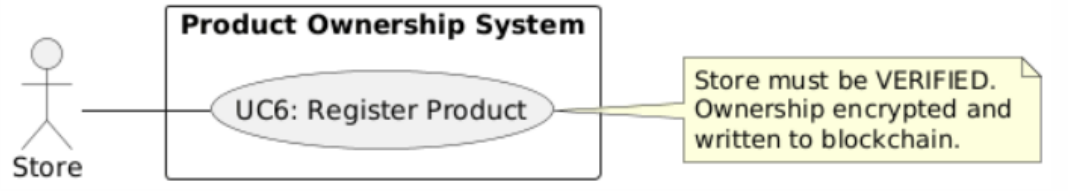

Figure 3.6: Use Case Diagram: UC6 — Register Product

Table 3.9: Use Case: Search Product by Code

Use Case ID	UC7
Use Case Name	Search Product by Code
Actor(s)	Public, User
Trigger	Actor enters `productCode`.
Pre-Conditions	Product exists.
Post-Conditions	Product public details are displayed (owner info hidden by default).
Primary Data	Inputs: `productCode`. Outputs: product details.
Priority	High
Basic Flow	1. Actor inputs `productCode`. 2. System retrieves and displays product details.
Alternative Flow	Code not found: show not-found message.

In Figure 3.7, the **Visitor** and **User** actors interact with the **Product Ownership System** to search for a product using its unique code. This use case diagram illustrates that both unauthenticated visitors and logged-in users can query the system with a product code to retrieve basic product and ownership-related status information.

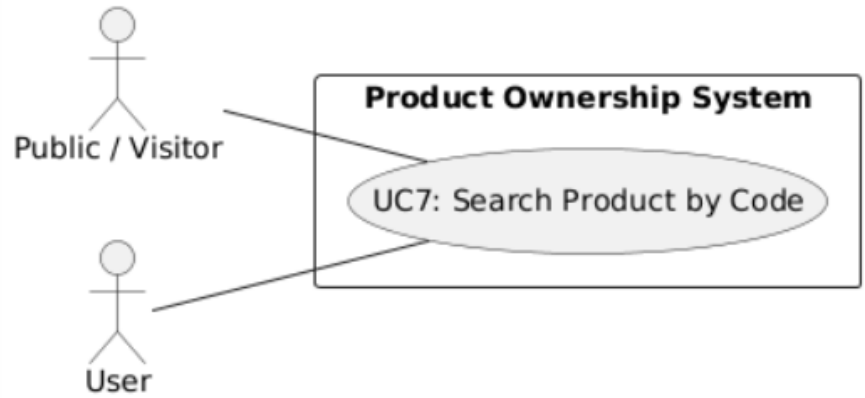

Figure 3.7: Use Case Diagram: UC7 — Search Product by Code

Table 3.10: Use Case: View Product & Conditional Ownership

Use Case ID	UC8
Use Case Name	View Product & Conditional Ownership
Actor(s)	Public, User
Trigger	Actor opens a product page.
Pre-Conditions	Product exists,ownership is stored on-chain.
Post-Conditions	Owner info is shown only if reveal permission exists for the user.
Primary Data	Inputs: product ID/code. Outputs: product public data, conditional owner details.
Priority	Medium
Basic Flow	1. System shows public product info. 2. If reveal permission exists, system decrypts and displays owner details.
Alternative Flow	No permission: owner info hidden; suggest requesting reveal.

In Figure 3.8, the **Public/Visitor** and **User** actors interact with the **Product Ownership System** to view product details along with ownership information. This use case diagram illustrates that while product data is visible to everyone, detailed owner information is displayed only when a valid reveal permission exists for the requesting party.

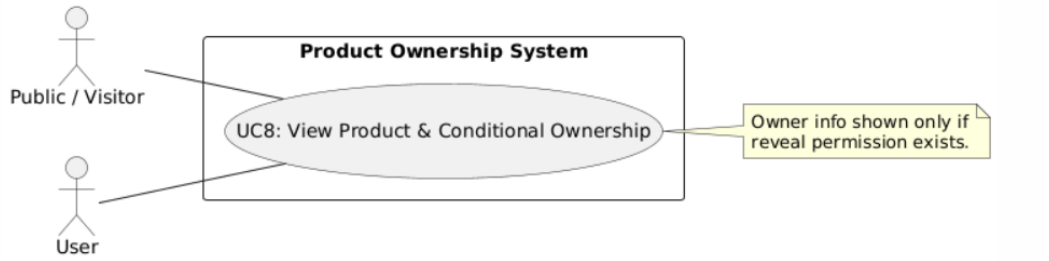

Figure 3.8: Use Case Diagram: UC8 — View Product & Conditional Ownership

Table 3.11: Use Case: Request Owner-Info Reveal

Use Case ID	UC9
Use Case Name	Request Owner-Info Reveal
Actor(s)	User (Requester)
Trigger	User requests to view current owner details.
Pre-Conditions	Authenticated,user is not the current owner.
Post-Conditions	Reveal request recorded with status *pending*; owner notified.
Primary Data	Inputs: product reference. Outputs: reveal request record.
Priority	Medium
Basic Flow	1. User submits reveal request. 2. System stores request as *pending* and notifies the owner.
Alternative Flow	Duplicate pending request: inform user.

In Figure 3.9, the **User** interacts with the **Product Ownership System** to request that the current owner's information be revealed. This use case diagram illustrates that a user can initiate an ownership-info reveal request, which the system forwards to the owner for approval before any sensitive details are disclosed.

Figure 3.9: Use Case Diagram: UC9 — Request Owner-Info Reveal

Table 3.12: Use Case: Decide Reveal Request

Use Case ID	UC10
Use Case Name	Decide Reveal Request
Actor(s)	Current Owner
Trigger	Owner reviews a pending reveal request.
Pre-Conditions	A pending reveal request exists.
Post-Conditions	Request approved/denied and timestamped; permissions updated if approved.
Primary Data	Inputs: decision. Outputs: updated request, permission grant if approved.
Priority	Medium
Basic Flow	1. Owner opens pending request. 2. Owner approves or denies. 3. System timestamps and updates permissions if approved.
Alternative Flow	Request withdrawn/expired: close with reason.

In Figure 3.10, the **Current Owner** interacts with the **Product Ownership System** to decide on incoming owner-information reveal requests. This use case diagram illustrates that the current owner has the authority to either approve or reject each request, thereby controlling whether their ownership details are disclosed to the requesting user.

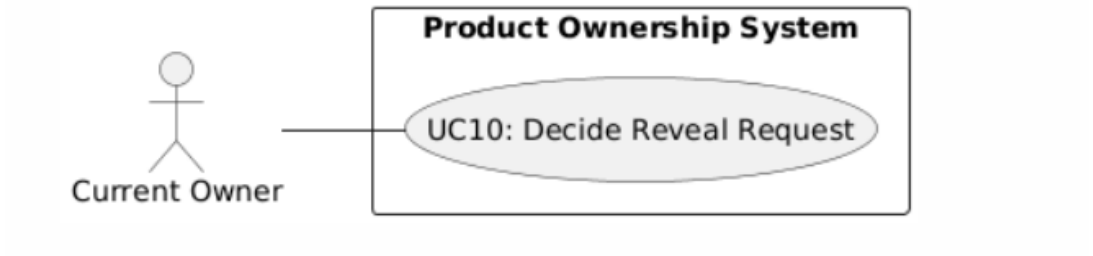

Figure 3.10: Use Case Diagram: UC10 — Decide Reveal Request

Table 3.13: Use Case: Create Purchase Request

Use Case ID	UC11
Use Case Name	Create Purchase Request
Actor(s)	User (Buyer)
Trigger	User decides to buy a product.
Pre-Conditions	Authenticated,product status allows sale (e.g., ACTIVE), user is not current owner.
Post-Conditions	Purchase request recorded with status pending.
Primary Data	Inputs: product reference. Outputs: purchase request.
Priority	High
Basic Flow	1. User submits purchase request to current owner. 2. System stores request as pending.
Alternative Flow	Product RECALLED/STOLEN/RETIRED: block and inform; duplicate pending request: notify.

In Figure 3.11, the **User** interacts with the **Product Ownership System** to create a purchase request for a product. This use case diagram illustrates that a user can only initiate such a request when the product is in an **ACTIVE** state, ensuring that purchase attempts target valid and currently transferable items.

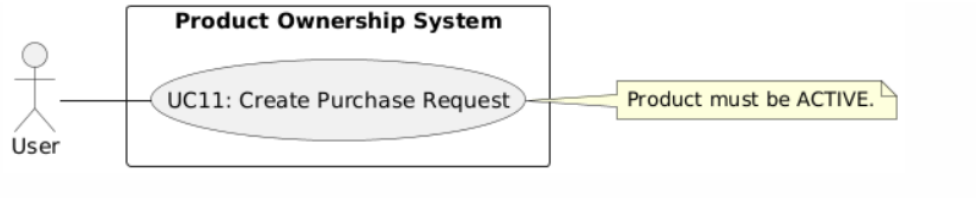

Figure 3.11: Use Case Diagram: UC11 — Create Purchase Request

Table 3.14: Use Case: Accept/Reject Purchase Request & Transfer

Use Case ID	UC12
Use Case Name	Accept/Reject Purchase Request & Transfer Ownership
Actor(s)	Current Owner, User (Buyer)
Trigger	Owner reviews a pending purchase request.
Pre-Conditions	Valid pending purchase request exists.
Post-Conditions	If accepted: ownership transferred to buyer, encrypted record written on-chain ,DB reflects new owner. If rejected: request closed as rejected.
Primary Data	Inputs: decision. Outputs: ownership records, updated product owner/status.
Priority	High
Basic Flow	1. Owner accepts the request. 2. System ends current ownership, creates new ownership for buyer, encrypts payload, writes to blockchain, updates DB.
Alternative Flow	Owner rejects: request marked rejected and requester notified. Blockchain failure: rollback and report error.

In Figure 3.12, the **Current Owner** and **Buyer (User)** interact with the **Product Ownership System** to accept or reject a purchase request and, if accepted, transfer ownership. This use case diagram illustrates that when the current owner approves a request, the system closes the existing ownership record, creates a new encrypted ownership record for the buyer, and writes this updated ownership state to the blockchain.

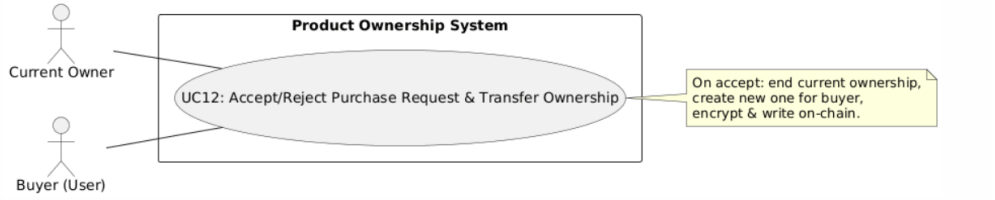

Figure 3.12: Use Case Diagram: UC12 — Accept/Reject Purchase Request & Transfer

Table 3.15: Use Case: Update Product Status (Moderation)

Use Case ID	UC13
Use Case Name	Update Product Status (Moderation)
Actor(s)	Admin
Trigger	Admin moderates product lifecycle/flags.
Pre-Conditions	Admin authenticated, product exists.
Post-Conditions	Status updated with reason,timestamps recorded.
Primary Data	Inputs: new status, reason. Outputs: updated product record.
Priority	Medium
Basic Flow	1. Admin selects product. 2. Admin sets new status with reason. 3. System records change and timestamps.
Alternative Flow	Invalid transition per policy: show error.

In Figure 3.13, the **Admin** interacts with the **Product Ownership System** to update and moderate product status. This use case diagram illustrates that the admin can change a product's state (for example, marking it active, suspended, or removed), ensuring that only compliant and valid products remain available within the system.

Figure 3.13: Use Case Diagram: UC13 — Update Product Status (Moderation)

Table 3.16: Use Case: View Ownership History

Use Case ID	UC14
Use Case Name	View Ownership History
Actor(s)	User, Admin
Trigger	Actor requests history for a product.
Pre-Conditions	Product exists.
Post-Conditions	Ownership timeline displayed,identities visible only if permission exists.
Primary Data	Inputs: product reference. Outputs: startedAt/endedAt and ownership events; conditional identities.
Priority	Medium
Basic Flow	1. System composes timeline from on-chain refs and DB. 2. System decrypts permitted parts and displays the history.
Alternative Flow	No permission for identities: redact owner identifiers.

In Figure 3.14, the **User** and **Admin** interact with the **Product Ownership System** to view a product's ownership history. This use case diagram illustrates that while the full transfer trail is visible, individual owner identities may be redacted unless explicit permission exists to reveal those details.

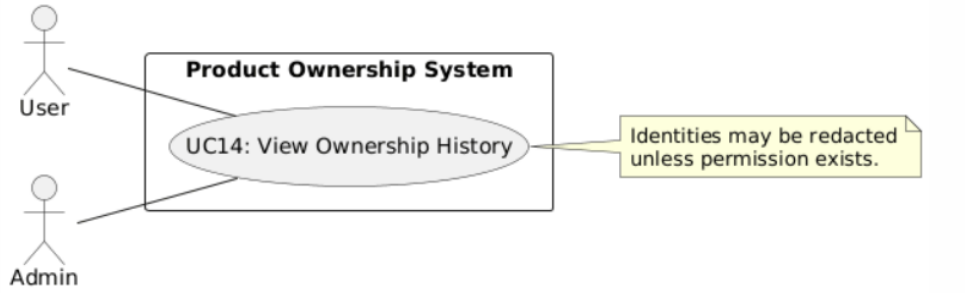

Figure 3.14: Use Case Diagram: UC14 — View Ownership History

Table 3.17: Use Case: Public Product Discovery

Use Case ID	UC15
Use Case Name	Public Product Discovery
Actor(s)	Public
Trigger	Visitor browses or searches products.
Pre-Conditions	Products exist.
Post-Conditions	Public fields displayed,private ownership info hidden.
Primary Data	Inputs: productCode or filters. Outputs: product list/details.
Priority	Low
Basic Flow	1. Visitor uses search or browsing. 2. System shows public product fields only.
Alternative Flow	No results: show empty state.

In Figure 3.15, the **Public/Visitor** interacts with the **Product Ownership System** to perform public product discovery. This use case diagram illustrates that anyone, without creating an account or logging in, can explore or look up products exposed by the system while still respecting privacy and access-control rules for sensitive ownership data.

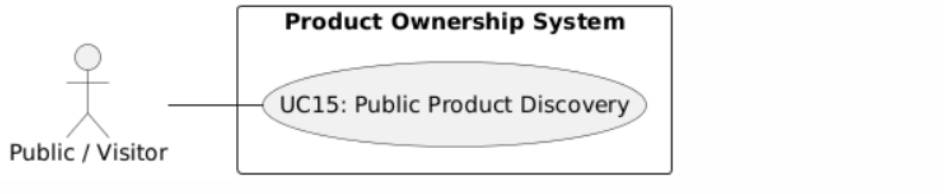

Figure 3.15: Use Case Diagram: UC15 — Public Product Discovery

Table 3.18: Use Case: Security — Crypto Abstraction

Use Case ID	UC16
Use Case Name	Security — Crypto Abstraction
Actor(s)	System
Trigger	Ownership write/read operations occur.
Pre-Conditions	Keys securely provisioned/stored,caller is authorized.
Post-Conditions	Ownership data encrypted before on-chain write, decrypted on authorized reads, keys never exposed.
Primary Data	Inputs: ownership payload. Outputs: encrypted on-chain data / decrypted view.
Priority	High (security)
Basic Flow	1. On write: system encrypts payload and commits to blockchain. 2. On authorized read: system decrypts and returns data to permitted user.
Alternative Flow	Key retrieval/integrity failure: deny operation; log security event.

In Figure 3.16, the **Product Ownership System** performs an internal security and cryptography abstraction that is hidden from end users. This use case diagram illustrates that the system automatically encrypts ownership data before writing it to the blockchain and decrypts it only for authorized reads, ensuring that cryptographic keys are never exposed to users or client applications.

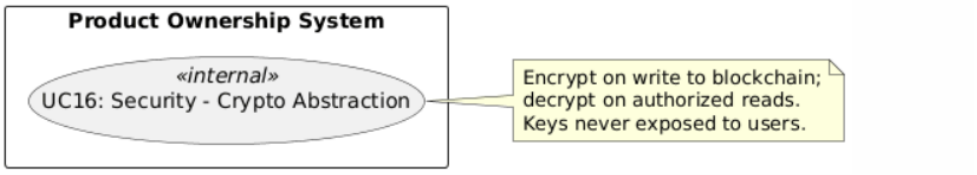

Figure 3.16: Use Case Diagram: UC16 — Security — Crypto Abstraction

Table 3.19: Use Case: Authorization Guard (Protected Endpoints)

Use Case ID	UC17
Use Case Name	Authorization Guard (Protected Endpoints)
Actor(s)	Admin, Store, User, System
Trigger	Any protected route is accessed.
Pre-Conditions	JWT present; role identified.
Post-Conditions	Access granted or denied per role policy (ADMIN/STORE/USER).
Primary Data	Inputs: JWT, route policy. Outputs: allow/deny response.
Priority	High
Basic Flow	1. System validates JWT and extracts role. 2. System checks route policy and enforces authorization.
Alternative Flow	Missing/invalid token: 401,insufficient role: 403.

In Figure 3.17, the **Admin**, **Store**, and **User** actors interact with the **Product Ownership System**, which internally applies an authorization guard on protected endpoints. This use case diagram illustrates that the system enforces role-based access control by validating JWTs on every protected request, returning an HTTP 401 status for missing or invalid tokens and an HTTP 403 status when the authenticated role lacks sufficient permissions.

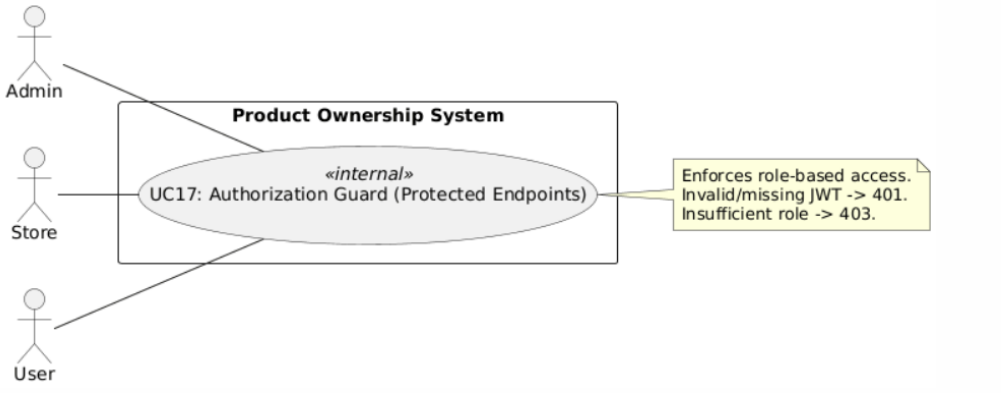

Figure 3.17: Use Case Diagram: UC17 — Authorization Guard (Protected Endpoints)

3.5 System Sequence Diagrams

In Figure 3.18, the sequence diagram illustrates how a visitor opens the landing page and searches for products. The Web UI sends requests (GET /landing and GET /products?filters) to the API, which queries the database for featured items and public fields. The API then returns the corresponding results so the Web UI can display the filtered product list to the visitor.

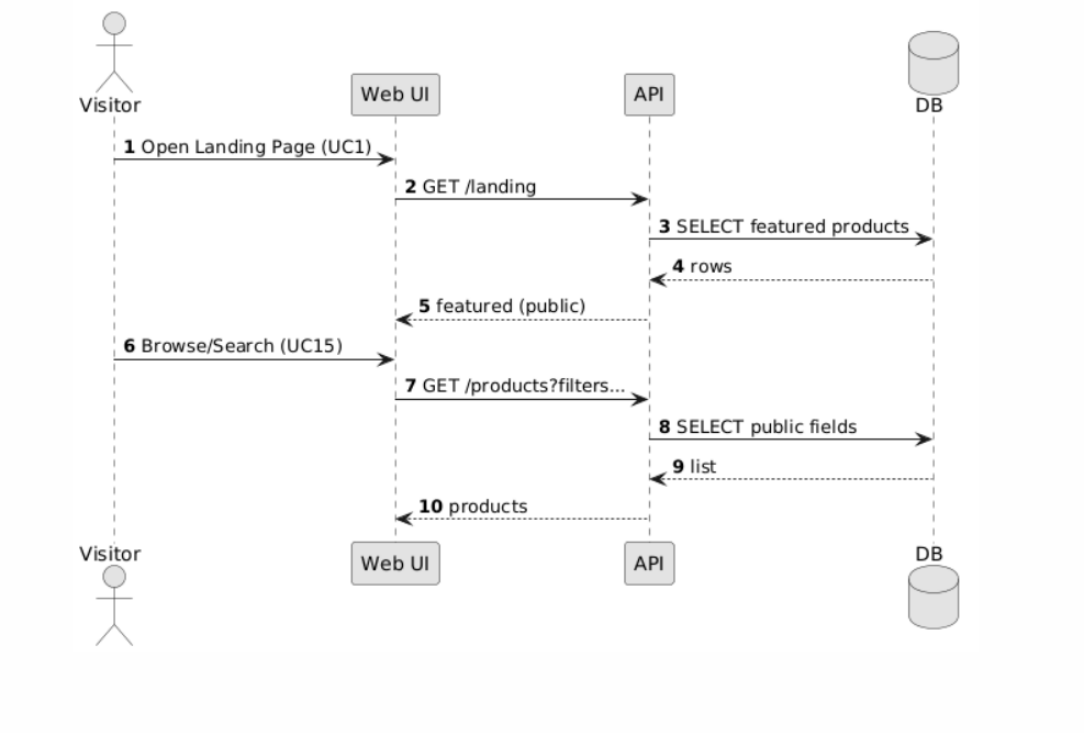

Figure 3.18: System Sequence Diagram

In Figure 3.19, the sequence diagram illustrates the registration flow for a new user. A visitor opens the sign-up page and the Web UI sends a POST /signup request with email, password, and role to the API, which delegates user creation to the Auth service and database. After a successful insert, the API returns a 201 response containing a role-encoded JWT to the Web UI, effectively authenticating the newly registered user.

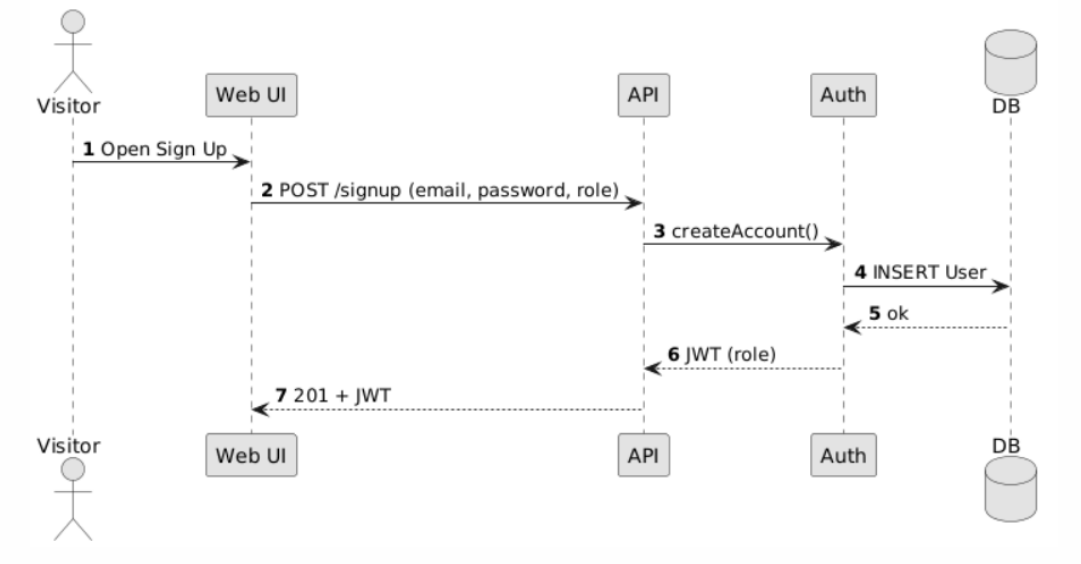

Figure 3.19: System Sequence Diagram

In Figure 3.20, the sequence diagram illustrates the authenticated login-to-dashboard flow. The user submits their credentials, the Web UI sends them to the API for verification, and upon success the API returns a JWT to the client. Using this token, the Web UI requests the dashboard, the API validates the JWT and associated role, executes role-based database queries, and returns the data needed for the dashboard to render.

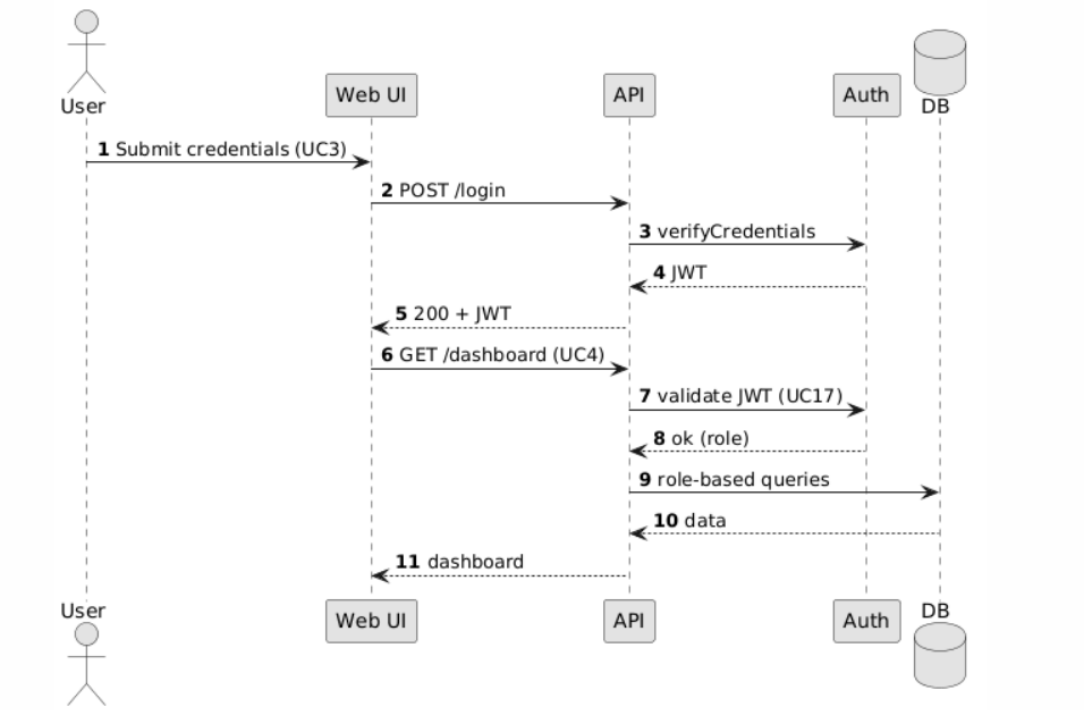

Figure 3.20: System Sequence Diagram

In Figure 3.21, the sequence diagram illustrates how the admin moderates store onboarding. The admin requests pending stores with GET /stores?status=pending, reviews the returned list, and then approves or rejects a specific store using PATCH /stores/id to set status = verified or rejected. The API updates the store record in the database accordingly and returns a 200 response on success.

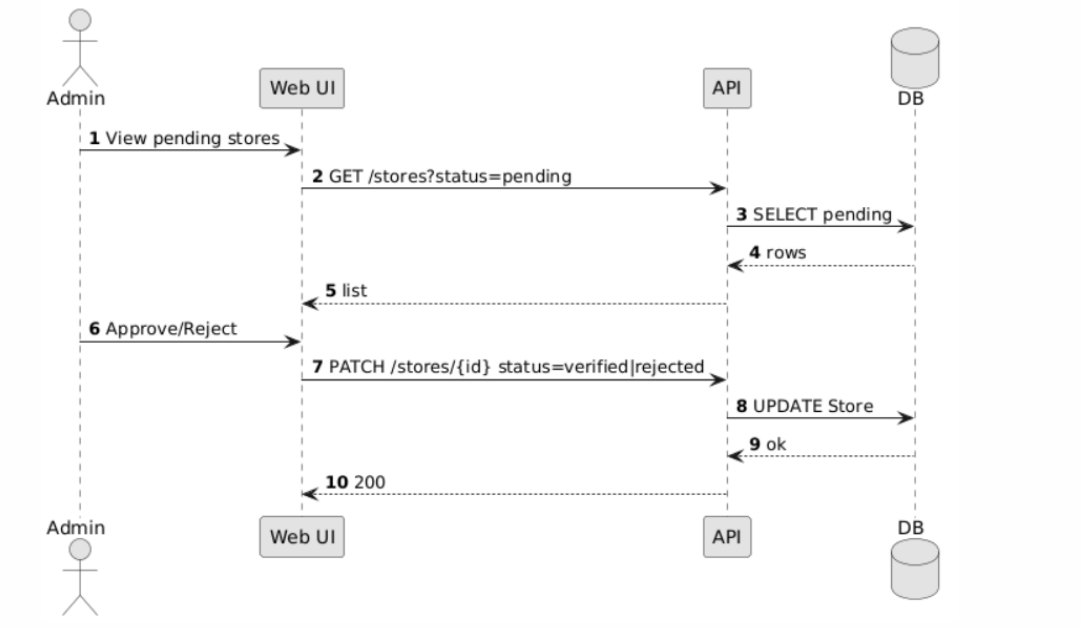

Figure 3.21: System Sequence Diagram

In Figure 3.22, the sequence diagram illustrates how a verified store registers a new product. The Web UI sends the product details to the backend, which creates a unique product identifier, encrypts the ownership payload, writes a reference transaction to the blockchain, and persists the related product record in the database. The backend then returns a confirmation, including the generated product code, to the store.

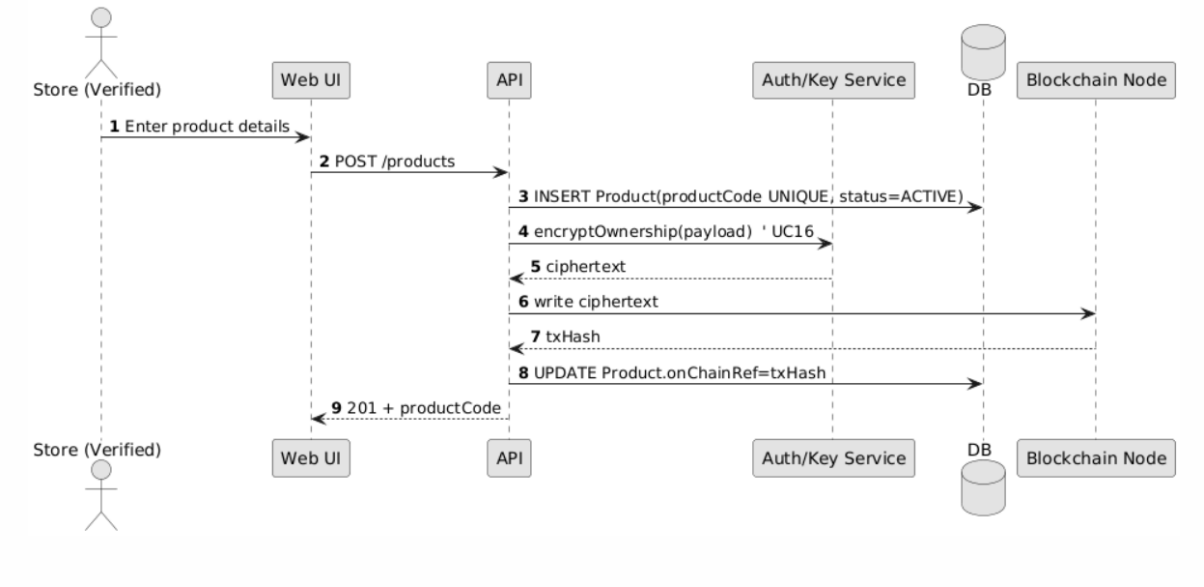

Figure 3.22: System Sequence Diagram

In Figure 3.23, the sequence diagram illustrates the search-to-reveal process. A requester looks up a product by code, the API checks existing permissions, and if authorized reads the encrypted ownership data from the blockchain, decrypts it via the key service, and returns the owner details. If the requester is not authorized, they submit a reveal request that the owner later approves or denies. When approved, a permission record is created and the owner information becomes visible to the requester.

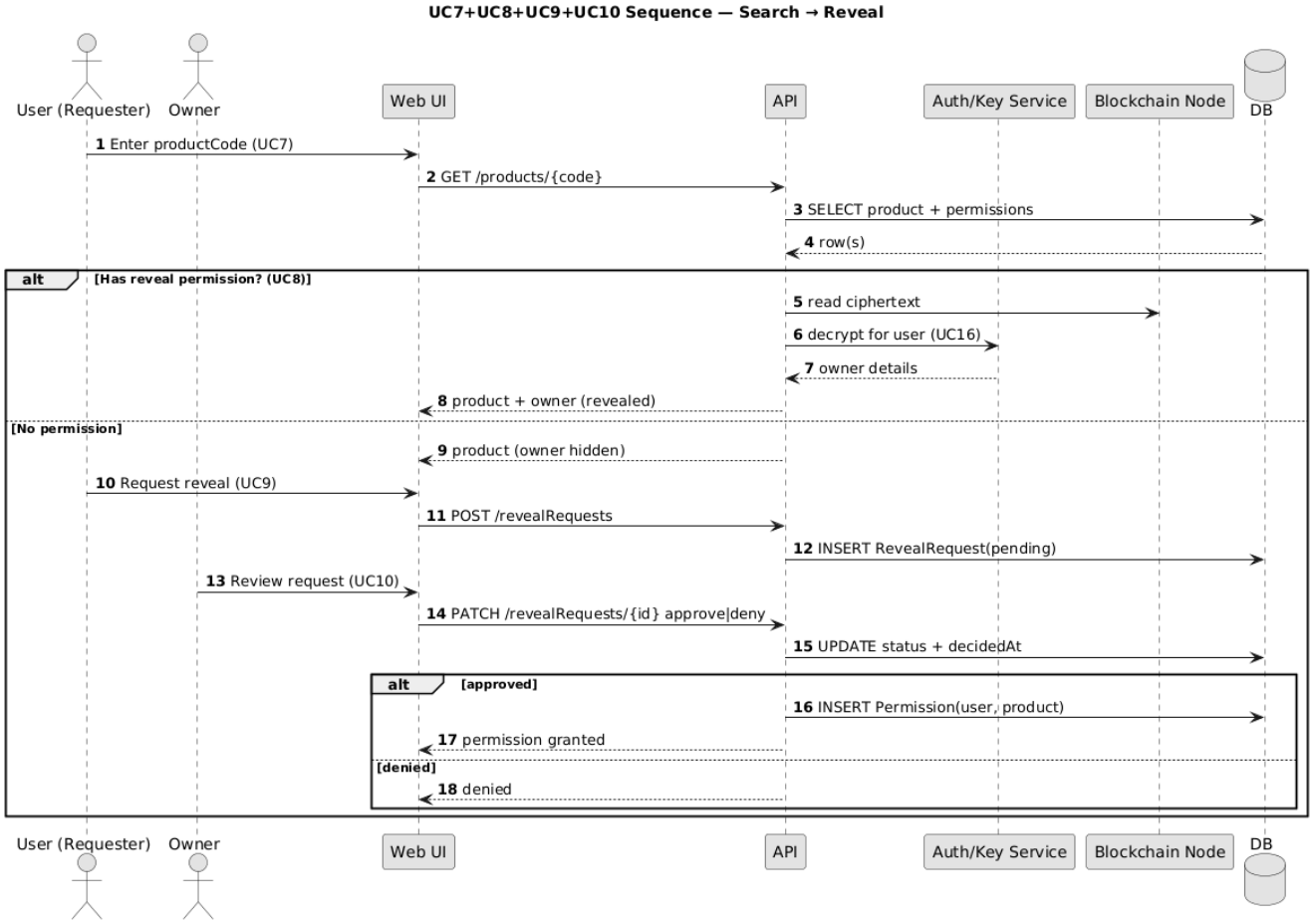

Figure 3.23: System Sequence Diagram

In Figure 3.24, the sequence diagram illustrates the purchase and ownership transfer flow. A buyer creates a purchase request that is stored with a pending status until the current owner reviews and accepts it. On acceptance, the API encrypts the new ownership record for the buyer, writes it to the blockchain, ends the previous ownership, and updates both the product's currentOwner field and the request status in the database. The Web UI then receives a success response, confirming that the ownership transfer has been completed.

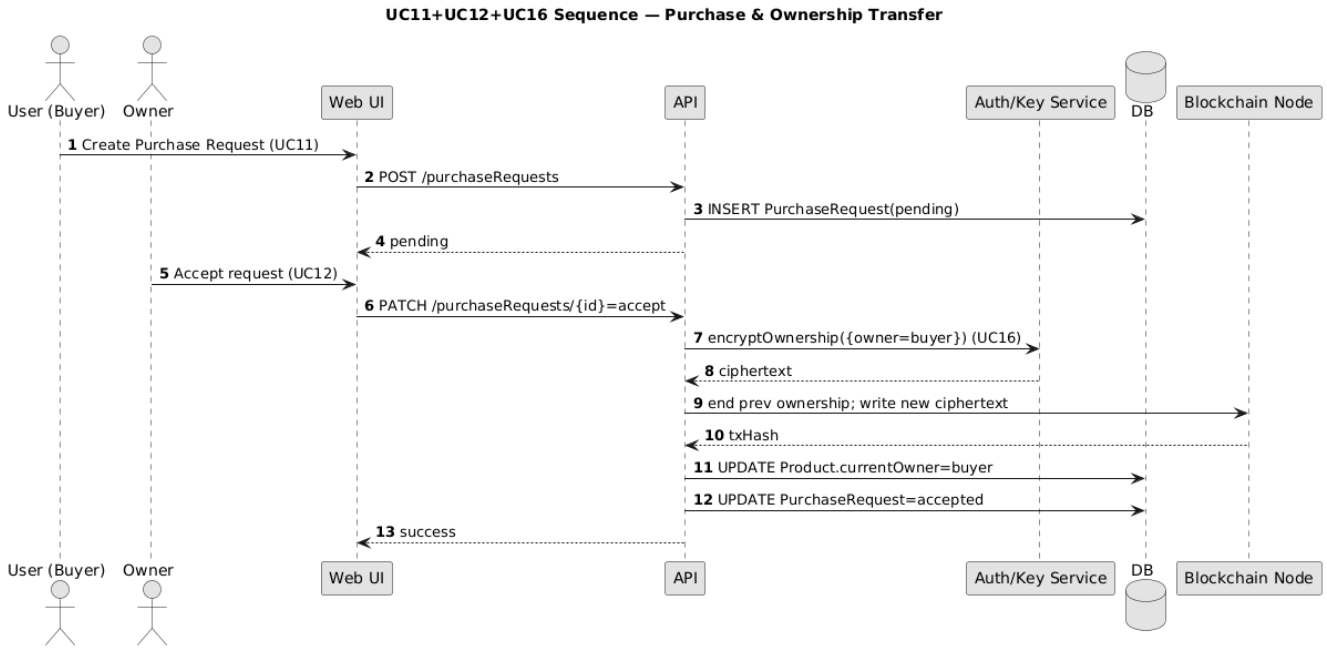

Figure 3.24: System Sequence Diagram

In Figure 3.25, the sequence diagram illustrates admin product moderation. The admin updates a product's status using PATCH /products/id with one of the allowed values (ACTIVE, RECALLED, STOLEN, or RETIRED), the API persists the new Product.status, records the action in a ModerationLog entry in the database, and returns a 200 response to the Web UI.

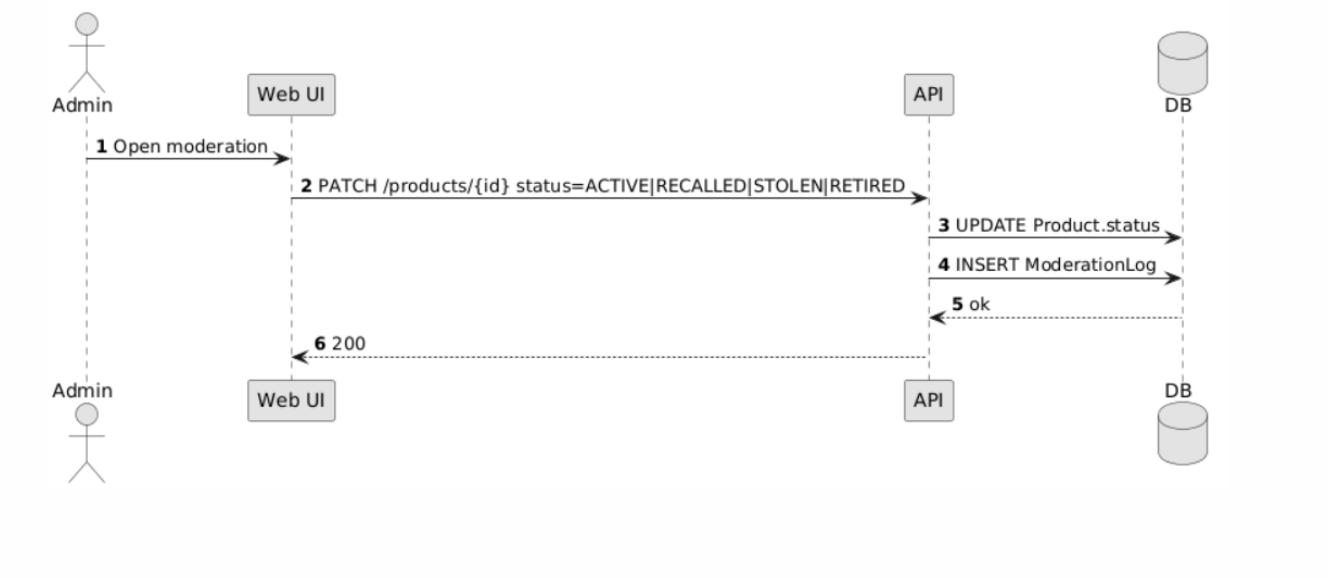

Figure 3.25: System Sequence Diagram

In Figure 3.26, the diagram illustrates how a user views a product's ownership history as a timeline of past and current owners. When the user has the necessary permission, owner identities are displayed. Otherwise, the same timeline is returned with those identities hidden.

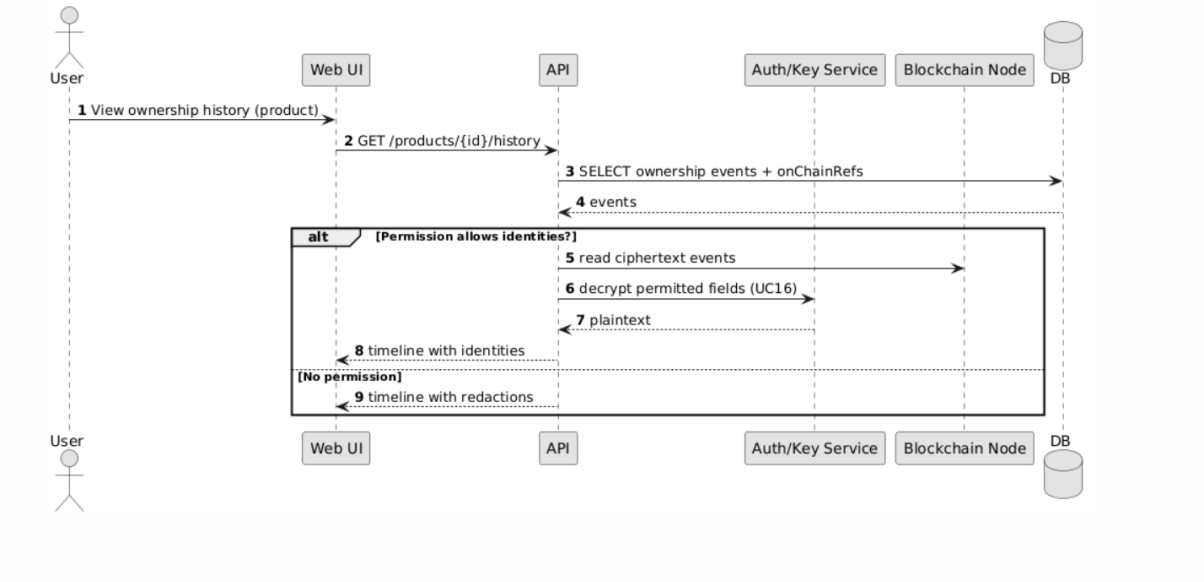

Figure 3.26: System Sequence Diagram

3.6 System Activity Diagrams

The activity diagram shows the very first interaction with the system. In Figure 3.27, the process starts when the user opens the application URL in a browser. In response, the system loads and renders the landing page, which displays public information about the application along with a search box for looking up products. Once the landing page is shown and ready for interaction, this initial activity ends.

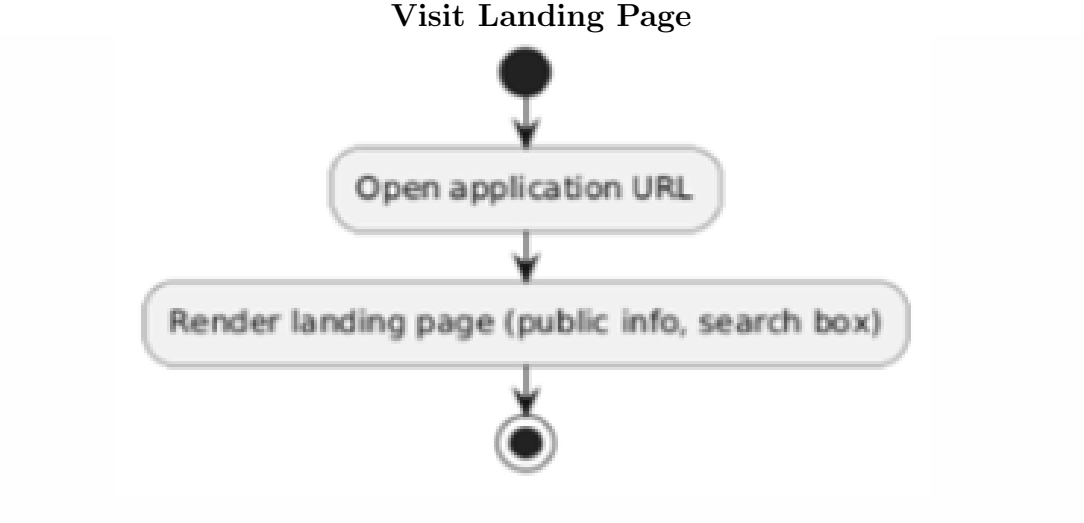

Figure 3.27: System Activity Diagram 1

In Figure 3.28, the activity diagram illustrates the sign-up flow for new users in the system. The user first opens the sign-up page and enters their email, password, and role, after which the system validates the provided input. If the data is invalid, an error message is shown and the user is allowed to retry the process; if the data is valid, the system hashes the password using bcrypt with salt and creates a new user record. When the selected role corresponds to a store, an additional store entity is created with a pending status that awaits admin verification. Once these steps are completed, the system issues a JWT and automatically logs the user in.

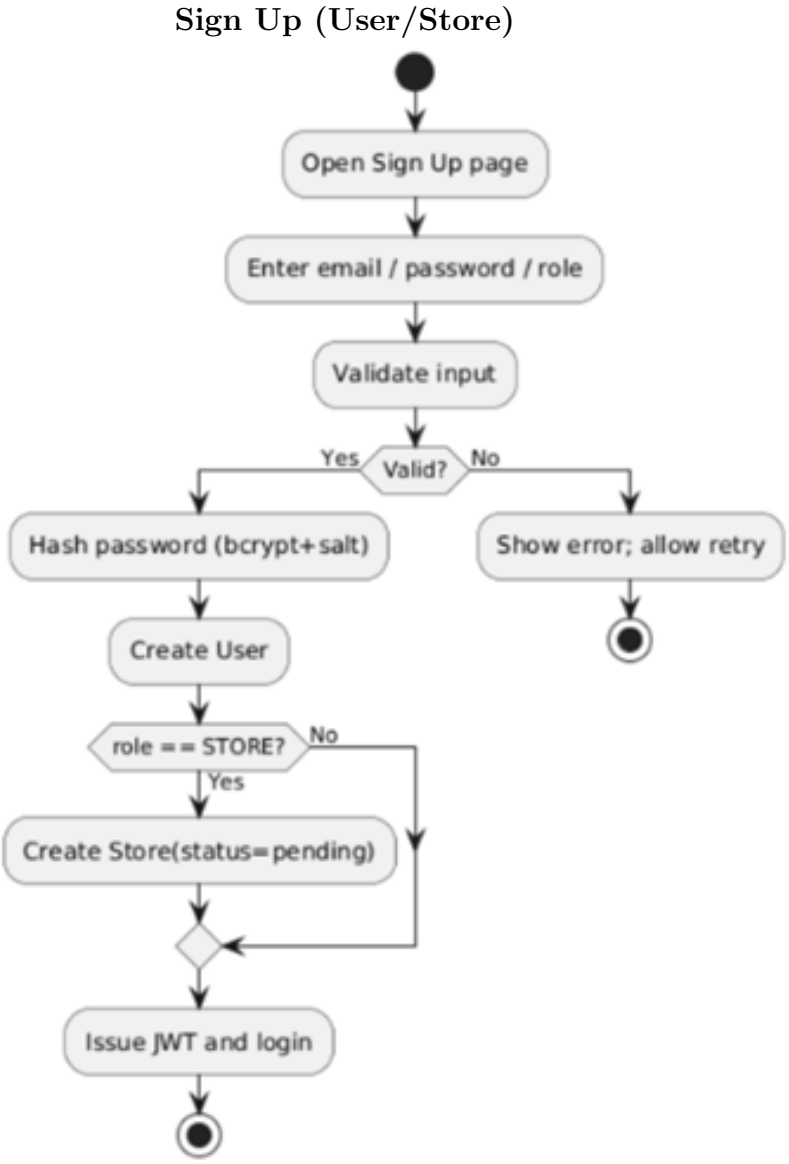

Figure 3.28: System Activity Diagram 2

In Figure 3.29, the activity diagram illustrates the login process for existing users. The user enters their credentials, which the system then verifies against stored records. If the credentials are valid, the system issues a JWT and redirects the user to their role-specific dashboard; if the credentials are invalid, an error message is displayed and the login flow terminates.

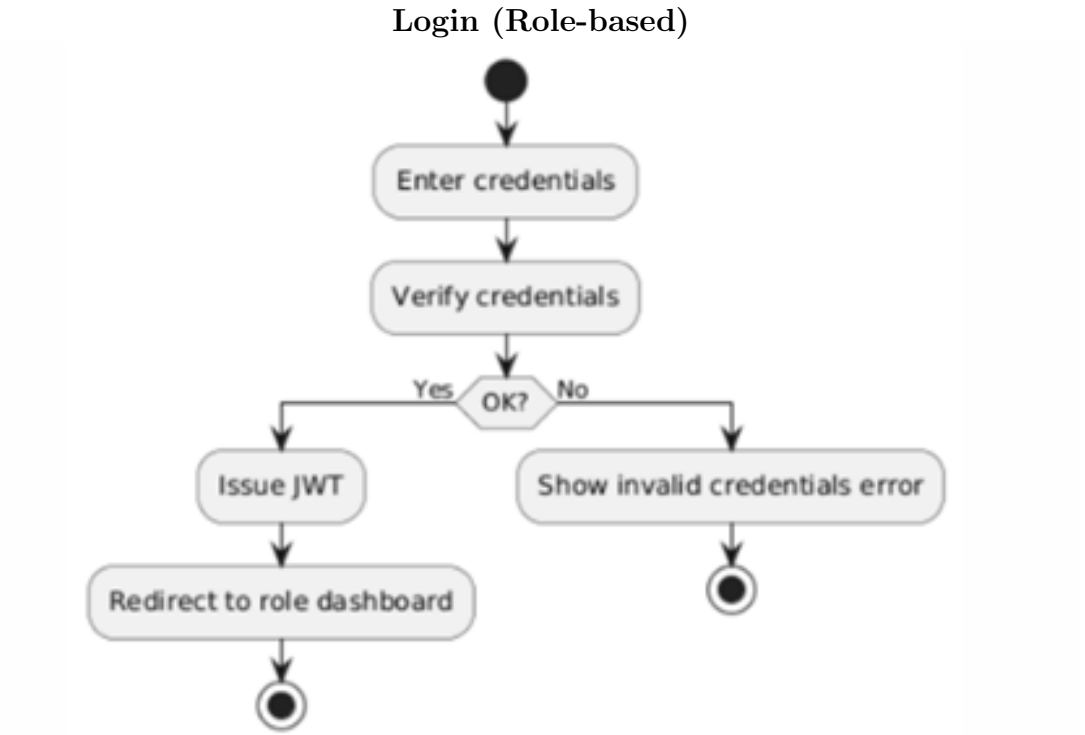

Figure 3.29: System Activity Diagram 3

In Figure 3.30, the activity diagram illustrates how access to the dashboard is controlled based on authorization and user role. When the dashboard route is opened, the system first validates the user's JWT, if the token is invalid or missing, access is denied with a 401/403 response. If the token is valid, the system detects the user's role (ADMIN, STORE, or USER) and renders the corresponding role-specific widgets and actions on the dashboard.

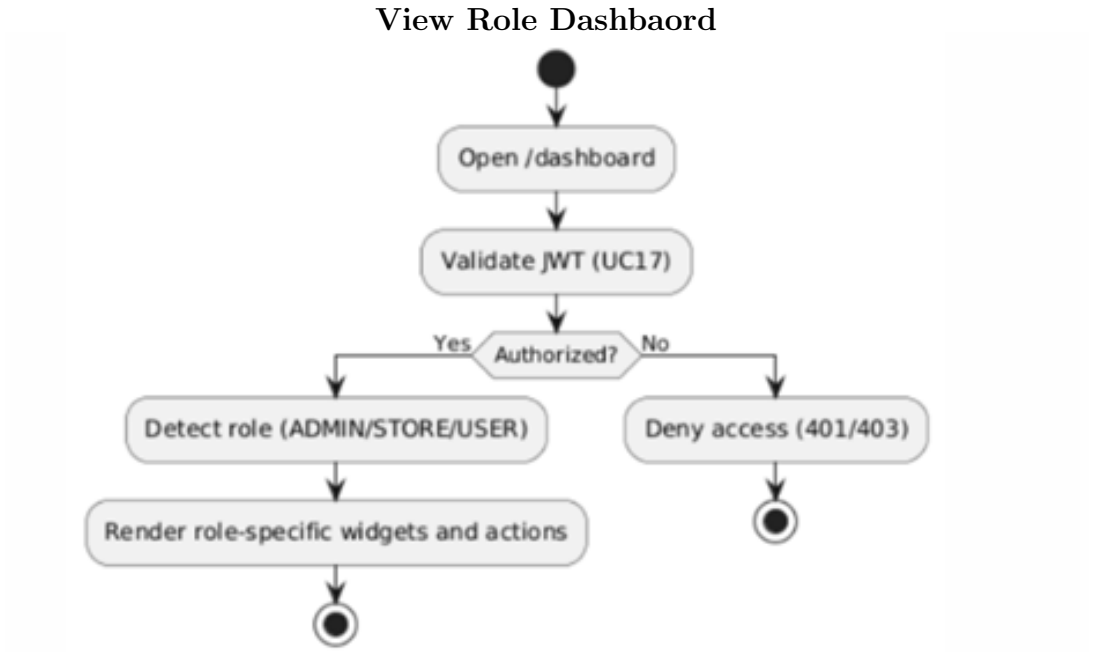

Figure 3.30: System Activity Diagram 4

In Figure 3.31, the activity diagram illustrates the admin workflow for moderating store registrations. The system first lists all pending stores, and the admin selects a specific store to review. Based on the decision, the store's status is updated to either verified or rejected, and the system then records the corresponding timestamp and auditor information before completing the process.

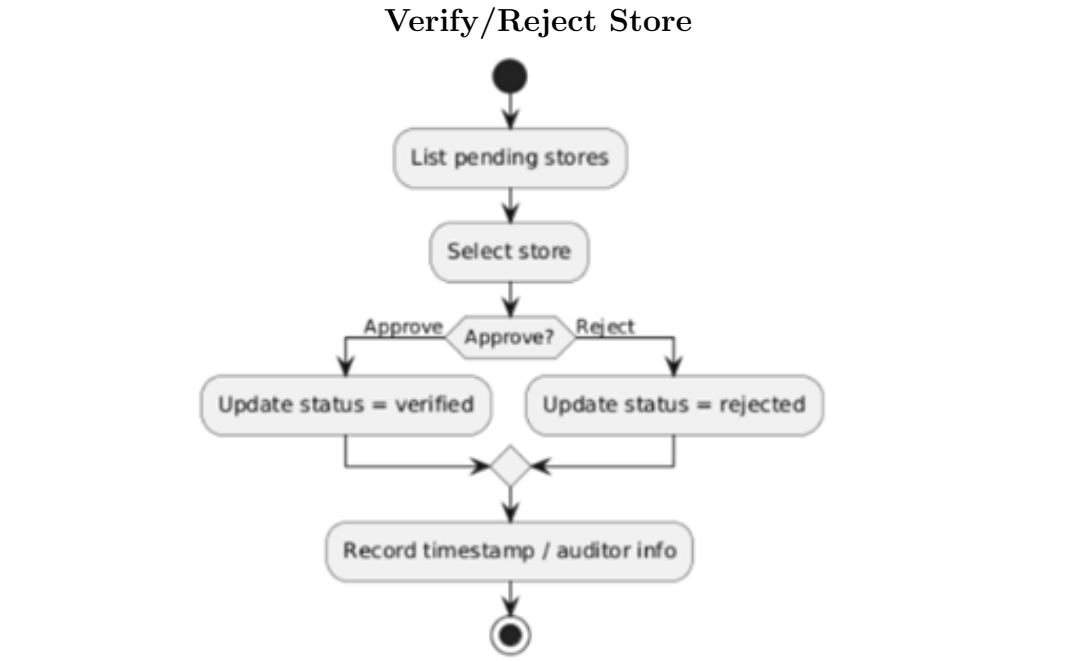

Figure 3.31: System Activity Diagram 5

In Figure 3.32, the activity diagram illustrates the process of registering a new product by a verified store. The store inputs the product details, after which the system generates a unique product code and creates the product record with an ACTIVE status. It then builds the initial ownership payload with the store as the owner, encrypts this payload, and writes the resulting ciphertext to the blockchain, capturing the transaction hash as the on-chain reference. Finally, the system stores this reference with the product record and returns the generated product code to the store.

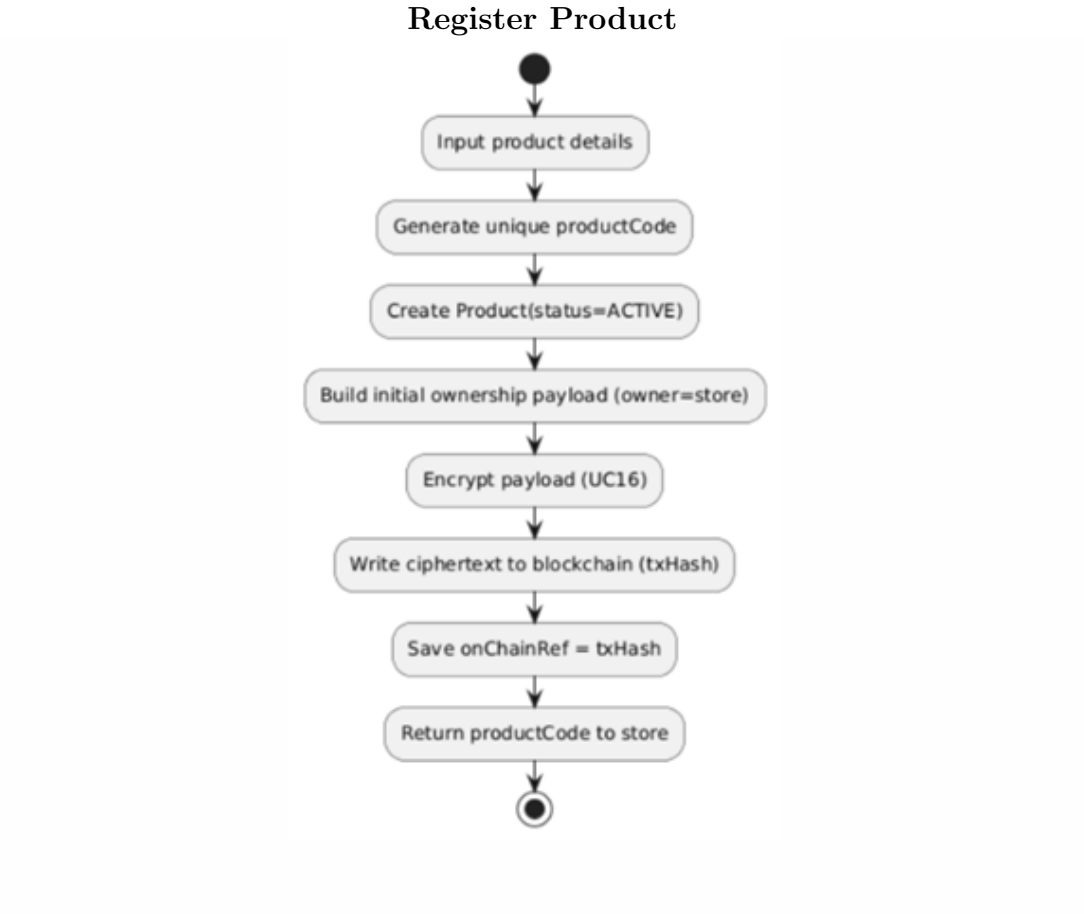

Figure 3.32: System Activity Diagram 6

In Figure 3.33, the activity diagram illustrates how a user searches for a product using its unique product code. The user enters the product code, and the system attempts to fetch the corresponding product record. If a matching product is found, the system displays the public product fields otherwise, it shows a not found message and the flow terminates.

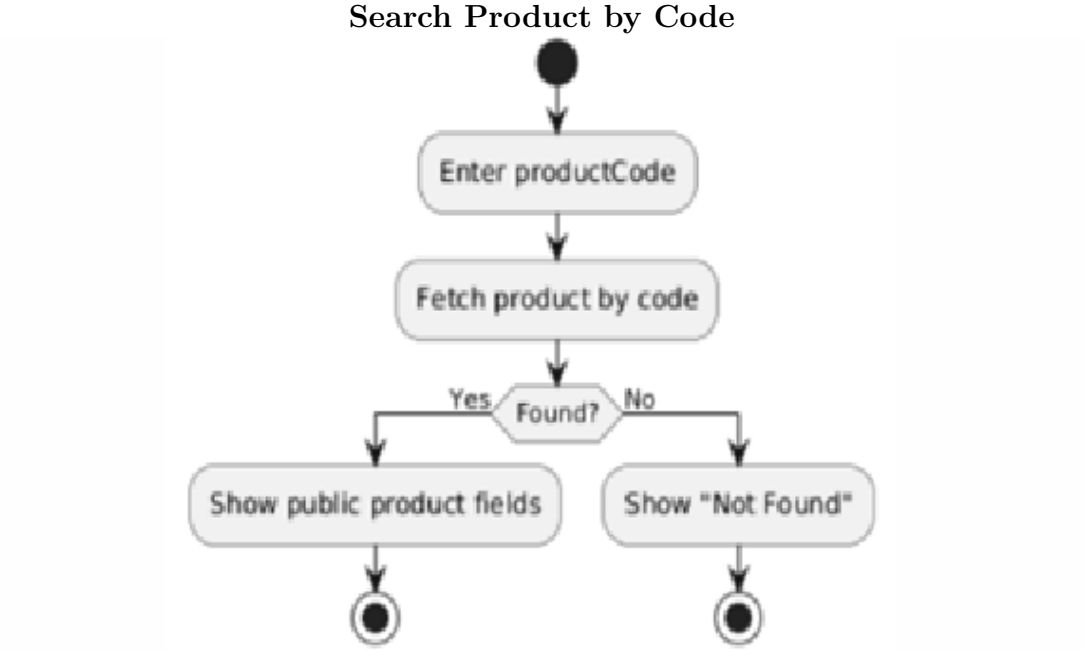

Figure 3.33: System Activity Diagram 7

In Figure 3.34, the activity diagram illustrates how the system conditionally displays product owner information based on reveal permissions. When a user opens a product page, the system loads the product data together with the current permission state and checks whether a valid reveal permission exists. If permission is granted, the system reads the encrypted ownership ciphertext from the blockchain, decrypts the authorized parts, and displays the owner details. If permission is not available, the system hides owner details and instead suggests that the user request an ownership-info reveal.

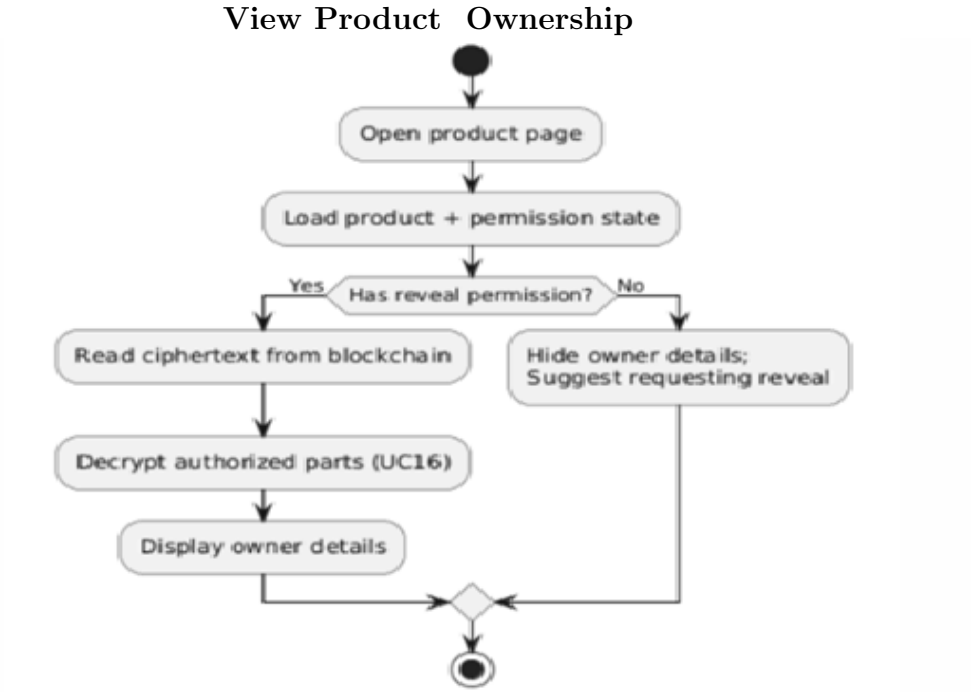

Figure 3.34: System Activity Diagram 8

In Figure 3.35, the activity diagram illustrates how a user requests that ownership information be revealed. The user opens the product page and submits a reveal request, after which the system creates a RevealRequest record with a pending status. The current owner is then notified so they can later approve or reject the request.

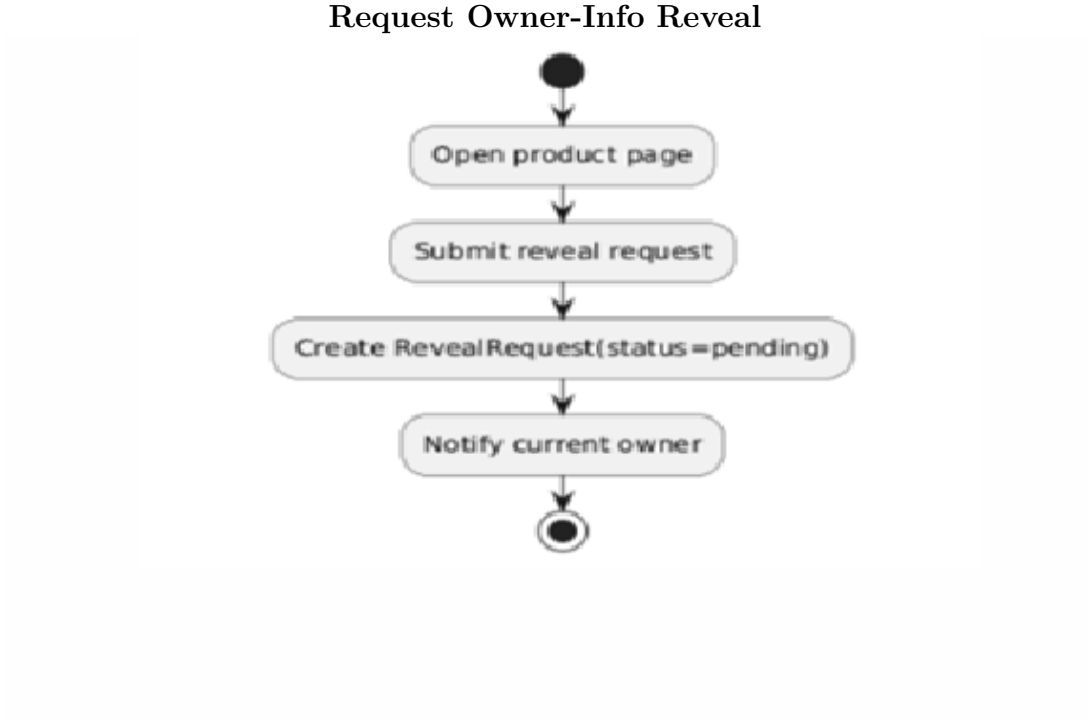

Figure 3.35: System Activity Diagram 9

In Figure 3.36, the activity diagram illustrates how the current owner processes ownership-info reveal requests. The owner first views the list of pending requests and opens a specific request to review it. If the request is approved, the system grants reveal permission for the requester and product, updates the request status to approved, records the decision timestamp, and notifies the requester, if the request is denied, the system updates the status to denied, records the decision timestamp, and likewise notifies the requester before the flow ends.

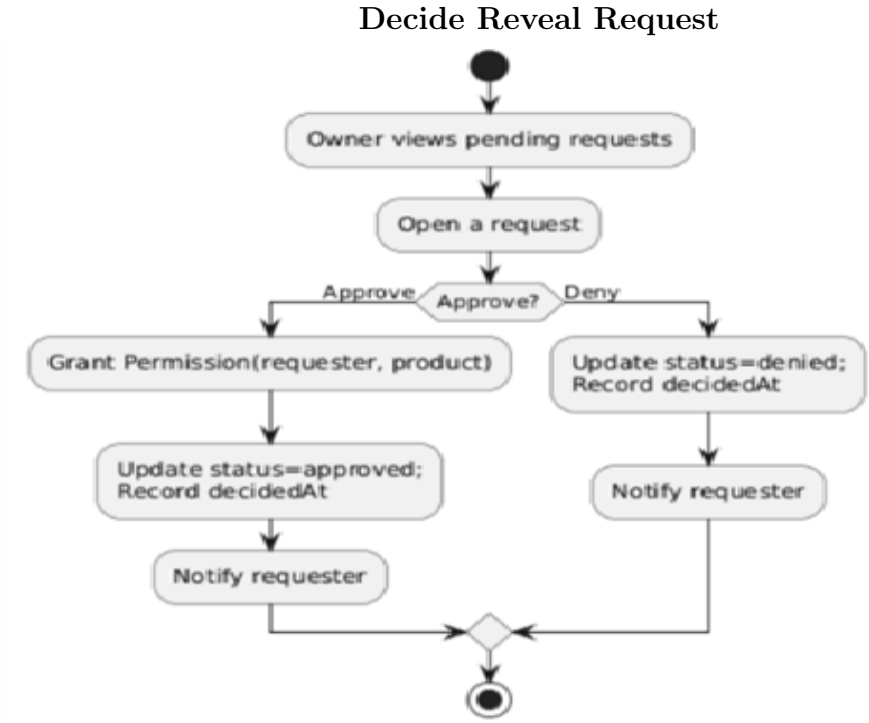

Figure 3.36: System Activity Diagram 10

In Figure 3.37, the activity diagram illustrates how a user initiates a purchase request for a product. After opening the product page, the system first checks whether the user is eligible to buy (for example, they are not the current owner and the product status is ACTIVE). If eligible, the user submits a purchase request, the system creates a PurchaseRequest record with a pending status, and the current owner is notified. If the user is not eligible, the system blocks the action and shows an appropriate reason, such as the product being stolen, retired, or already owned by the user.

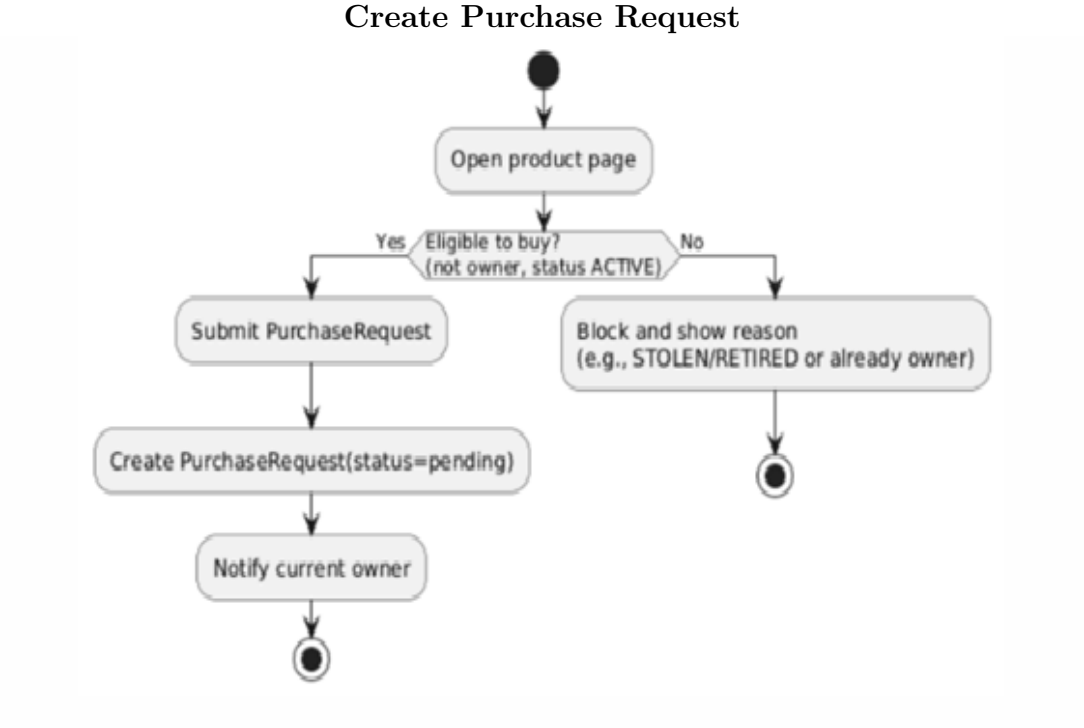

Figure 3.37: System Activity Diagram 11

In Figure 3.38, the activity diagram illustrates how the current owner processes a purchase request and how ownership is transferred on acceptance. The owner reviews the PurchaseRequest and either rejects it marking the request as rejected and notifying the buyer or accepts it, in which case the system ends the previous ownership, creates a new ownership payload for the buyer, encrypts this payload, and writes it to the blockchain, capturing the transaction hash. The system then updates the current owner in the database, marks the purchase request as accepted, and notifies the buyer of the successful transfer.

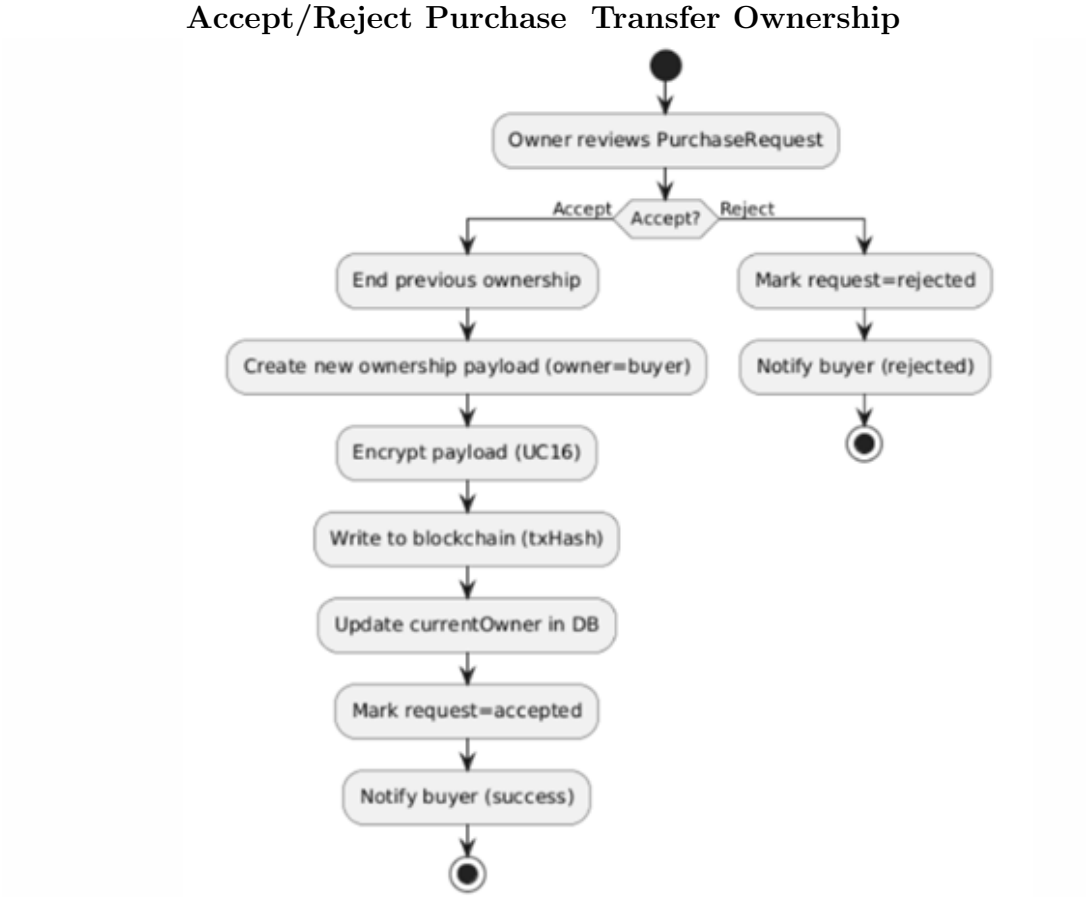

Figure 3.38: System Activity Diagram 12

In Figure 3.39, the activity diagram illustrates how the admin moderates a product's status. The admin selects a product, chooses a new status (ACTIVE, RECALLED, STOLEN, or RETIRED), and the system validates whether this transition is allowed by the defined policy. If the transition is valid, the system updates the product status and inserts a moderation log entry capturing the reason and timestamps if it is invalid, an error is shown and the change is rejected.

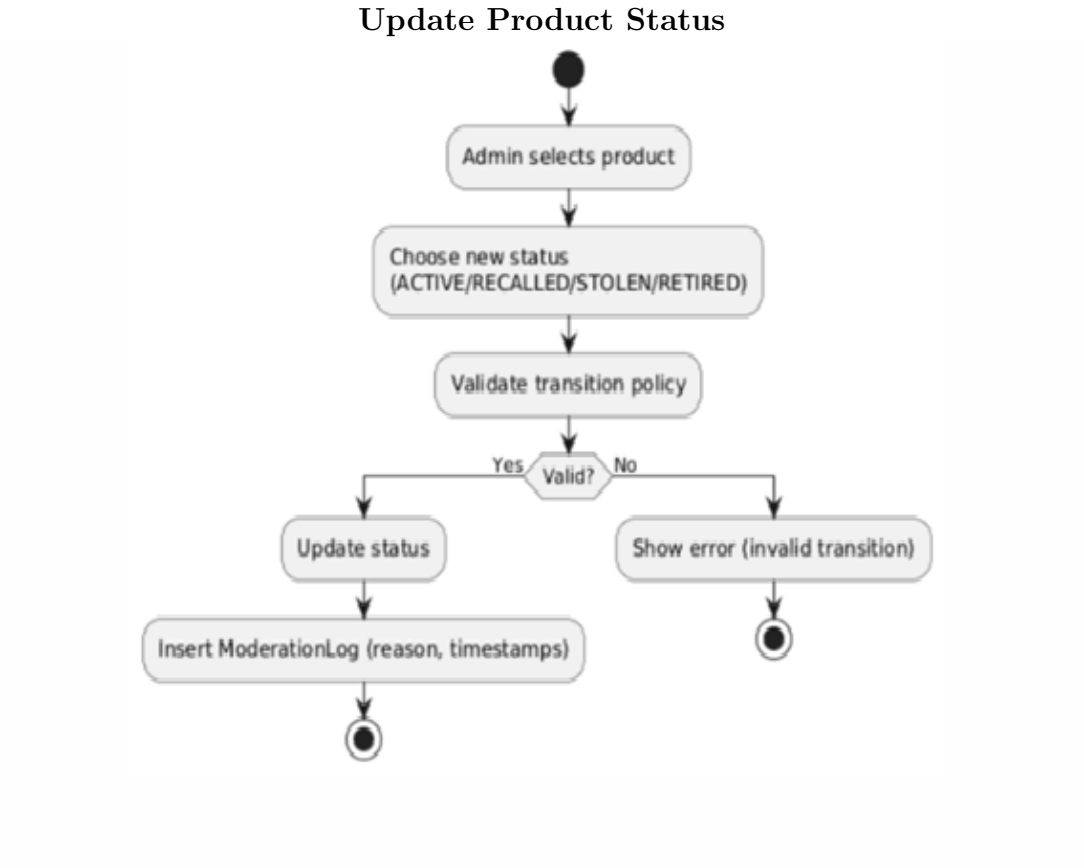

Figure 3.39: System Activity Diagram 13

In Figure 3.40, the activity diagram illustrates how the system returns a product's ownership history while respecting identity permissions. When a user requests the ownership history, the system loads on-chain references together with database events and checks whether the user has permission to see owner identities. If permission is granted, it reads the encrypted events from the blockchain, decrypts the permitted fields, and renders a full timeline including identities.If not,it skips decryption and instead renders the same timeline with sensitive identity information redacted.

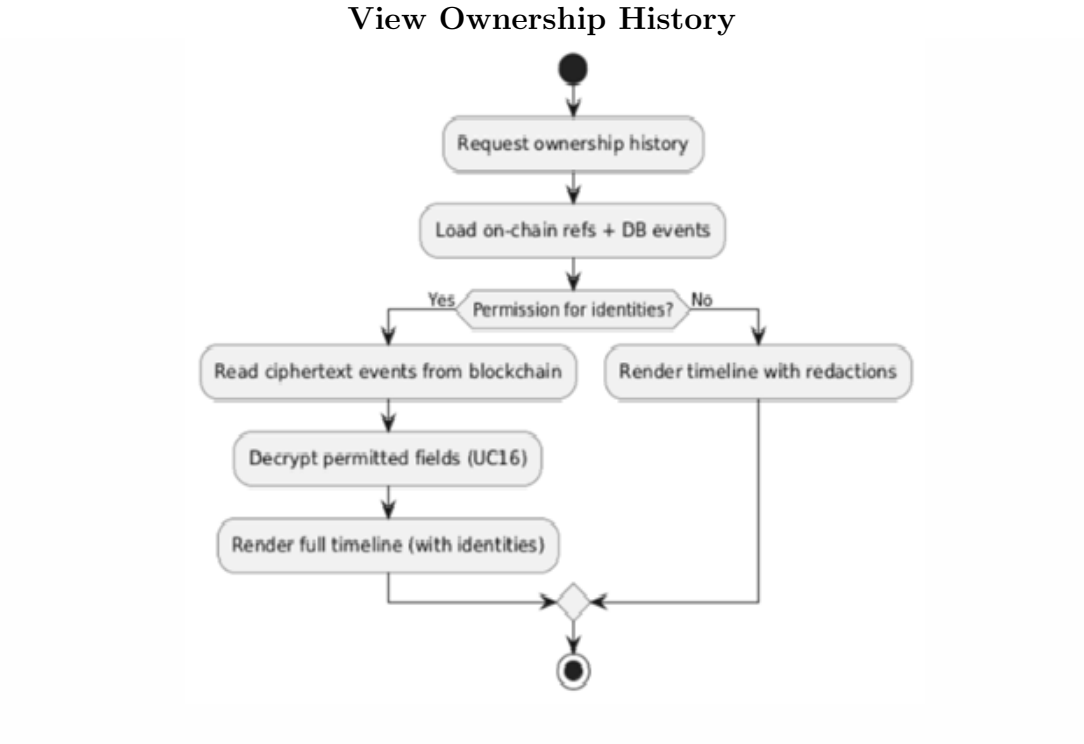

Figure 3.40: System Activity Diagram 14

In Figure 3.41, the activity diagram illustrates how a visitor performs public product discovery. The user opens the discovery page, applies filters or a search query, and the system fetches the matching public product list. If results are found, the system displays product cards or a list showing only public fields if no results match, it shows an appropriate empty state.

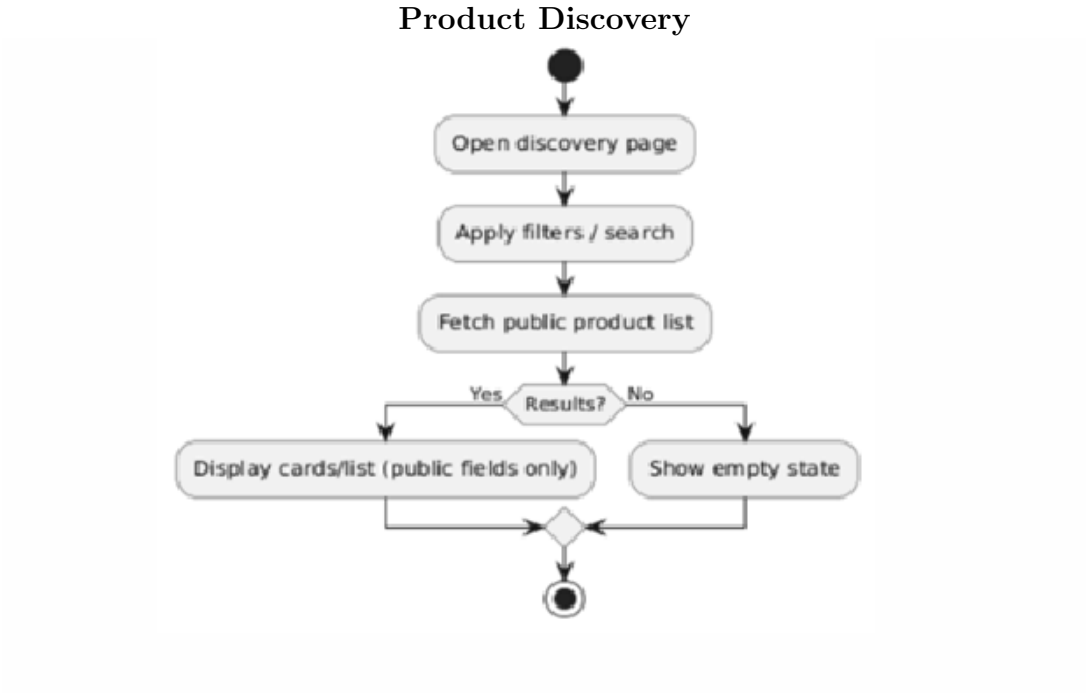

Figure 3.41: System Activity Diagram 15

In Figure 3.42, the activity diagram illustrates the internal cryptographic handling for writing ownership data to the blockchain and reading it back securely. For a write operation, the system composes the ownership payload, fetches encryption keys from a secure store, and, if the keys are valid, encrypts the payload and returns the ciphertext if the keys are invalid, the write is denied and a security event is logged. For a read/decrypt operation, the system fetches the ciphertext using the on-chain reference, checks whether the caller has permission, and, if permitted and keys are valid, decrypts the relevant fields and returns plaintext scoped to the caller's permissions otherwise it denies the read (403) and logs a security event.

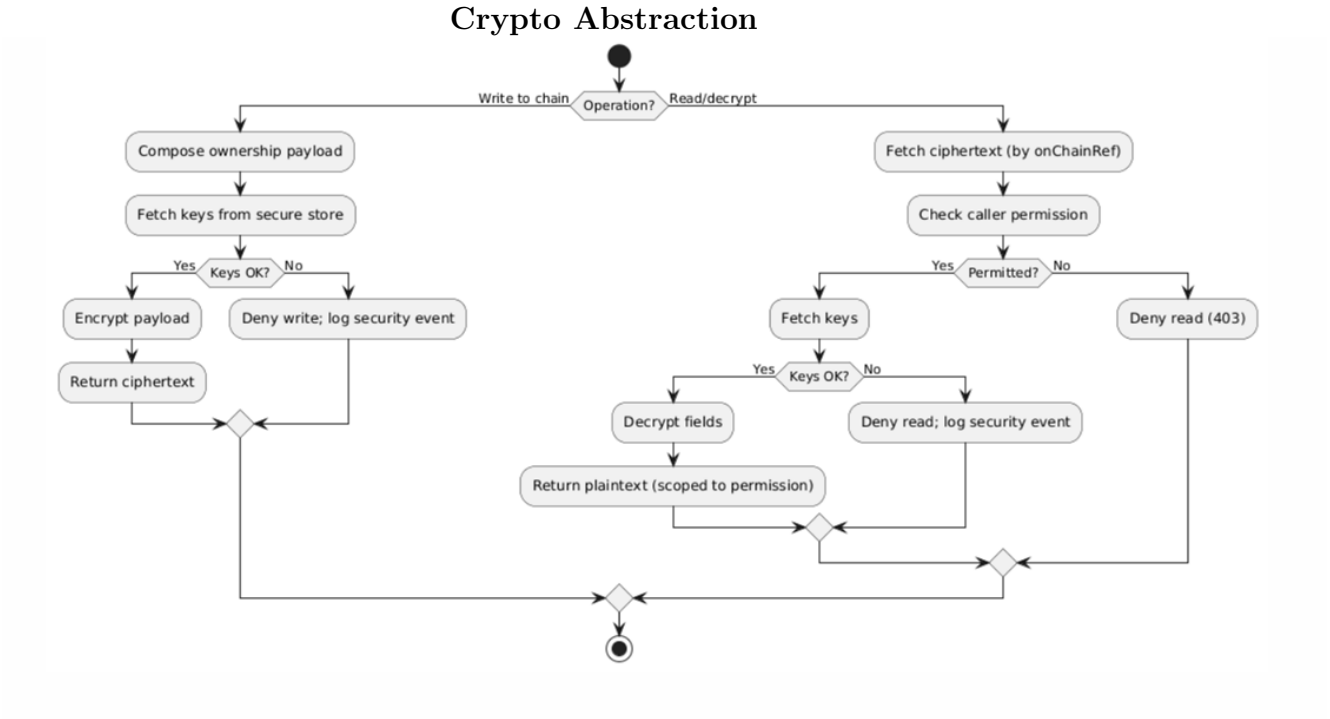

Figure 3.42: System Activity Diagram 16

In Figure 3.43, the activity diagram illustrates how the system guards access to protected routes using JWT-based authorization. When a request reaches a protected endpoint, the system extracts the JWT and first checks whether it is present and valid; if not, the request is denied with a 401 Unauthorized response. If the token is valid, the system extracts the user's role from the token claims and compares it with the route's access policy. If the role is allowed, the request proceeds with a 200 OK response; otherwise, access is denied with a 403 Forbidden response.

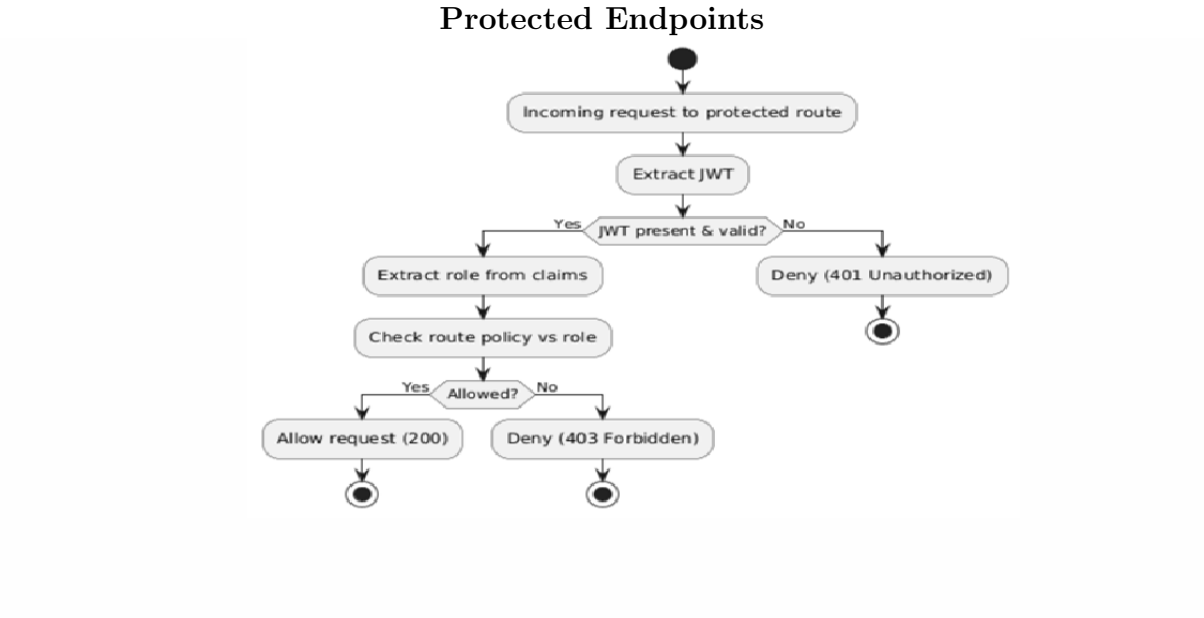

Figure 3.43: System Activity Diagram 17

3.7 System ERD Diagram

The ERD illustrates how the system connects identities, products, and trust-related processes. Store owners are responsible for creating and registering products, each of which is associated with a specific store. Users can interact with products by reviewing or purchasing them. Ownership entities represent both the current holder and the historical chain of owners, and transaction records capture transfers between parties. Verification requests manage access and disclosure workflows, while product flags and contact records support moderation and user assistance.

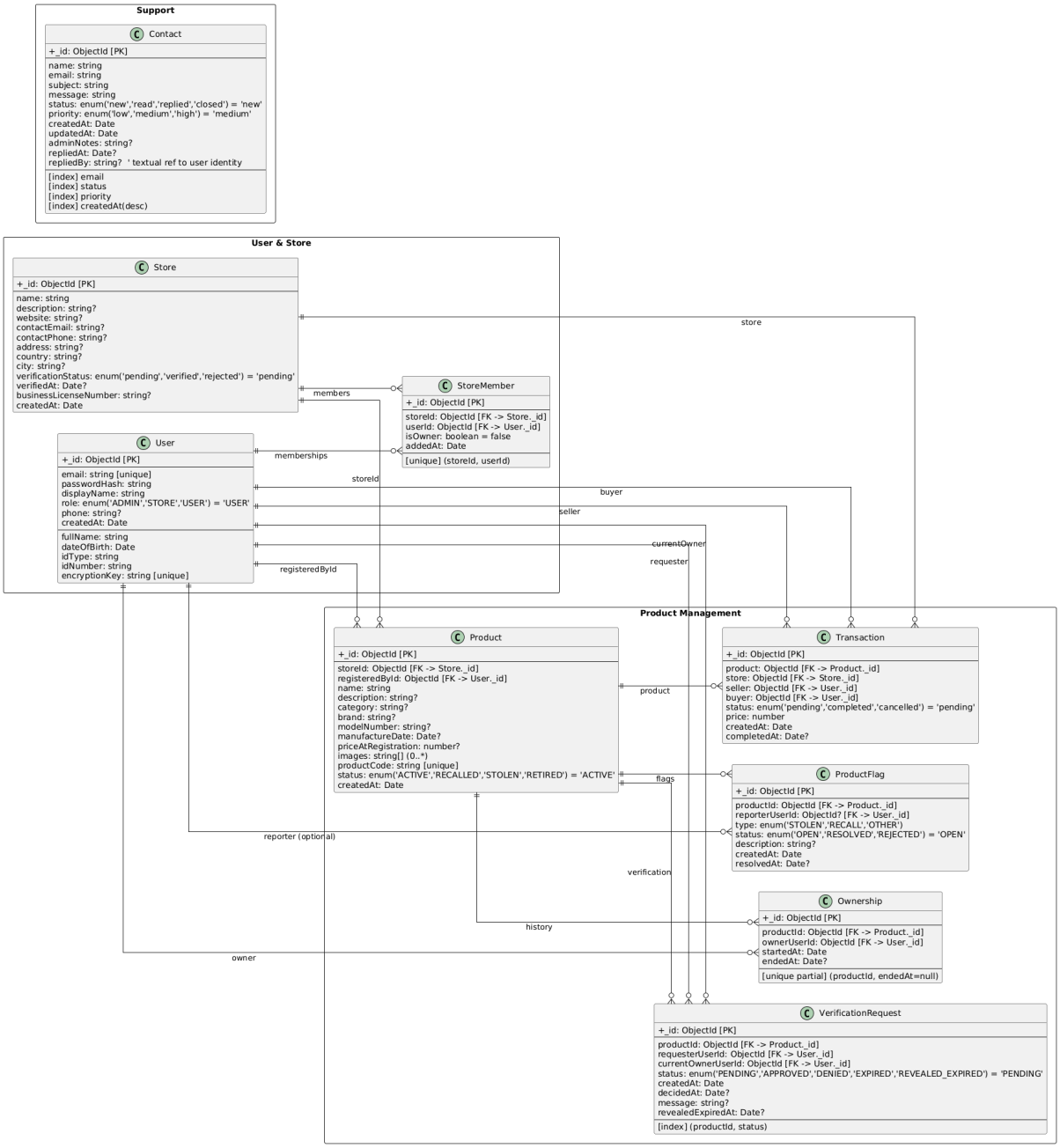

Figure 3.44: System ERD Diagram

Chapter 4
Design and Methodology

This chapter describes the implemented design of the Blockchain-Based Product Authenticity Verification System. The system follows a hybrid approach, where encrypted ownership records are written to the blockchain, while product metadata, workflow entities, and statuses are managed in the backend database. Three roles are supported: ADMIN, STORE, USER. The core flows include product registration, public search by code, verification and visibility requests, purchase transfer, and admin moderation.

4.1 System Architecture

The architecture consists of a Next.js client, an Express-based REST API, MongoDB for persistence, and a Solidity contract accessed via a back-end blockchain API. Ownership data is encrypted on the server side, and the resulting encrypted string is submitted on-chain, ensuring that no plaintext is stored on-chain.

- **Client (React/Next.js + TypeScript + Tailwind):** Provides role-based dashboards of ADMIN, STORE and USER. It enables searching by product Code in public and shows product views and also processes request flows (purchase and verification).
- **API (Node.js/Express):** Handles authentication with JWT, RBAC middleware, product and ownership logic, submission of encrypted ownership data to the blockchain API, and admin status moderation.
- **Database (MongoDB via Mongoose):** Maintains collections for users, products, ownerships, transactions, and verification requests with indexes such as `productCode`.
- **Blockchain (Solidity contract via backend API):** Provides an append-only function that stores encrypted ownership records keyed by `productCode`.
- **Cryptography:** Ensures ownership data is securely encrypted for each user with a private key created at signup, so sensitive information remains protected.

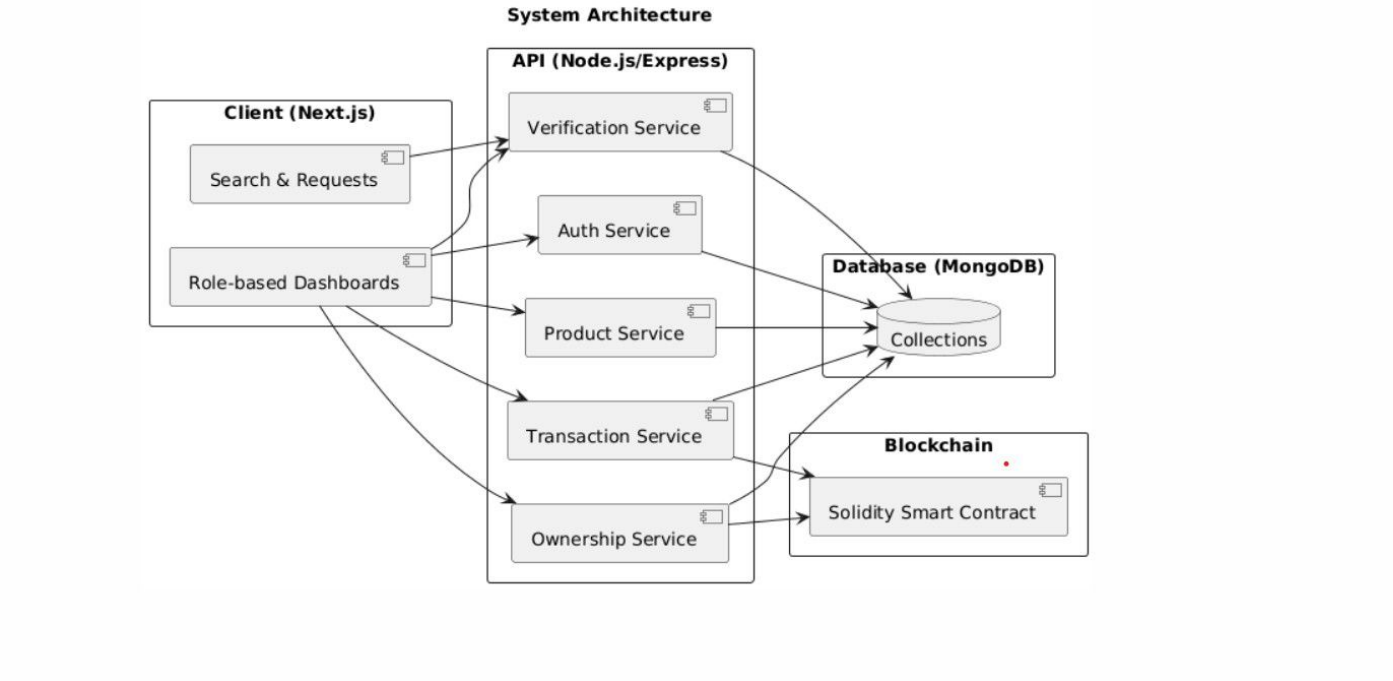

Figure 4.1. System Architecture of the Blockchain-Based Product Authenticity Verification System

4.2 Data Design

On-chain: Encrypted ownership records per `productCode`. **Off-chain:** Product metadata, workflow entities (transactions and verification requests), and product status stored in MongoDB.

4.2.1 Collections (representative)

- `users`(_id, email, passwordHash, role, encryptionKey, createdAt)
- `products`(_id, productCode, storeId, name, description, category, brand, modelNumber, manufactureDate?, priceAtRegistration, status, createdAt)
- `ownerships`(_id, productId, ownerUserId, startedAt, endedAt?)
- `transactions`(_id, productId, buyerId, ownerId, status, createdAt)
- `verificationRequests`(_id, productId, requesterId, ownerId, status, createdAt)

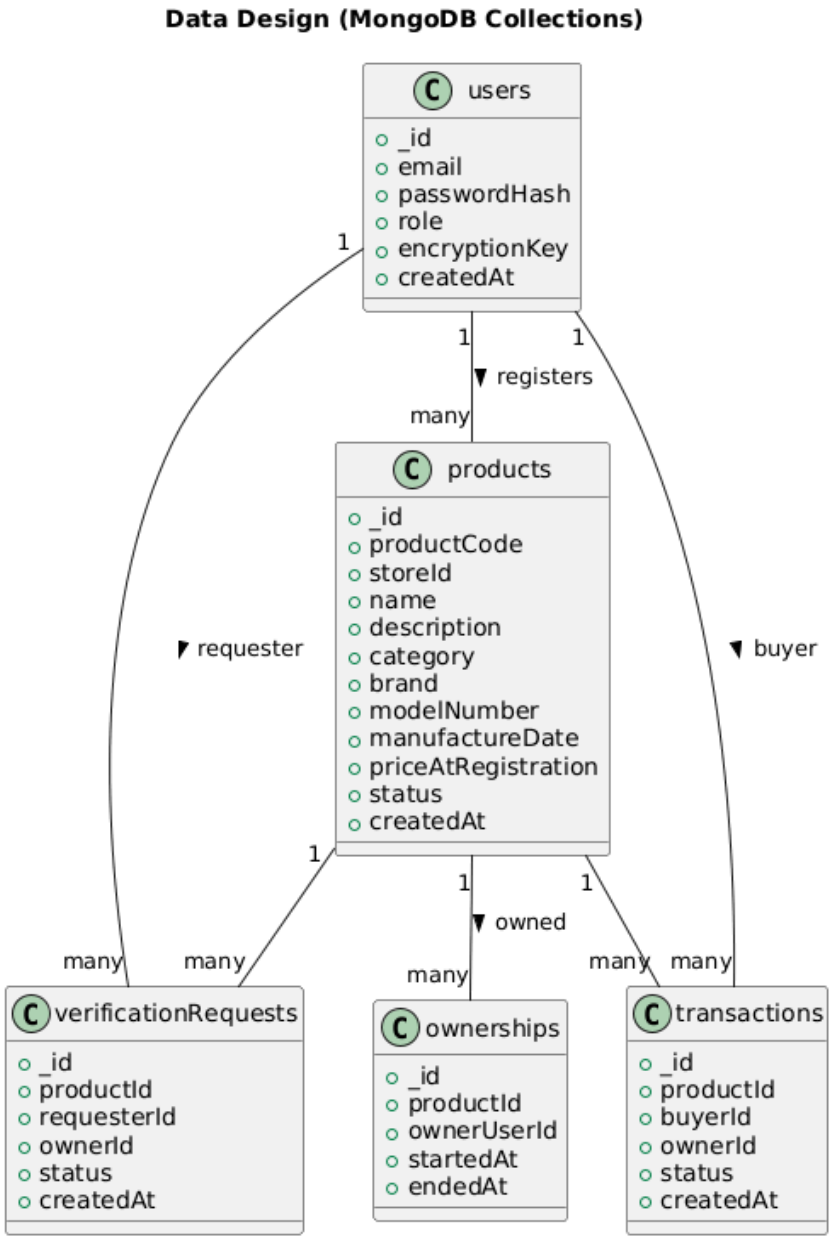

Figure 4.2. Data Design of the Blockchain-Based Product Authenticity Verification System

4.3 High-Level Design

4.3.1 Concepts

- **Product** is identified by a human-friendly `productCode`.
- **Ownership** is represented as encrypted data stored on-chain.
- **Requests** are used to manage verification and purchase or transfer flows.
- **Status** (`ACTIVE`, `RECALLED`, `STOLEN`, `RETIRED`) is maintained in the database for display.

4.3.2 Core flows

1. **Register Product:** The store fills out a form to create a new product. The backend saves the product details and sets the STORE as the initial owner. The ownership data is encrypted and stored on the blockchain, while all relevant information is also stored in the database.
2. **Search by Code:** A user can enter a product code to search for a specific product. The product details and status are displayed, while the owner's information remains hidden from public view.
3. **Verification:** The USER sends a request asking the owner to share their information. The owner can choose to accept or reject the request. If the owner approves, the permitted information is revealed to the USER, while the on-chain data remain encrypted.
4. **Purchase & Transfer:** A USER sends a purchase request to the current owner. Once the request is accepted by the owner, the ownership is securely updated, with records saved in the database and also stored on the blockchain in encrypted form.

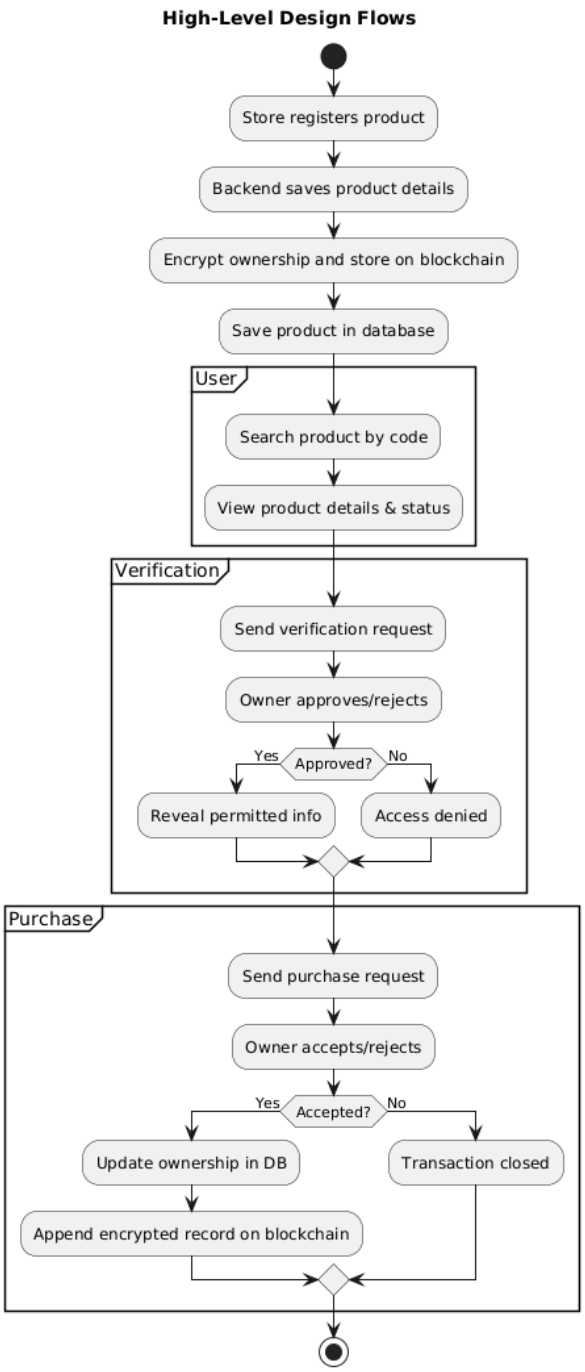

Figure 4.3. High-Level Design of the Blockchain-Based Product Authenticity Verification System

4.4 Interfaces

4.4.1 Smart Contract

- `addOwnershipRecord(productCode, encryptedData, ownershipStartDate)` — appends an encrypted ownership record.

4.4.2 REST API (key routes as implemented)

- **Auth:** `POST /auth/login`, `POST /auth/signup`
- **Products:** `GET /products`, `GET /products/code/:productCode`, `POST /products` (STORE)
- **Transactions:** `POST /transactions/initiate/:productId`, `POST /transactions/confirm/:t` `POST /transactions/reject/:transactionId`, `GET /transactions/my-requests`
- **Ownership:** `GET /ownership/product/:productId/current`, `GET /ownership/user/:userId`, `GET /ownership/my-products`, `POST /ownership`
- **Verification:** Representative `/verification` routes for listing by owner, product, or user (protected)
- **Admin:** Product status moderation via protected admin routes

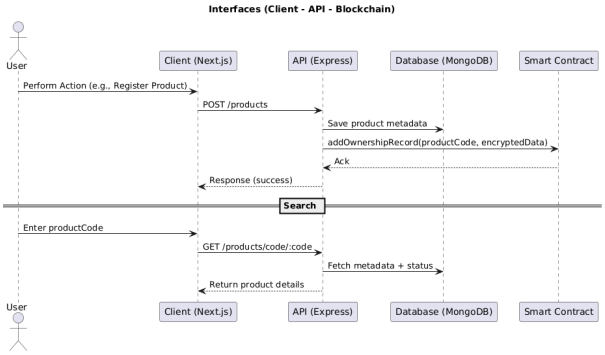

Figure 4.4. Interfaces between client, API, database, and blockchain in the proposed system, showing product registration (metadata to MongoDB, encrypted ownership on-chain) and product search (lookup by code, combining database and blockchain data for the client response).

4.5 Security

- **Authentication:** Uses JWT with role claims.
- **Authorization:** Enforced with RBAC middleware for `ADMIN`, `STORE`, and `USER`.
- **Passwords:** Passwords are securely hashed before being saved to the database.
- **Encryption:** Uses a unique key generated for each user at signup, and encryption is performed on the server before storing data on the blockchain.
- **Privacy:** Data is encrypted and stored on the blockchain. For ownership verification, the owner receives a request and must approve it. Only after approval is the relevant data decrypted and shared with the requesting user.

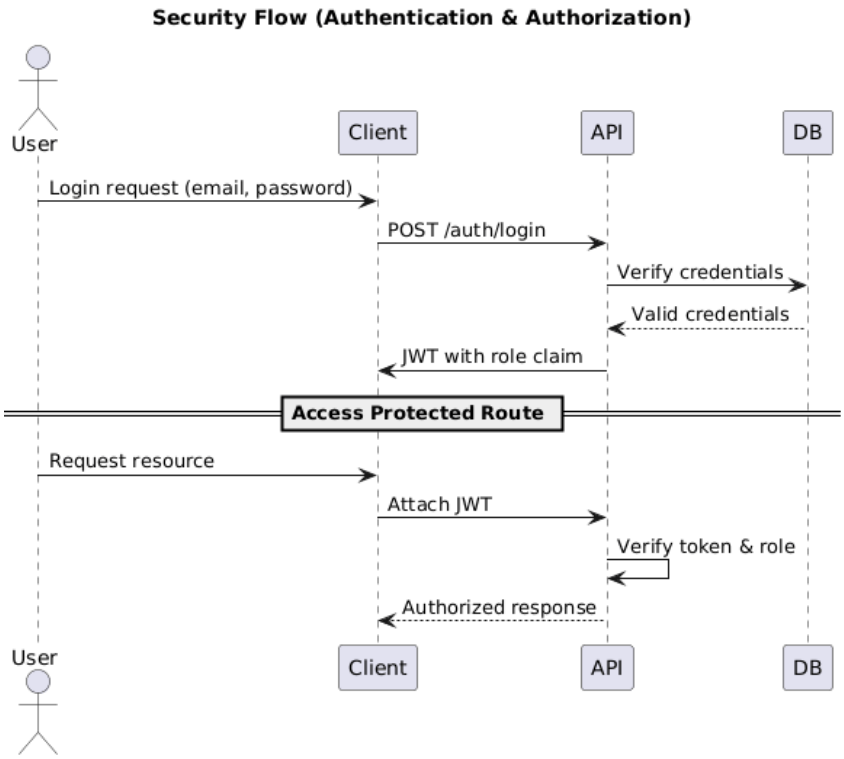

Figure 4.5. Security flow for authentication and authorization in the proposed system, where the client sends login credentials to the API for validation, receives a role-bearing JWT on success, and then attaches this token on subsequent requests so the API can verify it and enforce role-based access to protected routes.

4.6 Design Constraints

- **On-chain costs:** The blockchain only stores essential encrypted ownership information, and all other data is not stored on the blockchain to minimize costs associated with the blockchain.
- **Scalability:** Instead of constantly reading from the blockchain, heavy read operations are handled through MongoDB for faster and more efficient access.

4.7 Design Methodology

The system is designed according to an incremental and modular design approach that guarantees a clear system, scalability and safe management of product ownership information. The stages involve the continuous enhancement of the functionality with the high-level architectural discipline.

- **Requirement Analysis:** This task was done by ensuring that the non-functional and functional requirements of the system were well identified. The main needs such as registering technology products, transfer of ownership as well as identifying who they are were divided into smaller activities to comprehend system constraints and user requirements. The special focus was given to the traceability, records of the authenticity of products and the adherence to the real-world ownership processes.
- **Hybrid Approach:** To trade off cost, performance and security a hybrid architecture was embraced. Immutable ownership data is stored on-chain to ensure transparency

and tamper-proof verification with details about products being stored off-chain to lower the costs of blockchain storage and ensure scalability. The division also enables effective updates of the product metadata without any interference of the integrity of core ownership records.

- **Role Based Design:** The system defines separate workflows for ADMIN, STORE, and USER to ensure controlled access and prevent operational overlap. ADMIN manages system-wide configurations, while STORE handles product registration and ownership verification. USER is allowed to search products through their product code, purchase items, and send a request to the seller to reveal seller information when needed. This role-based structure provides a secure and organized user experience.
- **Security-driven Development:** Security was also integrated into all the stages as opposed to being added later. Methods like encryption of sensitive information, protection authentication system, and Role-Based Access Control (RBAC) were implemented to avoid unauthorized access. Other concerns were the protection of API endpoints, smart contract integrity, and safe transactions handling in an attempt to secure user and ownership information.
- **Iterative Refinement:** This application uses customer messages as its main source of data. The system was built in small increments where every module was created, tested and validated including the authentication module, product registration, blockchain storage module, and transfer workflow module. Integration and readjustment of modules was done after the verification to guarantee a seamless integration. The iterative method enhanced maintenance, minimized errors at a young age and continuous improvement due to the testing feedback.

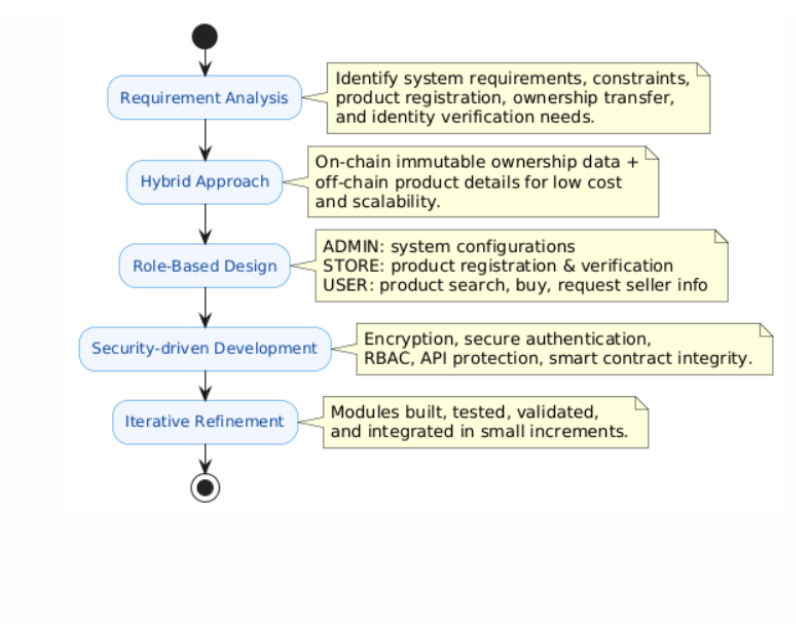

Figure 4.6. Design Methodology

4.8 Gui Design

Landing Screen: The initial page greets users and explains the objective to authenticate a product. Here, users can start verification, view a brief demonstration, or explore key sections of the site such as Features and how it works.

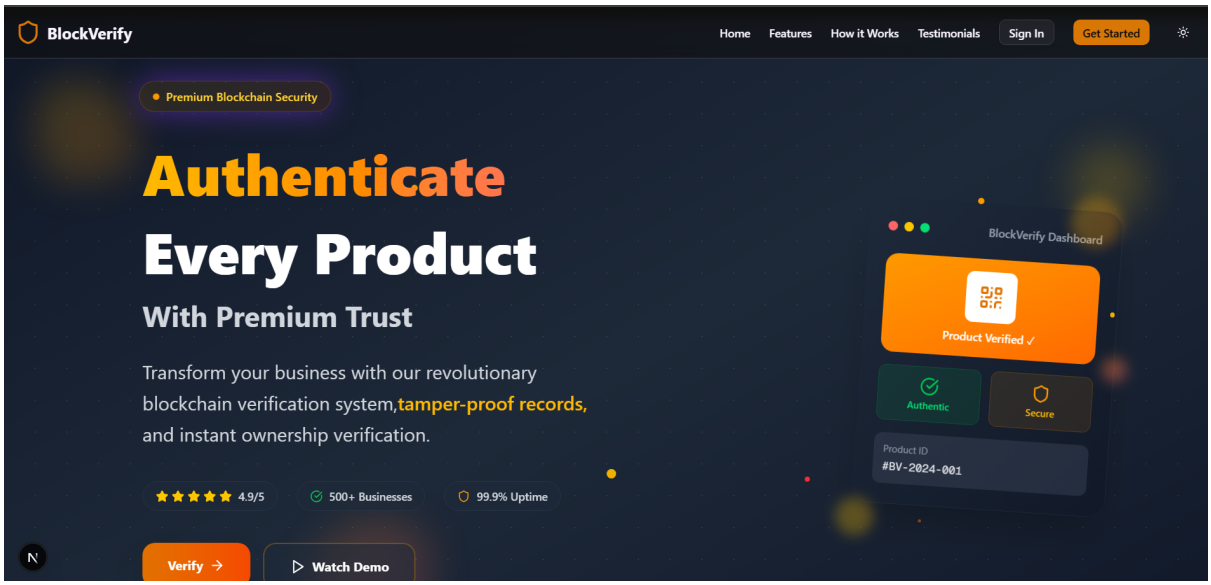

Figure 4.7. Landing Page

User Search Page: Customers begin at this point. Here, users type in the product code issued at registration to search for a product and see the basic information to ensure authenticity before expressing interest in making an offer.

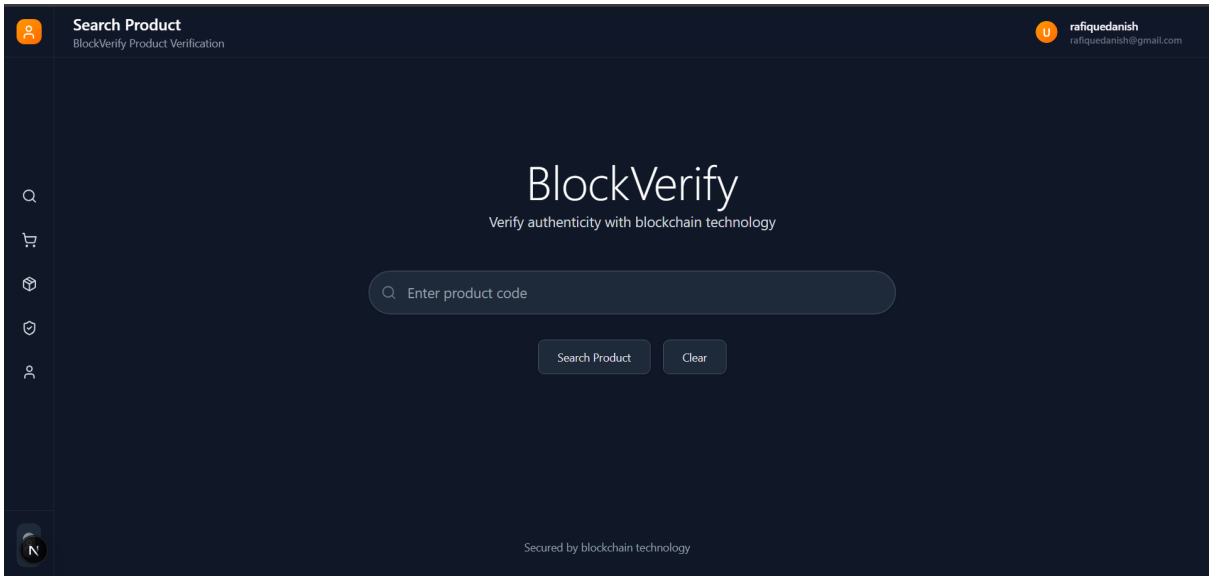

Figure 4.8. User Search Page

Product Verification and Purchase Offer: After entering a code, the user sees key details and the current status. From here, they can send a verification request or submit a purchase offer, which is then reviewed by the store.

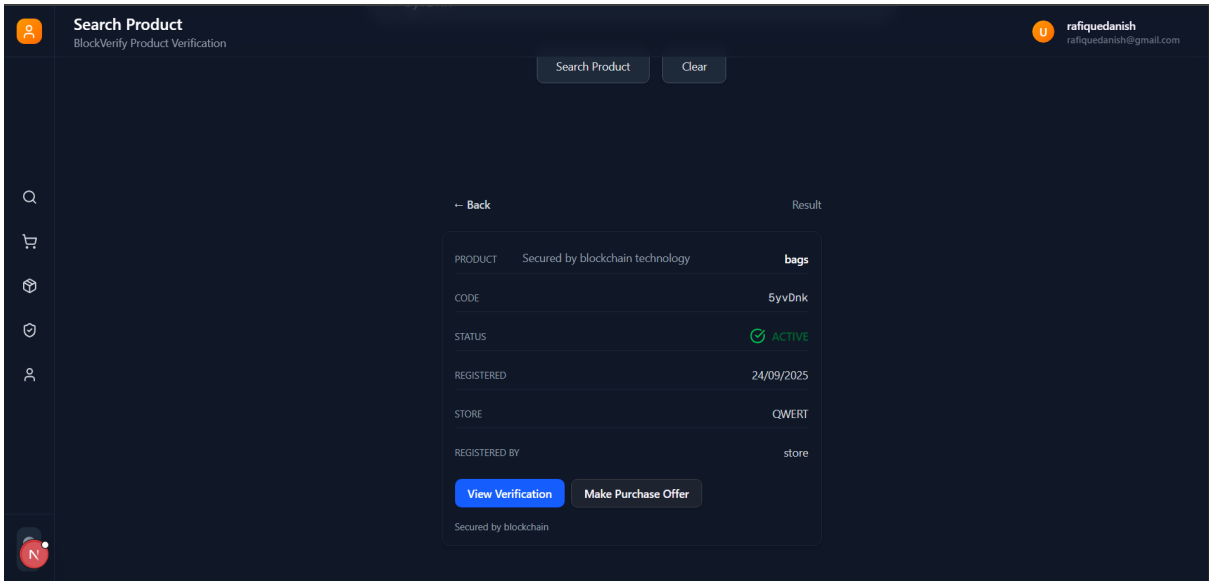

Figure 4.9. Product Information

Purchase Offer: An overlay lets the user review the product summary, enter an offer amount, and submit it for store/owner review. A success message confirms that the offer has been sent.

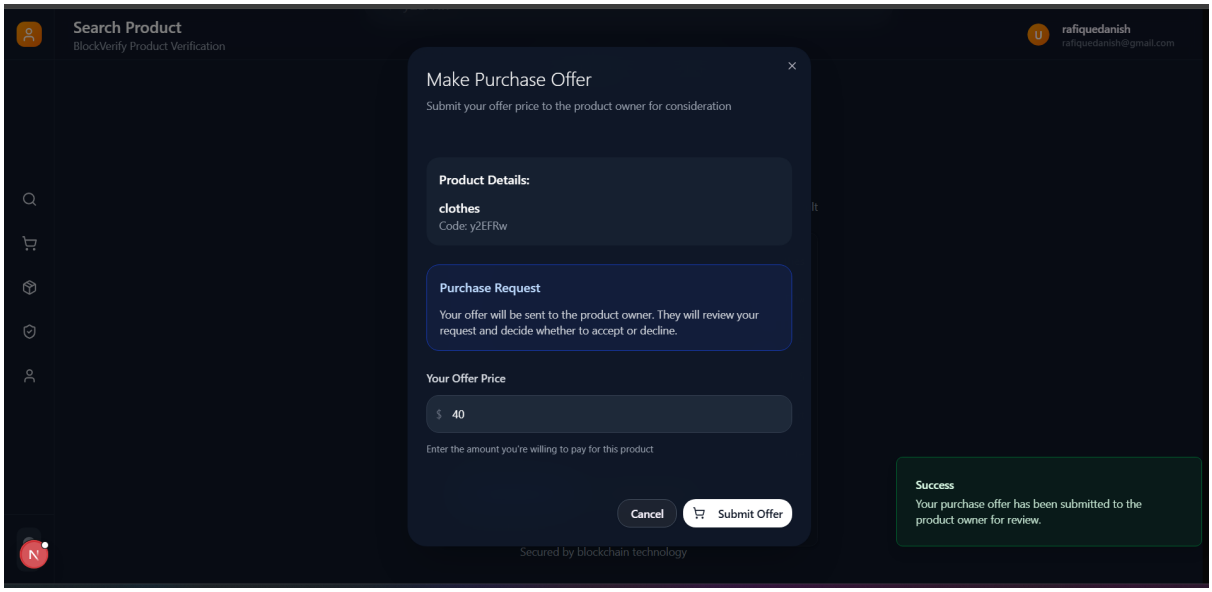

Figure 4.10. Purchase Request

Store Dashboard: The starting page for store owners gives a quick view of how many stores and products they have and shows if anything needs attention. From here, they can create a store, add a product, search, open settings, and quickly jump to recent items.

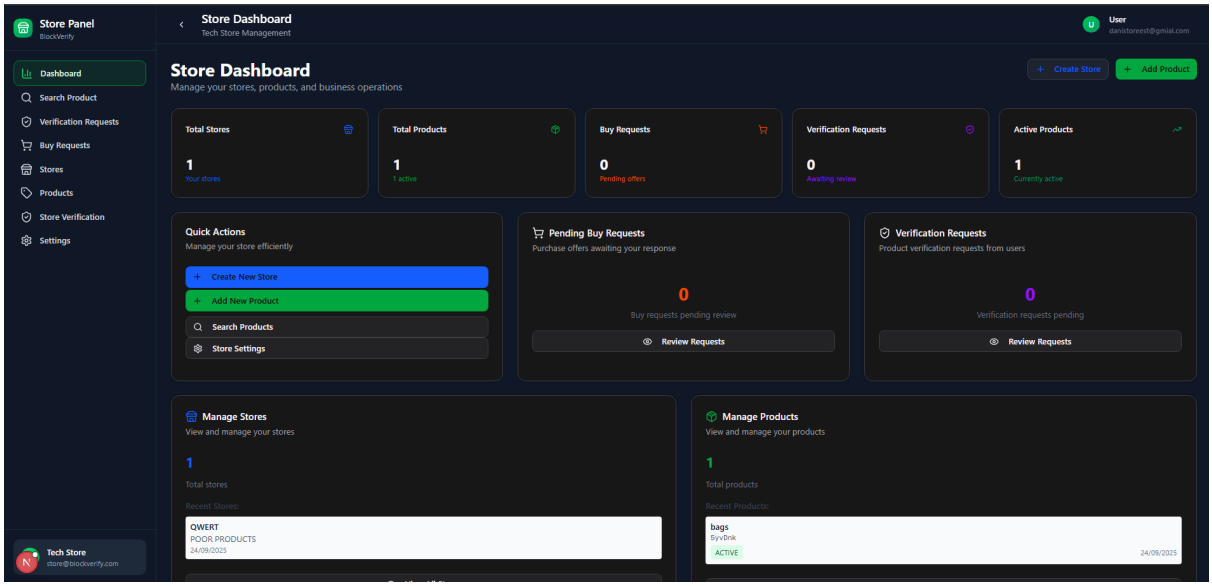

Figure 4.11. Store Dashboard

Store Managing User Buy Requests: The place where the store reviews offers from users. It shows simple counts, a search bar, and tabs for Pending, Sold, and Cancelled. From here, the store can open any request, check details, and decide what to do.

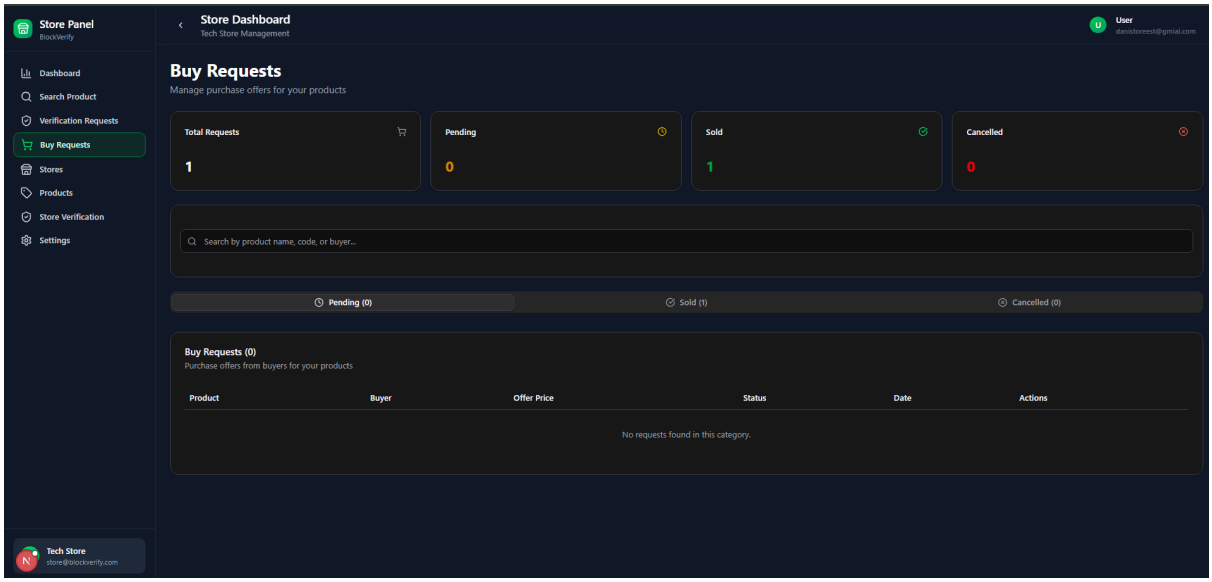

Figure 4.12. Buy Requests (Store)

Store Verifying User: A page where the store reviews and responds to users' requests to view product information. Each card shows the product, who requested it, and when. The store can open details and update the request.

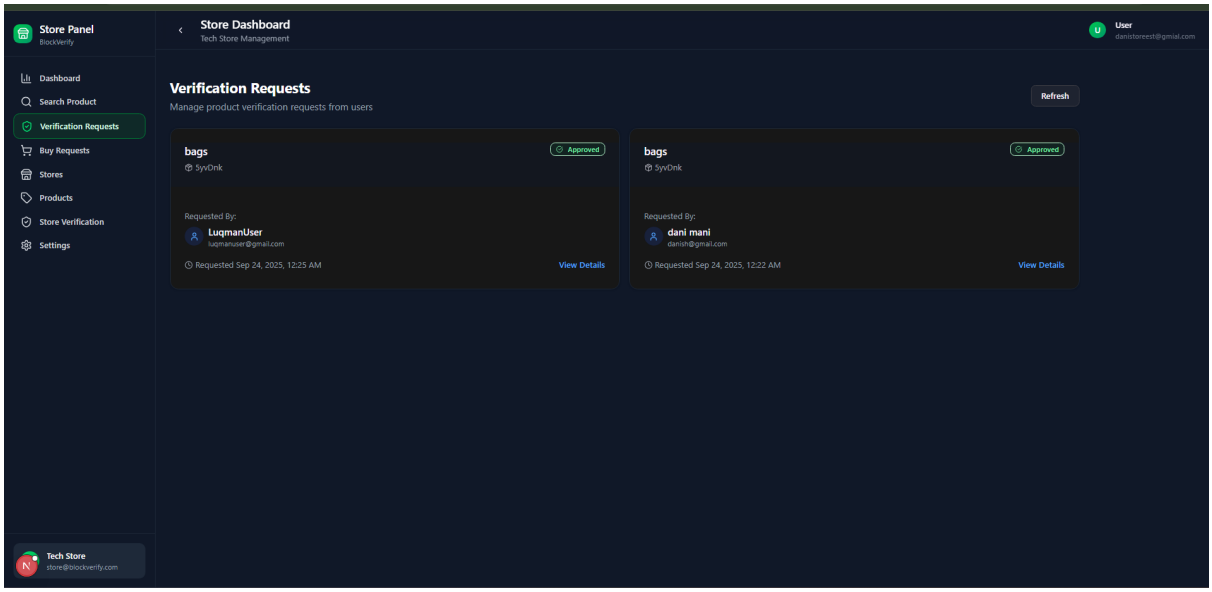

Figure 4.13. Verification Requests

Admin Store Management: An overview page where the administrator sees all stores at a glance. It shows simple totals, a search box, and quick filters (All, Verified, Pending, Rejected). Each row lists a store with basic contact and location information, and the admin can open a store to view more or take action.

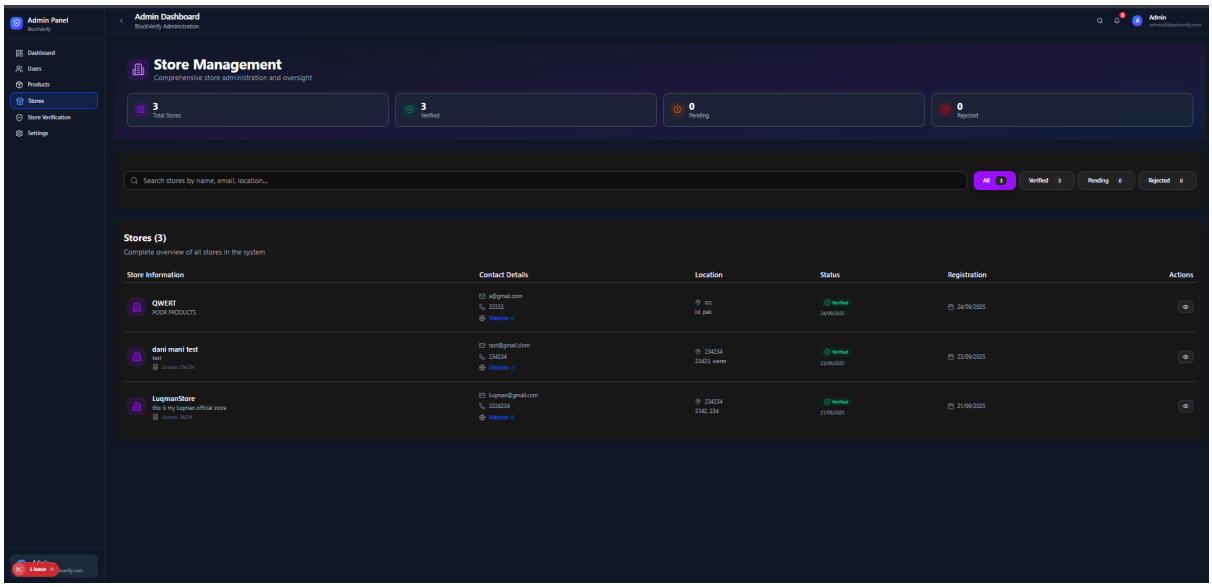

Figure 4.14. Admin — Store Management

Admin Store Verification: The page where the administrator reviews store registrations, checks basic details, and approves or rejects each store.

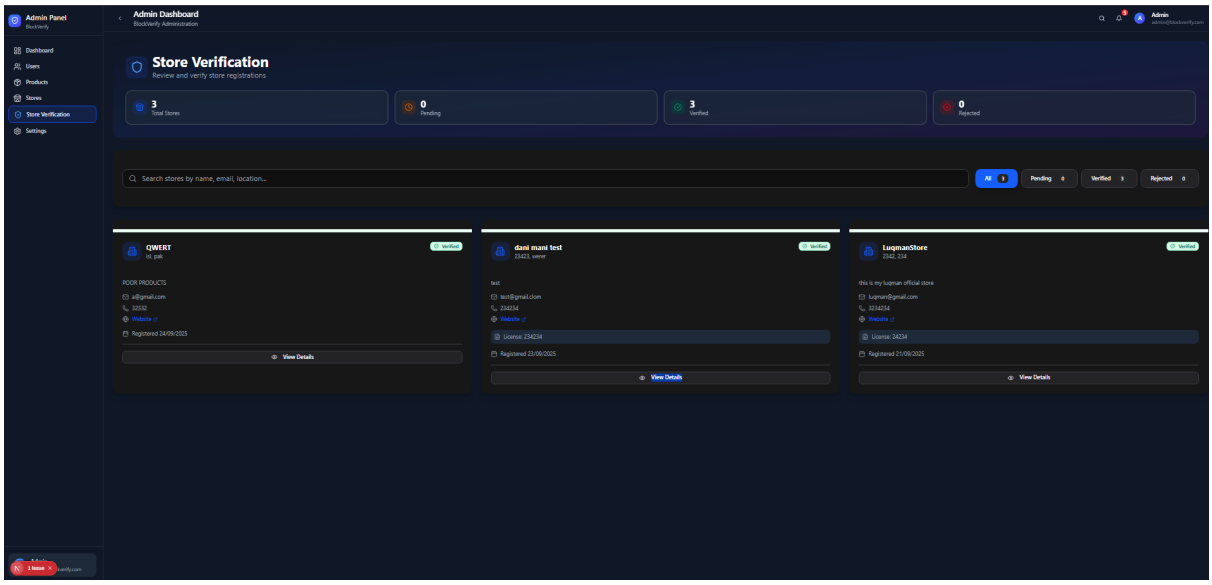

Figure 4.15. Admin — Store Verification

Admin Dashboard: The administrator home screen. It provides a concise summary of the entire system: the number of users, products, and stores, areas of concern, current activity, and shortcuts to popular tasks.

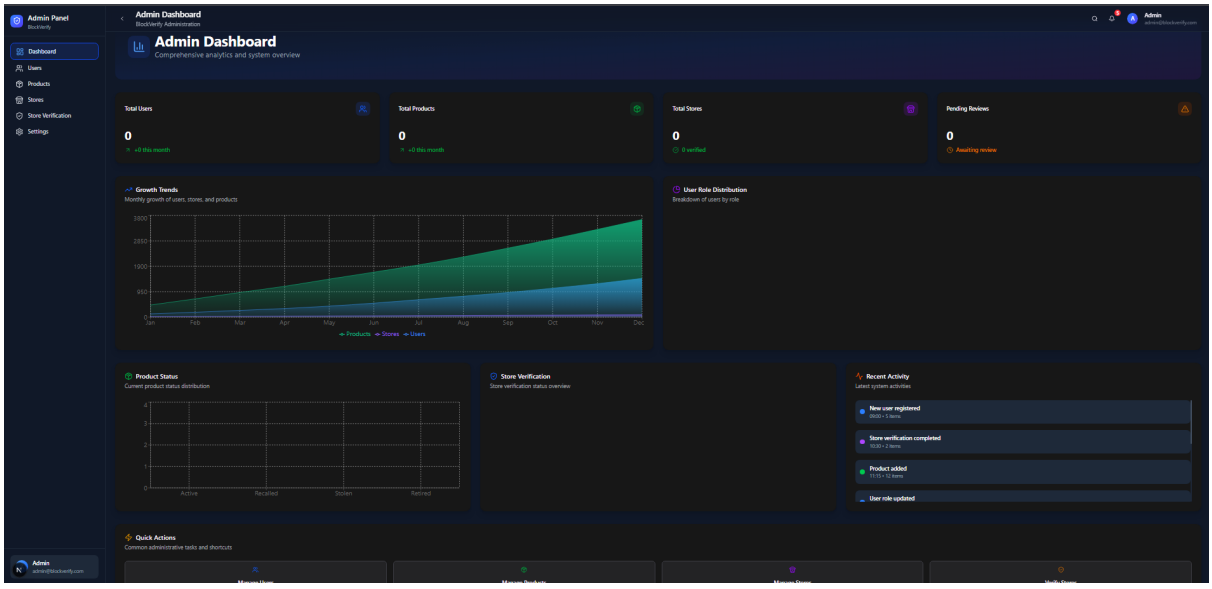

Figure 4.16. Admin Dashboard

Chapter 5
System Implementation

This chapter documents the concrete implementation of the Blockchain-Based Product Authenticity Verification System. The deployed model follows a hybrid architecture, with encrypted ownership data stored on-chain and application metadata and workflow state stored in the traditional database. The system supports three roles: **ADMIN**, **STORE**, **USER**, and implements the key flows: product registration, purchase transfer, and verification workflows.

5.1 System Internal Components

- **Frontend Layer (React/Next.js + TypeScript + Shadcn UI):** Role dashboards for ADMIN, STORE, and USER.
- **API Layer (Express.js REST):** Authentication, product registration and search, purchase and transfer workflows, verification request endpoints, and administration tasks.
- **Data Access Layer (MongoDB):** User collection, product collection, ownership snapshot collection, transactions collection, and verification request collection.
- **Blockchain Access Layer (Solidity + Hardhat Ethereum Node + API):** A thin contract interface, which is a productCode-keyed encrypted ownership record, and interacts with a blockchain API on the backend.
- **Cryptography Layer:** Ownership data is securely encrypted using a unique key issued to the user during sign-up and stored with their account.

5.2 Functionality of Components

- **Frontend:** Displays a responsive UI, interacts with REST endpoints, presents public information where authorized, and indicates states for operations.
- **Backend:** Validates requests, enforces RBAC, issues and authenticates JWTs, generates productCode, encrypts ownership payloads, stores records, posts encrypted ownership to the blockchain API, and handles status in the database.
- **Database (MongoDB):** Stores (users, products, ownerships, transactions, verificationRequests) and maintains indexes on identifiers such as productCode and request status.
- **Smart Contracts:** Encrypted records are added for each product.
- **Cryptography:** Ownership records are encrypted and decrypted on the server side.

5.3 Component Communication

- **Frontend → Backend:** REST/JSON for login, product registration and search, purchase requests, verification requests, and admin actions.
- **Backend → MongoDB:** CRUD operations with appropriate indexes for efficient reads.
- **Backend → Blockchain:** The backend sends encrypted ownership data to the blockchain via an API, and the blockchain updates the record accordingly.

5.4 Tools and Technology

5.4.1 Development Environment / Languages

- **TypeScript:** Provides type safety and an improved developer experience across both frontend and backend.
- **Node.js (Express.js):** Enables the server-side runtime and the creation of RESTful APIs while handling authentication.
- **Next.js (React, Shadcn) + Tailwind CSS:** Builds dynamic and responsive user interfaces with efficient styling and server-side rendering.
- **MongoDB:** Serves as the primary database for storing product details, ownership records, transactions, and verification requests in a flexible document format.
- **Solidity:** Implements and interacts with smart contracts on the blockchain.
- **Postman:** Helps in manual testing and debugging of API endpoints to ensure correct functionality.

5.4.2 Processing Logic

- **Product Registration:** The system generates a human-friendly productCode, encrypts the initial ownership payload with the user's per-user key, stores product and ownership metadata in MongoDB, posts the encrypted ownership data to the blockchain through a backend API, and returns product and operation details.
- **User Search:** Users can search using productCode to retrieve product details and current status, while owner identities remain confidential.
- **Verification Workflow:** Users may submit verification requests, which the owner reviews. When an owner approves a request, the relevant information is revealed.
- **Purchase & Transfer:** A buyer initiates a purchase request, and on acceptance, the system creates a new ownership snapshot in the database and appends a new encrypted ownership record on-chain for the same productCode.

5.5 Application Access Security

Security controls are aligned with the hybrid model and role separation.

5.5.1 Authentication

- **JWT-based authentication:** Tokens include role claims.
- **Password hashing:** bcrypt is used for all stored passwords with pre-save hashing.
- **Route protection:** The frontend hides restricted UI, while backend middleware enforces access with authentication and role checks.

5.5.2 Authorization

- **RBAC:** ADMIN moderates status and manages admin-level views, verified STORE registers products, USER can submit verification and purchase requests, and the current owner participates in transfer and verification decisions as defined by protected flows.

5.5.3 User Data Protection

- **No plaintext on-chain:** Only encrypted ownership data is stored on the blockchain, while product metadata is stored in the conventional database.

5.5.4 Product / Ownership Data Security

- **Encrypted ownership on-chain:** Ownership data is encrypted on the server before being stored on the blockchain.
- **Per-user keys:** Each user has a unique encryption key created at signup and stored with their profile.
- **Visibility control:** Users cannot see the owner's information unless they request the owner to reveal it, and the ownership details are only visible if the owner approves the request.

Chapter 6
System Testing and Evaluation

6.1 Introduction

System testing validated that the backend (Express + MongoDB + blockchain API integration) and the frontend (Next.js) behave as intended across the core user journeys implemented in this application: authentication, product registration, public search by product code, verification workflow, purchase/transfer, and admin status moderation. The objectives were correctness, resilience to invalid inputs, responsiveness across devices, and acceptable performance.

6.2 Testing Approach

Backend APIs were tested using the **REST Client VS Code extension** (`.http` files) to verify request/response correctness, validation, and authorization. Frontend behavior was checked with Lighthouse audits and manual responsive testing using browser developer tools.

6.2.1 Backend Testing with REST Client

The REST Client extension was used to send API requests, check responses (status, payloads), and validate role-based access control.

6.2.2 Focus Areas

- Validation of required fields and formats with a consistent error shape { error: "message" }.
- JWT authentication and role enforcement across protected routes.
- Blockchain write on product creation (encrypted ownership record) and graceful handling of failures.

6.2.3 API Scenarios

Authentication (/auth)

- `POST /auth/signup` – Creates user, validates required fields, prevents duplicates, generates per-user encryption key.
- `POST /auth/login` – Returns JWT for valid credentials, rejects invalid ones.
- `GET /auth/validate/:id` – Confirms user validation.

- `GET /auth/profile` – Requires JWT; returns authenticated user profile.

Users (/user)

- `POST /user/become-seller` – Upgrade authenticated user to seller.
- `GET /user/profile` – Authenticated user fetches their own profile.
- Admin endpoints: `/user/admin/all`, `/user/admin/stats`, `/user/admin/:id` (view/update/delete).

Stores (/store)

- Public: `GET /store/`, `GET /store/:id`.
- Authenticated: `POST /store/`, `GET /store/user/me`, `GET /store/user/:userId`.
- Owner/Admin: `PUT /store/:id`, `DELETE /store/:id`, `GET /store/:id/stats`.
- Admin-only: `GET /store/admin/all`, `GET /store/admin/pending`, `PATCH /store/:id/verify`.

Products (/product)

- Public: `GET /product/`, `GET /product/code/:productCode`, `GET /product/store/:storeId`, `GET /product/:id`.
- User: `GET /product/user/me`.
- Store role required: `POST /product/`, `PUT /product/:id`, `DELETE /product/:id`.
- Admin: `GET /product/admin/all`, `GET /product/admin/stats`, `PATCH /product/admin/:id/status`, `DELETE /product/admin/:id`.

Ownership (/ownership)

- Public: `GET /ownership/product/:productId/current`, `GET /ownership/product/code/:code`
- Authenticated: `GET /ownership/product/:productId`, `GET /ownership/user/:userId`, `GET /ownership/my-products`, `POST /ownership/`, `PUT /ownership/:id/end`, `GET /ownership/product/:productId/history`.

Verification (/verification)

- All routes require authentication.
- `POST /verification`, `PUT /verification/:id/status`.
- `GET /verification/owner`, `GET /verification/product/:productId`, `GET /verification/user/:userId`, `GET /verification/:id`.

Transactions (/transaction)

- Authenticated only.
- `POST /transaction/initiate/:productId`, `POST /transaction/confirm/:transactionId`.
- `POST /transaction/:transactionId/reject`, `GET /transaction/buy-requests`.

Store Members (/store-member)

- Store owner: `GET /store-member/store/:storeId`, `GET /store-member/user/:userId`, `POST /store-member/`, `DELETE /store-member/:storeId/:userId`, `PUT /store-member/:storeId/:userId`.
- Admin: `GET /store-member/`.

Admin (/admin)

- All require authentication and admin role.
- `GET /admin/stats`.

6.2.4 Backend Testing Screenshots

The following figures present screenshots of backend API testing carried out using the REST Client extension in VS Code.

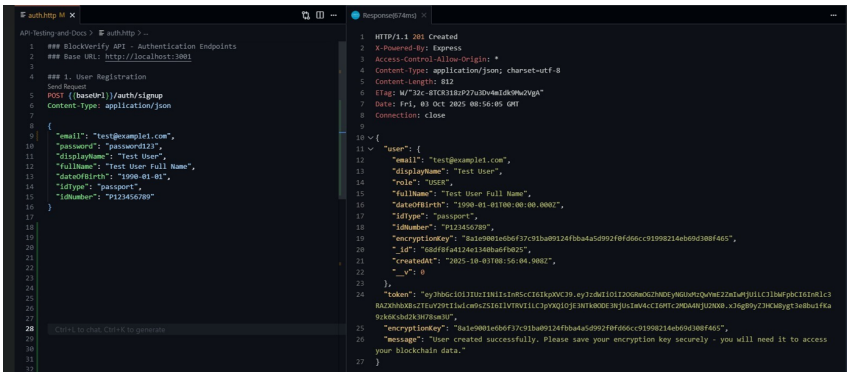

Figure 6.1. Backend-Testing

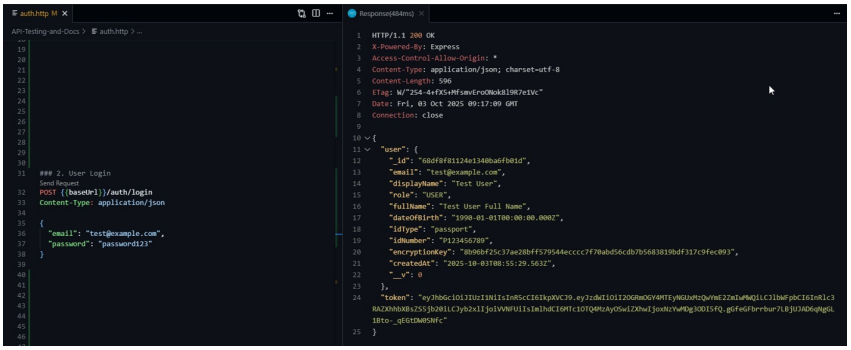

Figure 6.2. Backend-Testing

6.2.5 Frontend Testing with Lighthouse and Responsive Checks

Lighthouse audits were run on representative pages in the Next.js client: authentication, dashboards, product creation, public product search, and request workflows.

Frontend Testing Screenshots The following figures present screenshots from frontend testing carried out on the Next.js client application.

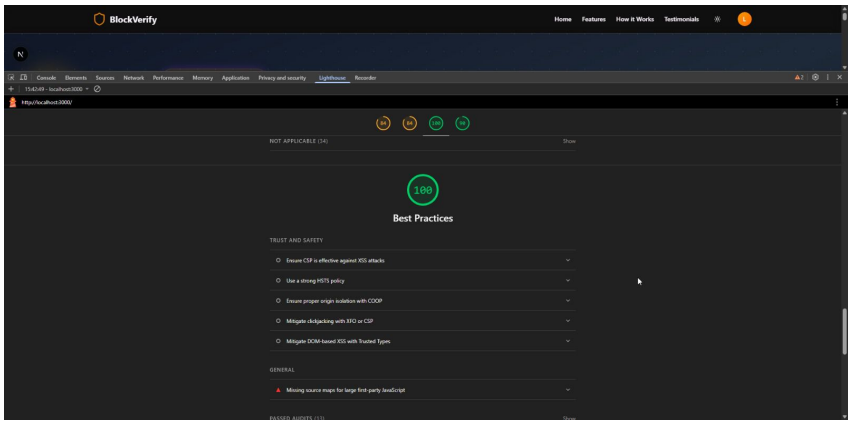

Figure 6.3. Front-end Testing

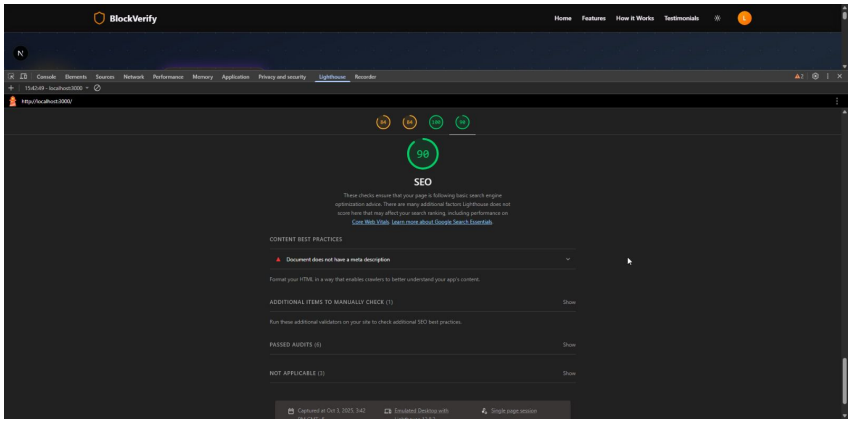

Figure 6.4. Front-end Testing

6.2.6 Test Cases

Backend Test Cases

Table 6.1. Backend Test Cases

Test Case ID	Module	Test Scenario	Input / Action	Expected Output
TC-BE-01	Admin	Admin Login	Login using seeded admin credentials	Admin logs in and accesses dashboard successfully.
TC-BE-02	User Registration	User Signup	Send valid signup data	User account created successfully.
TC-BE-03	Store Registration	Store Signup	Send valid signup data for store account	Store account created; pending admin approval.
TC-BE-04	Store Approval	Admin Approves Store	Admin approves a pending store	Store becomes active and can register products.
TC-BE-05	Authentication	Invalid Login	Enter wrong credentials	Error shown; access denied.
TC-BE-06	Product Registration	Store registers product	Approved store submits product data	Product created with product-Code; ownership encrypted on blockchain.
TC-BE-07	Ownership Transfer	Owner accepts buy request	Owner approves buyer request	Ownership transferred and updated on blockchain.
TC-BE-08	Blockchain Failure	Simulate blockchain error	Disconnect blockchain service	System handles error gracefully; no data loss.

Frontend Test Cases

Table 6.2. Frontend Test Cases

Test Case ID	Module	Test Scenario	Action	Expected Behavior
TC-FE-01	Landing Page	Landing page accessibility	Load application landing page	Landing page displays navigation to login and signup.
TC-FE-02	Authentication	User Signup	Fill user signup form and submit	Signup confirmation shown; user proceeds to login.
TC-FE-03	Authentication	Store Signup	Fill store signup form and submit	Store created and marked pending approval.
TC-FE-04	Dashboard	Role-based access	Login as admin, user, store	Correct dashboard and permissions shown for each role.
TC-FE-05	Product Registration	Add new product	Store fills out product form	Product added and visible in dashboard.
TC-FE-06	Product Search	Search product by code	Enter valid product code	Product details displayed; owner hidden.
TC-FE-07	Purchase Flow	Initiate and confirm purchase	Buyer initiates and owner accepts	Product ownership updated; UI confirms success.
TC-FE-08	Responsive Design	Resize browser	View site on different devices	Layout adapts without breaking alignment.

Summary of Test Results

All primary features of the system were tested and verified. Each role (Admin, Store, User) performed as expected within defined permissions. Ownership data encryption and blockchain interactions were validated successfully, while frontend components passed responsiveness and usability checks.

6.3 Evaluation Summary

Backend. Authentication, product registration with blockchain write, search, verification workflow, purchase/transfer (DB ownership change), and admin status moderation function correctly.
Frontend. Pages render and operate across devices; Lighthouse scores acceptable.

Chapter 7
Conclusions

The **Blockchain-Based Product Authenticity Verification System** was conceived and implemented to address the widespread issues of counterfeit goods and unverifiable provenance. By combining a modern web client with pragmatic blockchain integration and robust backend controls, the system delivers a trustworthy way to register products, verify public details, and manage ownership transitions.
The system's hybrid design,encrypted ownership records on-chain and product/workflow data in MongoDB—balances transparency with privacy. It empowers **ADMIN**, **STORE**, and **USER** roles through focused dashboards and APIs: stores can register authentic products, buyers can search and submit requests, and administrators can moderate statuses to alert the public. Server-side encryption ensures that personally identifiable information is never exposed on-chain, while append-only blockchain records provide an auditable history of encrypted ownership.
Through role-based flows (product registration, public search by `productCode`, verification, purchase & transfer, and admin moderation), the platform delivers practical authenticity checks without leaking sensitive identity data. Its security posture—bcrypt password hashing, JWT-based authentication with role claims, and encryption with per-user keys—further strengthens user trust and operational integrity.
Despite expected limitations typical of early-stage systems—such as reliance on a backend blockchain API and the absence of advanced cryptographic key rotation—the project establishes a solid foundation for authenticity verification.
Ultimately, this system demonstrates how blockchain can be applied responsibly to product authenticity: public enough to build confidence, private enough to protect users, and simple enough to be adopted by real stores and consumers.

7.1 Counterfeit Mitigation via User Search

Users can query products by their `productCode` and view essential details and current status. This enables quick authenticity checks without revealing owner identities. The combination of user metadata and encrypted on-chain ownership records discourages tampering and promotes accountability.

7.2 User-Friendly, Role-Based Design

The Next.js client provides clear role dashboards for `ADMIN`, `STORE`, and `USER`. Stores register products, users initiate verification or purchase requests, and admins review and moderate statuses. A responsive UI ensures accessibility across device sizes.

7.3 Tools for Trust and Workflow

- **Product Registration:** Stores generate a unique `productCode`, persist product metadata, and trigger server-side encryption of initial ownership before submitting the encrypted string to the blockchain.
- **Verification Requests:** Users can request visibility, and owners review and respond through protected endpoints. Responses selectively surface approved details while the on-chain payload remains encrypted.
- **Purchase & Transfer:** Buyers place requests, and upon acceptance, the system appends a new encrypted ownership record on-chain and updates off-chain ownership snapshots.

7.4 Administrator Oversight

Administrators can view products at scale and update product statuses (`ACTIVE`, `RECALLED`, `STOLEN`, `RETIRED`) via protected endpoints. Status changes provide public trust signals without disclosing information from private owners.

7.5 Technical Excellence

Robust Data Architecture

The system uses MongoDB (via Mongoose) for product and workflow data (products, ownerships, transactions, verification requests), indexed by identifiers such as `productCode`. Encrypted ownership strings are submitted to the blockchain through a backend API and stored on-chain in an append-only manner.

7.5.1 Security-by-Design

- **Authentication/Authorization:** JWT with role claims and middleware-enforced RBAC (`ADMIN`, `STORE`, `USER`).
- **Password Safety:** bcrypt hashing.
- **Encryption:** AES, per-user encryption keys generated at signup; encryption performed server-side; no plaintext on-chain.

7.5.2 Frontend Reliability and Accessibility

The Next.js/TypeScript/Tailwind stack delivers a responsive, componentized interface for all roles. User search is optimized for clarity, while protected flows are gated by authentication and role checks.

7.6 A Foundation for Authenticity and Trust

From product intake to ownership transitions, the system forms a coherent authenticity pipeline backed by blockchain immutability and practical privacy controls. It addresses essential needs (proof of registration, visibility upon approval, and ownership transfer) while remaining extensible.

7.7 Table:Main Features

Table 7.1. Main Functional Features of the Proposed System

Feature	**Description**
Product Registration	Stores create products with a unique `productCode`; initial ownership is encrypted server-side and submitted on-chain.
Public Search	Users can search by `productCode` to view product details and status (owner identities hidden).
Verification/Visibility Requests	Users submit requests; owners/admins review; approved details are returned via the backend while on-chain data remains encrypted.
Purchase Requests	Buyers initiate requests; on acceptance, a new encrypted ownership record is appended on-chain and DB snapshots are updated.
Ownership History (DB + Chain)	Off-chain snapshots reflect current/previous owners; on-chain records maintain an encrypted, append-only history.
Admin Status Moderation	Admin updates product status (`ACTIVE`, `RECALLED`, `STOLEN`, `RETIRED`) via protected endpoints.
Role-Based Access Control	JWT authentication with role claims; route-level guards for `ADMIN`, `STORE`, `USER`.
Server-Side Encryption	Encryption with per-user keys; no plaintext on-chain.

Chapter 8
User Manual

Audience: Buyers, store owners, and administrators. **Goal:** Verify authenticity, track ownership, and enable secure transfers.

8.1 Roles and Permissions

- **Buyer (User):** Browse and search products by code, request owner information reveal, view current owner/history, initiate purchases.
- **Store (Seller):** Create a store, register products, respond to reveal and purchase requests, manage store members.
- **Admin:** Verify stores, moderate products, view platform statistics, and handle flags.

8.2 Getting Started

Sign Up: Open the Sign Up page and provide email, password, display name, and identity details.
Log In: Submit your credentials. You will be redirected to a dashboard based on your assigned role.

8.3 Home and Product Discovery

- Open the landing page to view featured items.
- Search by product code to view basic product details.
- View current owner and ownership history timelines; identities may be redacted unless permission is granted.

8.4 Buyer Workflow

- **Request Owner Reveal:** From a product page, submit a reveal request if you do not have disclosure permission. You will be notified upon approval or denial.
- **Initiate Purchase:** Select request to buy. The current owner receives a pending purchase request.
- **Receive Decision:** If accepted, the system records the ownership transfer. Your dashboard will list the product under my products.
- **Verify Ownership:** Check the current owner and ownership history to confirm that the transfer has been successfully recorded.

8.5 Store (Seller) Workflow

- **Create a Store:** Provide store details and wait for admin verification.
- **Add Members:** Grant team members access to help register and manage products.
- **Register a Product:** Enter product details. The system generates a unique product code and records the initial ownership.
- **Handle Reveal Requests:** Approve to grant temporary access to owner information; permissions may be expired later.
- **Manage Purchase Requests:** Accept to transfer ownership or reject the request. Accepted transfers update the product's current owner.

8.6 Admin Workflow

- **Verify Stores:** Review pending store applications and set status to verified or rejected.
- **Moderate Products:** Update statuses such as *Active*, *Recalled*, *Stolen*, or *Retired*. All actions are logged.
- **Review Flags:** Inspect product-related flags raised by users and take appropriate action.
- **View Platform Statistics:** Access platform-level metrics and health indicators.

8.7 Ownership Views

- **Current Owner:** Shows who presently holds the product.
- **History:** Displays a timeline of ownership transfers. Owner identities appear only if permission is granted; otherwise, they remain redacted.

8.8 Verification and Permissions

- **Reveal Requests:** Buyers request access; owners may approve or deny.
- **Permission Expiry:** Owners may revoke previously granted access, hiding sensitive information again.

8.9 Transactions (Purchase and Transfer)

- Buyer submits a purchase request.
- Owner accepts or rejects the request.
- On acceptance, previous ownership ends and new ownership is recorded. Dashboards and product pages update automatically.

8.10 Flags and Support

- **Product Flags:** Report issues such as counterfeits or disputes. Admins review and take necessary action.
- **Contact and Support:** Create a support ticket to receive assistance. Updates are provided directly in the application.

8.11 Security and Privacy

- Product and ownership updates are stored immutably.

- Sensitive owner details are visible only when permission is explicitly granted and can be revoked at any time.
- Role-based access controls ensure users only access information relevant to their role.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] IBM, “IBM Food Trust,” 2021. [Online]. Available: https://www.ibm.com/blockchain/solutions/food-trust

[3] Everledger, “Provenance for diamonds,” 2020. [Online]. Available: https://www.everledger.io/

[4] VeChain Foundation, “VeChain whitepaper,” 2021. [Online]. Available: https://www.vechain.org/

[5] A. Smith, J. Doe, and R. Khan, “Limitations of centralized product authentication,” *Journal of Secure Systems*, vol. 34, no. 2, pp. 123–134, 2022.

[6] E. Androulaki *et al.*, “Hyperledger Fabric: A distributed operating system for permissioned blockchains,” in *Proc. EuroSys*, 2018, pp. 1–15, doi: 10.1145/3190508.3190538.

[7] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger (yellow paper),” 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[8] G. Zyskind, O. Nathan, and A. Pentland, “Decentralizing privacy: Using blockchain to protect personal data,” in *Proc. IEEE Security and Privacy Workshops*, 2015, pp. 180–184, doi: 10.1109/SPW.2015.27.

[9] S. Saberi *et al.*, “Blockchain technology and its relationships to sustainable supply chain management,” *Int. J. Production Research*, vol. 57, no. 7, pp. 2117–2135, 2019.

[10] N. Kshetri, “Blockchain’s roles in meeting key supply chain management objectives,” *Int. J. Information Management*, vol. 39, pp. 80–89, 2018.

[11] F. Tian, “An agri-food supply chain traceability system for China based on RFID and blockchain technology,” in *Proc. ICSSSM*, 2016, pp. 1–6.

[12] F. Casino *et al.*, “A systematic literature review of blockchain-based applications,” *Telematics and Informatics*, vol. 36, pp. 55–81, 2019.

[13] K. Francisco and D. Swanson, “The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency,” *Logistics*, vol. 2, no. 1, pp. 1–13, 2018.

[14] W3C, “Decentralized identifiers (DIDs) v1.0,” 2022. [Online]. Available: https://www.w3.org/TR/did-core/

[15] W3C, "Verifiable credentials data model 1.1," 2022. [Online]. Available: https://www.w3.org/TR/vc-data-model/